POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Autorzy

Piotr Kryszkiewicz, s20908
Karol Sobotka, s20439
Patryk Kępisty, s21461
Tomasz Pyrzak, s21670
Tomasz Fiedoruk, s14714
Mateusz Górski, s20226
Marcin Hojarski, s12881
Natalia Sawicka, s18288
Krzysztof Sidor, s12513
Kamil Kuryłek, s20242
Krzysztof Maj, s18533

E-montażysta
Platforma do zarządzania firmą montażową

Praca inżynierska

dr. inż. Mariusza Trzaski

Warszawa, lipiec 2023

Streszczenie

Celem pracy jest stworzenie innowacyjnego oprogramowania, które stanowiłoby narzędzie wspomagające firmy zajmujące się montażem oraz zarządzaniem własnymi magazynami. Projekt został stworzony, aby usprawnić procesy zarządzania firmą oraz umożliwić łatwiejszą koordynację działań między pracownikami. Aplikacja została zrealizowana w architekturze trójwarstwowej, wykorzystując technologie Spring Boot oraz React, natomiast do przechowywania danych została wykorzystana baza PostgreSQL. W celu zwiększenia efektywności pracy zespołu, skorzystaliśmy z metodyki Agile. Aby zapewnić wysoką jakość oprogramowania, projekt uwzględnił zaprojektowanie i wdrożenie procesu testowania. Dodatkowo, wszystkie komponenty systemu zostały skonteneryzowane, co umożliwiło łatwą migrację do chmury.

Słowa kluczowe

Java, Swagger, Lombok, Rest API, JUnit, Rest-assured, Selenium, Figma, Draw.io ClickUp, Spring Boot, Typescript, React, Kotlin, PostgreSQL, CSS, JavaScript, IntelliJ, Android Studio, Kubernetes, Postman, Adnroid, aplikacja webowa, aplikacja mobilna, aplikacja dla montażystów, zarządzanie montażystami, aplikacja do kontaktu magazynu z montażystami, zarządzanie pracownikami, zarządzanie magazynem, zarządzanie realizacjami, webowa aplikacja dla montażystów, mobilna aplikacja dla montażystów

Spis treści

1. Wstęp

Każde przedsiębiorstwo, które chce stwarzać konkurencje na rynku, musi zdawać sobie sprawę z potrzeby nadążania za dynamicznie rozwijającą się technologią. Dokładnie w takiej samej sytuacji znajdują się firmy zajmujące się montażem oraz zarządzaniem własnymi magazynami. Wiele z nich wciąż boryka się z problemami spowodowanymi przez ręczne zarządzanie zleceniami czy korzystanie z niedopasowanych do swoich potrzeb rozwiązań. Skutkiem czego może być między innymi:

- Niska wydajność, spowodowana niewłaściwym zarządzaniem zasobami firmy;
- Wysoka podatność na błąd ludzki ze względu na brak odpowiednich mechanizmów zapobiegających;
- Brak elastyczności i szybkiego dostosowywania się do zmieniającego się rynku;
- Utrata lub wyciek danych krytycznych prowadzący do poważnych strat biznesowych;

W ramach niniejszej pracy przeprowadzono analizę wymagań wyżej wymienionej niszy związanej z branżą montażową i zarządzaniem magazynami. Zidentyfikowanie najważniejszych problemów umożliwiło określenie wymagań dotyczących, zaprojektowania i zaimplementowania nowego oprogramowania odpowiadającego na potrzeby branży.

1.1. Cel

Celem projektu było stworzenie rozwiązania w oparciu o założenia metodologii ERP dedykowane małym i średnim firmom. System ma za zadanie usprawnić sposób zarządzania pracownikami oraz zasobami przedsiębiorstwa w celu optymalizacji pracy. Projekt zakładał również usprawnienie zarządzaniem ewidencją posiadanych narzędzi, serwis oraz zgłaszanie usterek. W ramach projektu zaprojektowano i stworzono dedykowane narzędzie do komunikacji między pracownikami oraz bezpieczne miejsce do przechowywania dokumentacji i materiałów związanych z realizacją zleceń.

Projekt został zrealizowany w ramach specjalizacji Inżynierii Oprogramowania i Baz Danych na Polsko-Japońskiej Akademii Technik Komputerowych.

1.2. Zakres prac

Praca nad projektem została podzielona na pięć zespołów, w skład których wchodziło łącznie jedenaście osób. Każdy zespół był odpowiedzialny za określony obszar projektu:

1. Analizy;
2. Projektowania;
3. Implementacji;
4. Testów;
5. Dokumentacji;

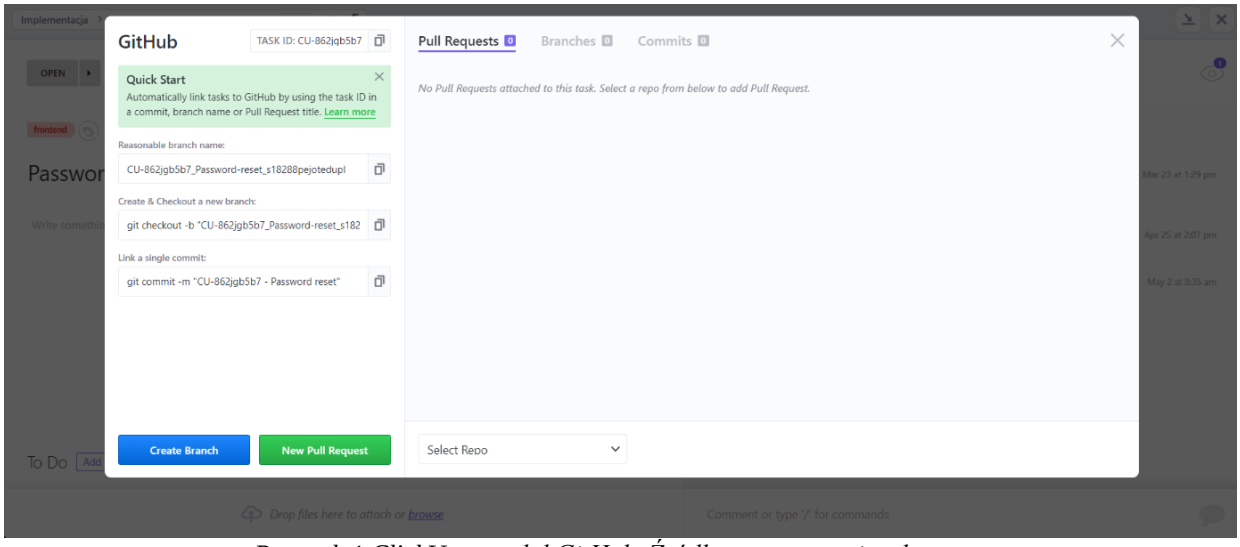
Nad wszystkim czuwał kierownik projektu. Z racji obszerności pracy, wszyscy członkowie należeli do minimum 2 zespołów. Wstępne planowanie miało miejsce na spotkaniach projektowych, jednak potrzebne było narzędzie do komunikacji poza uczelnią. Z pomocą narzędzia ClickUp zespół był w stanie komunikować się płynnie i bez przeszkód w metodyce zwinnej. Każdy z zespołów miał oddzielny folder, w którym mógł umieszczać swoje zadania, dodatkowo były oznaczana one specjalnymi etykietami.

Dzięki podziałowi na poniższe kategorie, było możliwe monitorowanie postępu prac:

- Open;
- In Progress;
- Review;
- For Tests;
- Closed;

Tygodniowe sprinty oraz cotygodniowe spotkania przyspieszyły pracę i ułatwiły komunikację między zespołami.

Dodatkowo, aplikacja ClickUp umożliwia integrację z GitHub. Największym atutem tego rozwiązania było automatyczne tworzenie nowych gałęzi, co przedstawia Rysunek 1.



Rysunek 1 ClickUp - moduł GitHub. Źródło: opracowanie własne.

1.3. Organizacja pracy

Dokument ten przedstawia proces tworzenia aplikacji oraz ma za zadanie udokumentowanie całości działań zespołu. Rozdział drugi przedstawia istniejącą konkurencje na rynku polskim i zagranicznym. Rozdział trzeci zawiera szczegółową specyfikację systemu wraz z opisem funkcjonalności, wymaganiami funkcjonalnymi i niefunkcjonalnymi, projektem aplikacji, diagramami przypadków użycia, scenariuszami, diagramem klas. Rozdział czwarty opisuje wykorzystane technologie do stworzenia oprogramowania. Piąty rozdział szczegółowo omawia proces projektowania i implementacji. Szósty rozdział dotyczy interfejsu użytkownika, natomiast siódmy poświęcony jest testom funkcjonalnym. Rozdział ósmy poświęcony został wnioskom i omówieniu projektu po procesie wytwórczym.

1.4. Struktura organizacyjna zespołu

Tabela 1 przedstawia skład oraz podział członków zespołu projektu wraz z podziałem zadań.

Tabela 1 Podział podzespołów i przydział zadań. Źródło: opracowanie własne.

Kierownik projektu E-montażysta
Krzysztof Sidor

Zespół Analizy	
Kierownik	Marcin Hojarski
Członkowie zespołu	Natalia Sawicka Tomasz Fiedoruk Patryk Kępisty Mateusz Górski
Podział zadań	
Stan sztuki	Marcin Hojarski Natalia Sawicka Tomasz Fiedoruk Patryk Kępisty Mateusz Górski
Funkcjonalności systemu	Marcin Hojarski Natalia Sawicka Tomasz Fiedoruk Patryk Kępisty Mateusz Górski
Przypadki użycia	Marcin Hojarski Natalia Sawicka Tomasz Fiedoruk Patryk Kępisty Mateusz Górski
Analityczny diagram klas	Marcin Hojarski Natalia Sawicka Tomasz Fiedoruk Patryk Kępisty Mateusz Górski
Słownik	Tomasz Fiedoruk Mateusz Górski

Zespół projektowy	
Kierownik	Kamil Kuryłek
Członkowie zespołu	Natalia Sawicka Krzysztof Sidor Patryk Kępisty Tomasz Pyrzak
Podział zadań	
Projektowy diagram klas	Kamil Kuryłek Natalia Sawicka

	Krzysztof Sidor Patryk Kępisty Tomasz Pyrzak
Mapa strony	Kamil Kuryłek Natalia Sawicka Krzysztof Sidor Patryk Kępisty Tomasz Pyrzak
Scenariusze	Kamil Kuryłek Natalia Sawicka Krzysztof Sidor Patryk Kępisty Tomasz Pyrzak
Projektowanie ekranów aplikacji webowej	Patryk Kępisty
Projektowanie aplikacji mobilnej	Krzysztof Sidor
Logo aplikacji	Krzysztof Sidor

Zespół implementacji warstwy serwerowej	
Kierownik	Kamil Kuryłek
Członkowie zespołu	Piotr Kryszkiewicz Krzysztof Sidor Tomasz Pyrzak Karol Sobotka
Podział zadań	
Szkolenie Spring i Postman	Mateusz Górski Tomasz Pyrzak
Szkolenie z dobrych praktyk programowania	Kamil Kuryłek
Utworzenie klas modelowych	Karol Sobotka Piotr Kryszkiewicz Kamil Kuryłek Tomasz Pyrzak
Utworzenie obiektów transferu danych (DTO)	Piotr Kryszkiewicz Kamil Kuryłek Tomasz Pyrzak
Utworzenie warstwy serwisowej	Piotr Kryszkiewicz Kamil Kuryłek Tomasz Pyrzak
Konfiguracja CORS	Krzysztof Sidor
Konfiguracja Swaggera	Piotr Kryszkiewicz
Implementacja systemu autoryzacji i autentykacji	Tomasz Pyrzak Piotr Kryszkiewicz
Implementacja walidacji klas modelowych	Piotr Kryszkiewicz Kamil Kuryłek Tomasz Pyrzak
Implementacja obsługi błędów - zastosowanie paradygmatu programowania aspektowego	Piotr Kryszkiewicz Tomasz Pyrzak
Utworzenie adnotacji walidacyjnych	Piotr Kryszkiewicz

Konfiguracja połączenia z bazą danych	Tomasz Pyrzak
Konfiguracja połączenia z serwerem SMTP	Krzysztof Sidor Kamil Kuryłek
Seedowanie danych testowych	Kamil Kuryłek Tomasz Pyrzak
Implementacja filtrowania przez Criteria API	Tomasz Pyrzak Kamil Kuryłek
Integracja projektu z Hibernate	Tomasz Pyrzak Piotr Kryszkiewicz
Implementacja przechowywania załączników	Piotr Kryszkiewicz
Implementacja systemu resetowania hasła	Piotr Kryszkiewicz
Implementacja usuwania nietrwałego	Piotr Kryszkiewicz
Zespół implementacji warstwy webowej	
Kierownik	Patryk Kępisty
Członkowie zespołu	Natalia Sawicka Kamil Kuryłek Marcin Hojarski
Podział zadań	
Szkolenie React	Tomasz Fiedoruk
Implementacja widoków	Natalia Sawicka Kamil Kuryłek Patryk Kępisty Marcin Hojarski
Implementacja modeli danych	Natalia Sawicka Kamil Kuryłek Patryk Kępisty Marcin Hojarski
Implementacja połączenia z warstwą serwerową	Natalia Sawicka Kamil Kuryłek Patryk Kępisty Marcin Hojarski
Implementacja filtrowania i paginacji tabel	Patryk Kępisty
Wykonanie stron błędów	Patryk Kępisty
Implementacja komponentu kalendarza	Patryk Kępisty
Implementacja komponentu lokalizacji – integracja z Nominatim API	Patryk Kępisty
Implementacja komponentu szczegółów zlecenia	Marcin Hojarski
Implementacja wyświetlania komunikatów	Patryk Kępisty Kamil Kuryłek
Implementacja responsywnego interfejsu graficznego z wykorzystaniem media query	Patryk Kępisty
Implementacja nawigacji	Patryk Kępisty
Implementacja systemu dostępu do widoków	Patryk Kępisty Kamil Kuryłek Marcin Hojarski
Implementacja widoku powiadomień	Kamil Kuryłek
Zespół implementacji aplikacji mobilnej	

Kierownik	Krzysztof Sidor
Członkowie zespołu	Krzysztof Sidor Krzysztof Maj
Podział zadań	
Implementacja logowania	Krzysztof Sidor
Implementacja widoków	Krzysztof Sidor Krzysztof Maj
Implementacja modeli danych	Krzysztof Sidor Krzysztof Maj
Implementacja połączenia z warstwą serwerową	Krzysztof Sidor Krzysztof Maj
Implementacja filtrowania	Krzysztof Sidor
Implementacja skanowania kodów QR	Krzysztof Sidor
Projekt interfejsu graficznego	Krzysztof Sidor
Zespół implementacji rozwiązań DevOps	
Kierownik	Krzysztof Sidor
Członkowie zespołu	Natalia Sawicka
Szkolenie Docker	Natalia Sawicka Krzysztof Sidor
Analiza zapotrzebowania projektu	Krzysztof Sidor Natalia Sawicka
Konfiguracja orkiestratora	Krzysztof Sidor Natalia Sawicka
Konfiguracja narzędzia Traefik	Krzysztof Sidor Natalia Sawicka
Konfiguracja środowisk	Krzysztof Sidor
Implementacja Github Workflow	Krzysztof Sidor
Utworzenie manifestu Dockerfile	Krzysztof Sidor Natalia Sawicka
Utworzenie manifestu docker-compose	Krzysztof Sidor Natalia Sawicka
Utworzenie manifestów Kubernetes	Krzysztof Sidor Natalia Sawicka
Konfiguracja narzędzia Portainer	Krzysztof Sidor

Zespół testów	
Kierownik	Karol Sobotka
Członkowie zespołu	Piotr Kryszkiewicz Tomasz Fiedoruk Mateusz Górski
Podział zadań	
Szkolenie Testy jednostkowe	Piotr Kryszkiewicz
Szkolenie REST-ASSURED	Karol Sobotka

Szkolenie Selenium	Karol Sobotka
Szkolenie Postman	Karol Sobotka
Pisanie Scenariuszy Testowych	Karol Sobotka Krzysztof Maj Matusz Górski Piotr Kryszkiewicz Tomasz Fiedoruk
Testy manualne	Krzysztof Maj Matusz Górski Piotr Kryszkiewicz Tomasz Fiedoruk
Testy API Manualne	Karol Sobotka Krzysztof Maj Matusz Górski Piotr Kryszkiewicz Tomasz Fiedoruk
Testy API z wykorzystaniem REST-ASSURED	Karol Sobotka Krzysztof Maj Matusz Górski Piotr Kryszkiewicz Tomasz Fiedoruk
Testy automatyczne z wykorzystaniem SELENIUM	Karol Sobotka Krzysztof Maj Matusz Górski Tomasz Fiedoruk
Testy manualne aplikacji mobilnej	Tomasz Fiedoruk

Zespół dokumentacji	
Kierownik	Natalia Sawicka
Członkowie zespołu	Tomasz Fiedoruk Mateusz Górski Krzysztof Sidor
Podział zadań	
Wstęp	Natalia Sawicka
Rozdział 2	Tomasz Fiedoruk
Rozdział 3	Natalia Sawicka
Rozdział 4	Natalia Sawicka Tomasz Fiedoruk Mateusz Górski
Rozdział 5	Natalia Sawicka Tomasz Fiedoruk Mateusz Górski Krzysztof Sidor
Rozdział 6	Mateusz Górski Natalia Sawicka
Rozdział 7	Mateusz Górski
Podsumowanie i wnioski	Natalia Sawicka

1.5. Wprowadzenie do problematyki

Każda firma produkcyjna, która posiada magazyn oraz zespoły montażowe, niezależnie czy są to wewnętrzni pracownicy, czy zewnętrzni podwykonawcy, ma swoje unikalne podejście do zarządzania i komunikacji. Ze względu na specyfikę branży, w której sytuacje mogą zmieniać się w bardzo krótkim czasie, często zaniedbuje się znaczenie wewnętrznej komunikacji, a tym bardziej specjalistycznego oprogramowania.

Najczęściej stosowane rozwiązania ad hoc generują wiele problemów, takich jak brak zabezpieczenia danych, rozproszenie w wielu miejscach i brak jakichkolwiek zabezpieczeń - to tylko wierzchołek góry lodowej.

W związku z tym oprogramowanie opracowywane, w ramach projektu mogłoby rozwiązać wiele, jeżeli nie wszystkie problemy firm zajmujących się tematyką montażu. Dedykowana przestrzeń umożliwiająca zarządzanie plikami i komunikację, zabezpieczona i dostosowana do specyficznych wymagań branży, wraz z modułami personalizowanymi pod potrzeby konkretnego klienta, sprawia, że projekt ten jest unikatowy i może odnieść duży sukces na rynku, wypełniając istniejącą lukę w ofercie.

2. Istniejące systemy dla firm zajmujących się montażem oraz zarządzaniem własnymi magazynami

Coraz więcej przedsiębiorców zauważa, że dobrze zorganizowany proces zarządzania jest niezwykle istotny przy prowadzeniu własnej firmy. Dlatego popyt na wykorzystanie specjalnych systemów informatycznych, które pozwalają na bardziej efektywne nadzorowanie prac zwiększa się z roku na rok.

W tym rozdziale zostaną omówione istniejące platformy dla firm zajmujących się montażem oraz zarządzaniem własnymi magazynami. Przedstawimy najpopularniejsze rozwiązania, ich zalety oraz wady, a także sposoby, w jaki pomagają zwiększyć efektywność pracy.

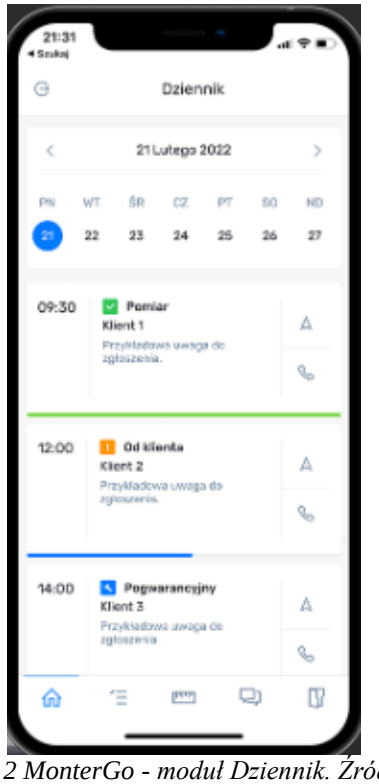
2.1. MonterGo - oprogramowanie przeznaczone głównie dla firm z branży budowlanej i usługowej.

Aplikacja MonterGo jest oferowana przez firmę MonterGO.com Sp. z.o.o. [2]. Oprogramowanie przeznaczone jest dla firm z branży budowlanej i usługowej. Najważniejsze możliwości systemu obejmują:

- Śledzenie statusu realizacji zleceń;
- Cyfrową kartę licznikową;
- Prowadzenie bazy klientów i pracowników;
- Bibliotekę dokumentów;
- Wewnętrzny czat;
- Mobilny protokół odbioru z możliwością zdalnego podpisywania na tablecie lub smartfonie;

2.1.1 Moduł Dziennik

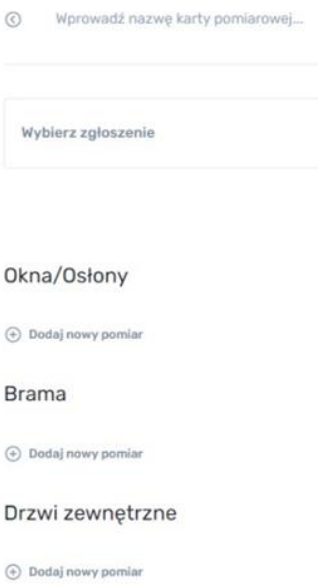
Ponadto w wybranej aplikacji MonterGo znajdziemy wiele modułów, które znacząco wyróżniły tę aplikację na tle innych. Pierwszym z nich jest moduł Dziennik, który pozwala na lepszą organizację pracy zespołowej poprzez harmonogram pracy. Logując się, użytkownicy mogą przeglądać zadania z bieżącego dnia i dodawać nowe aplikacje. Użytkownicy mogą nawigować do różnych dni i przeglądać zadania, a mini kalendarz po prawej stronie pokazuje zaplanowane aplikacje z kropkami. Na rysunku 2 przedstawiono widok modułu Dziennik.



Rysunek 2 MonterGo - moduł Dziennik. Źródło: [1]

2.1.2 Moduł Cyfrowa Karta Pomiarowa

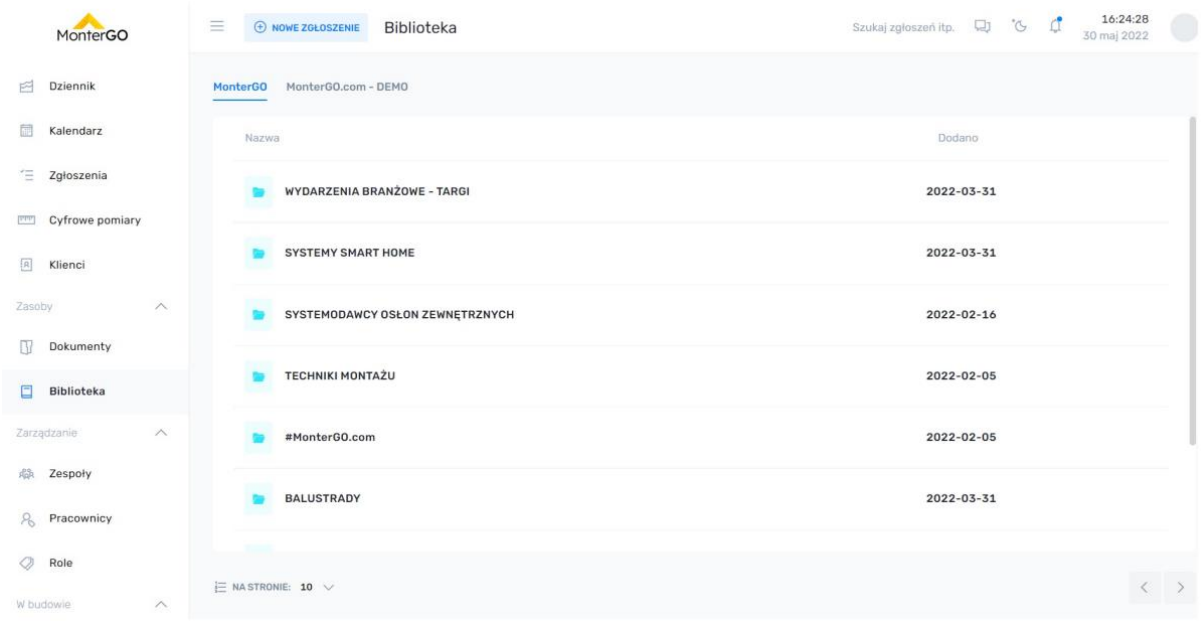
Kolejny ważny moduł to Cyfrowa Karta Pomiarowa, która umożliwia użytkownikom wprowadzanie pomiarów w terenie. Zawiera wszystkie Karty Pomiarowe wygenerowane na podstawie wprowadzonych danych. Klikając na plik, użytkownicy mogą przeglądać Kartę Pomiarową zawierającą dane takie jak data i godzina pomiaru, numer wniosku, dane klienta, wymiary/specyfikacje wraz z rysunkami. Na rysunku 3 widoczny jest widok modułu Cyfrowa Karta Pomiarowa.



Rysunek 3 Aplikacja MonterGo - moduł Cyfrowa Karta Pomiarowa Źródło: [2]

2.1.3 Moduł Biblioteka

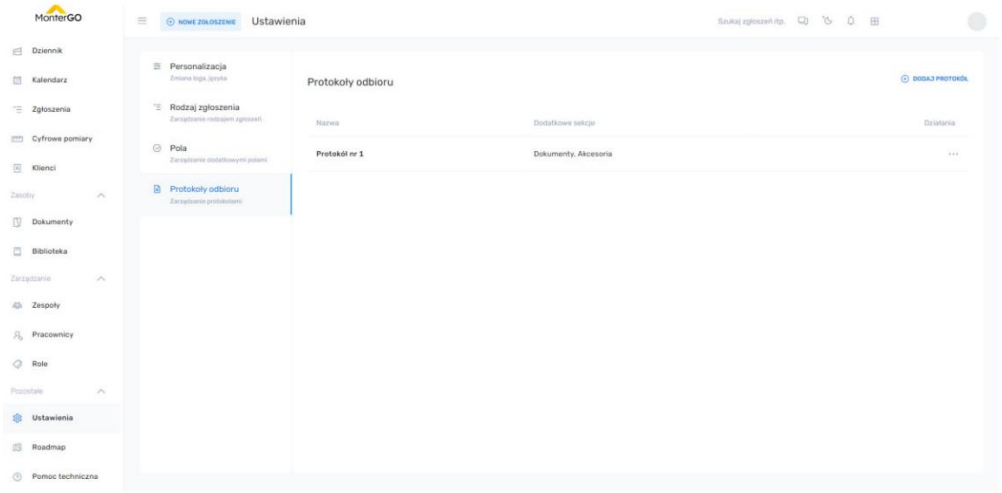
Moduł Biblioteka zapewnia szybki i łatwy dostęp do bazy dokumentów, takich jak wzory umów, karty gwarancyjne czy instrukcje montażu. Biblioteka podzielona jest na materiały dostarczone przez MonterGO (nieedytowalne) oraz własne (edytowalne) z nazwą firmy. Rysunek 4 przedstawia widok modułu Biblioteka.



Rysunek 4 Aplikacja MonterGo - moduł Biblioteka. *Źródło: [2]*

2.1.4 Moduł Mobilny protokół odbioru

Moduł Mobilny protokół odbioru umożliwia podpisanie protokołu na tablecie lub telefonie. Na karcie Konfiguracja - Protokoły akceptacji użytkownicy mogą tworzyć szablony protokołów, takie jak Protokół akceptacji okna lub Protokół usługi. Każdy protokół może mieć nazwę i adnotację w polu „Odbiorca oświadcza, że:", a użytkownicy mogą dodawać sekcje i elementy do każdego protokołu. Pracownicy mogą uzupełniać protokoły, zaznaczając pola wyboru obok pozycji. Widok modułu mobilnego protokołu odbioru przedstawia rysunek 5.



Rysunek 5. Aplikacja MonterGo - moduł Mobilny protokół odbioru. Źródło: [2]

Z pozostałych funkcjonalności aplikacji MonterGo możemy wyróżnić to, że system umożliwia wygodne przekazywanie dokumentów pracownikom oraz tworzenie zespołów i dodawanie nowych pracowników. Dostępne są również funkcje zlecania zadań oraz zarządzania uprawnieniami w zależności od stanowiska. Aplikacja MonterGo oferuje możliwość generowania plików PDF na podstawie dokonanych pomiarów oraz dostęp do bazy gotowych rysunków poglądowych, a także umożliwia rysowanie własnych szkiców lub załączanie plików. Pracownicy mogą korzystać z systemu w trybie offline, a także tworzyć indywidualne szablony i podpisywać dokumenty na urządzeniach mobilnych.

Dla większej wygody użytkowników, aplikacja MonterGo oferuje również możliwość wyboru wersji kolorystycznej. Posiada wersję webową dla pracowników biurowych oraz wersję mobilną dla pracowników w terenie. Logowanie do systemu jest możliwe bez konieczności instalacji oprogramowania.

2.2. Reveal Field - oprogramowanie do zarządzania pracownikami w terenie

Reveal Field **Błąd! Nie można odnaleźć źródła odwołania.** to oprogramowanie przeznaczone d o zarządzania pracownikami w terenie, które pomaga firmom skomputeryzować procesy pracy, eliminując potrzebę posiadania papieru. Umożliwiając pracownikom logowanie się do tego samego systemu, sprawniejszą komunikację i realizację zadań, które wcześniej były wykonywane ręcznie. Na przykład przewoźnicy mogą wprowadzać do systemu dane dotyczące zamówień, co zwiększa produktywność i oszczędza czas, eliminując konieczność telefonowania do pracowników w celu zmiany zamówień. Prowadzi to do mniejszej liczby przerw i umożliwia klientom śledzenie statusu ich zamówień, co zwiększa ich satysfakcję.

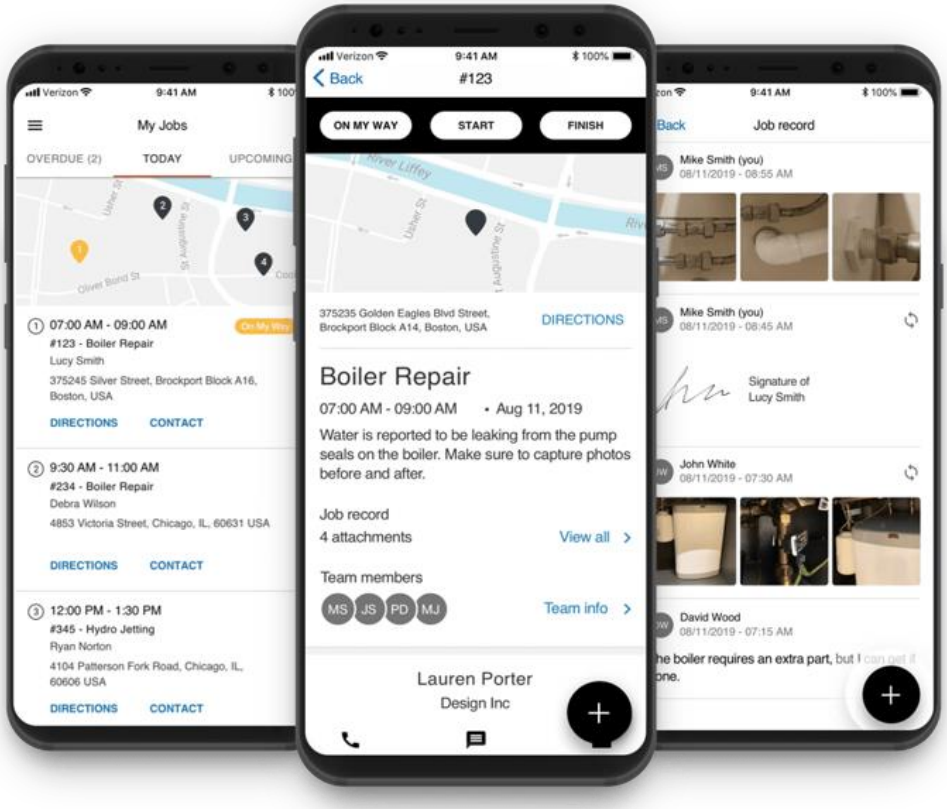
Dostarczając przypomnienia i potwierdzenia SMS-em i e-mailem, Reveal Field pomaga firmom dostarczać szybkie, spersonalizowane usługi, które budują pozytywną reputację wśród klientów. System pozwala również firmom podawać dokładniejsze okna czasowe przyjazdów pracowników. Dostępna w oprogramowaniu mapa na żywo i narzędzie do planowania pozwalają również szybko znaleźć najbliższego pracownika mobilnego i sprawdzić jego godziny pracy pod kątem pilnych zamówień, co prowadzi do lepszej obsługi klienta.

Z kluczowych usług oferowanych przez aplikację można wyróżnić:

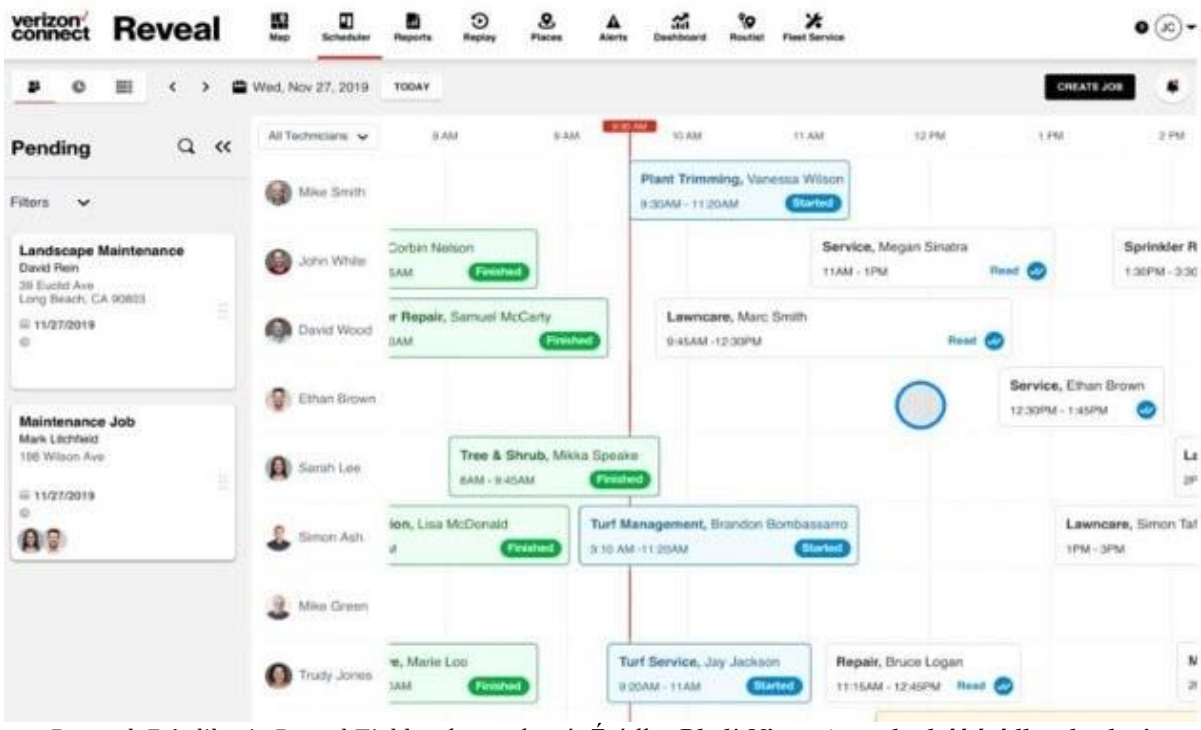
- Pełne monitorowanie - aplikacja ma na celu zapewnienie pełnego wglądu w statusy zadań podczas obsługi zamówień terenowych przez pracowników mobilnych. Zapewnia również system pojedynczego logowania dla wszystkich użytkowników, eliminując potrzebę synchronizacji oddzielnych kalendarzy i arkuszy kalkulacyjnych.
- Uproszczone planowanie - oprogramowanie ułatwia planowanie dzięki łatwym w użyciu kalendarzom i śledzeniu czasu, umożliwiając technikom serwisu przydzielanie zadań. Inteligentny harmonogram zarekomenduje pracownika na podstawie dostępności lokalizacji i wymaganych umiejętności, jeśli termin wykonania zadania ulegnie zmianie.
- Usprawniona komunikacja z pracownikami terenowymi - użytkownik może przydzielać zlecenia bezpośrednio pracownikom i umożliwiać im samodzielną zmianę statusu zlecenia. Mogą sprawdzać swoje plany, zbierać podpisy, zdjęcia i notatki za pomocą aplikacji mobilnej.
- Poprawiona obsługa klienta - system powiadamia Klienta, jeśli pracownik spóźnia się lub nie zapoznał się z przydzielonym zleceniem. Umożliwia dokładniejsze określanie ETA (estimated time of arrival – szacowany czas przybycia) dla techników serwisowych na miejscu za pomocą monitorowania GPS.

- Scentralizowane zarządzanie zadaniami - oprogramowanie umożliwia zarządzanie wszystkimi zadaniami/pracami w terenie z jednego miejsca, zapewniając jedno centrum zarządzania do sprawdzania statusu zadań, planowania zadań i monitorowania pojazdów w czasie rzeczywistym. Zapewnia również widok mapy i kalendarza na żywo, aby pomóc w pilnych i priorytetowych zadaniach.

Rysunek 6, prezentujący widoki zgłoszenia, lokalizacji i protokołu, oraz rysunek 7, obrazujący planer zleceń, są integralnymi częściami aplikacji Reveal Field. Dzięki nim użytkownicy mają możliwość monitorowania postępu wykonywania zadań, jak również precyzyjnego zarządzania pracownikami i lokalizacjami, w których prace są wykonywane. Planer zleceń umożliwia przeglądanie i zarządzanie zaplanowanymi zleceniami oraz przypisywanie ich do odpowiednich pracowników.



Rysunek 6 Aplikacja Reveal Field - widoki. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***



Rysunek 7 Aplikacja Reveal Field - planer zleceń. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***

Aplikacja Reveal Field posiada jeszcze szereg dodatkowych funkcjonalności, wśród których znajdują się m.in. monitorowanie statusu pracowników oraz potwierdzenie wykonania usługi, co pozwala na efektywną kontrolę nad przebiegiem prac. Dla klientów dostępne są specjalne powiadomienia informujące o postępach w wykonywaniu zlecenia, a także możliwość wystawiania ocen i recenzji, które są dostępne w specjalnym panelu.

Ważną funkcjonalnością aplikacji jest także planowanie obsługi wieloetapowych projektów, co pozwala na łatwe zarządzanie projektami składającymi się z wielu zadań i pracowników. Dzięki temu użytkownicy aplikacji mają pełną kontrolę nad przebiegiem prac oraz możliwość efektywnego zarządzania projektem od jego początku aż do zakończenia.

2.3. Zoho Inventory - oprogramowanie do zarządzania stanami magazynowymi

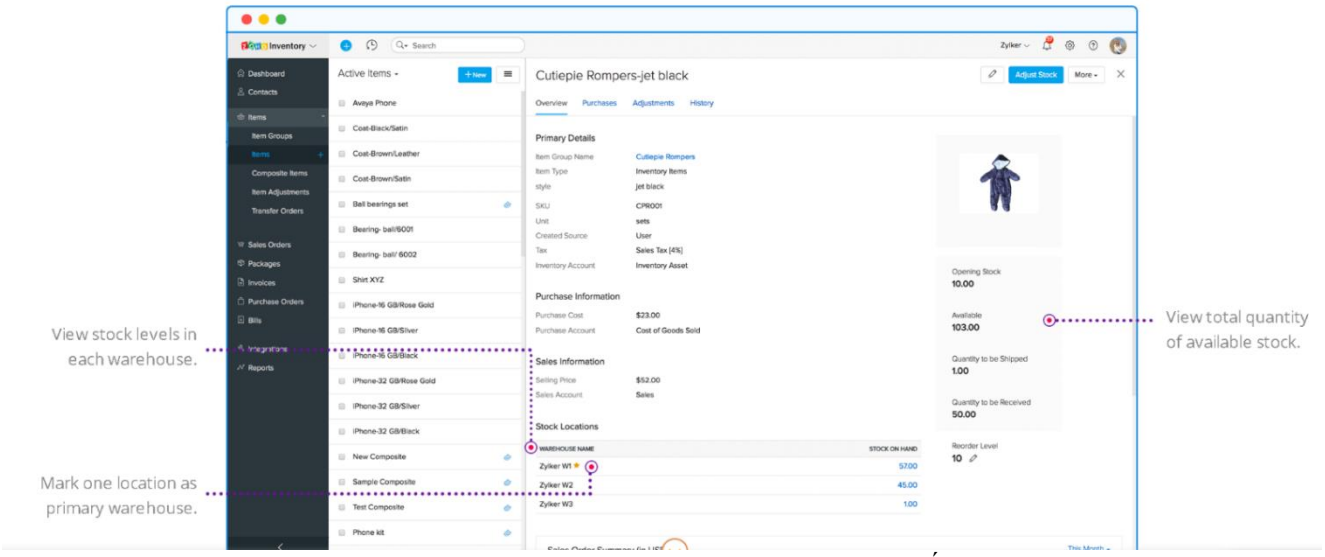
Zoho Inventory [3] to oparte na chmurze oprogramowanie do zarządzania stanami magazynowymi zaprojektowane, aby pomóc firmom zarządzać zapasami, sprzedażą, zakupami i wysyłką. Zapewnia funkcje, takie jak śledzenie zapasów w czasie rzeczywistym, wielokanałowe zarządzanie sprzedażą, zarządzanie zamówieniami, zarządzanie dostawcami oraz zarządzanie wysyłką i realizacją. Użytkownicy mogą łatwo integrować swoje kanały sprzedaży online, takie jak Amazon, eBay i Shopify, oraz synchronizować poziomy zapasów we wszystkich kanałach. Ponadto Zoho Inventory oferuje potężne możliwości raportowania i analizy, a także integracje z innymi produktami Zoho, takimi jak Zoho Books do księgowości i Zoho CRM do zarządzania klientami.

Do głównych funkcji zarządzania magazynem w Zoho Inventory należy:

- Śledzenie i zarządzanie zapasami;
- Kontrola poziomu zapasów;
- Zarządzanie zamówieniami zakupu;
- Zarządzanie zamówieniami sprzedaży;
- Skanowanie kodów kreskowych;

- Przesunięcia magazynowe;
- Raportowanie;

Dzięki śledzeniu i zarządzaniu zapasami użytkownicy mogą sprawdzać ich poziomy i monitorować ruch w wielu magazynach w czasie rzeczywistym. System zapewnia również alerty o niskim poziomie, umożliwiając użytkownikom ich uzupełnianie w razie potrzeby. Rysunek numer 8 przedstawia przykładowy widok tego modułu.

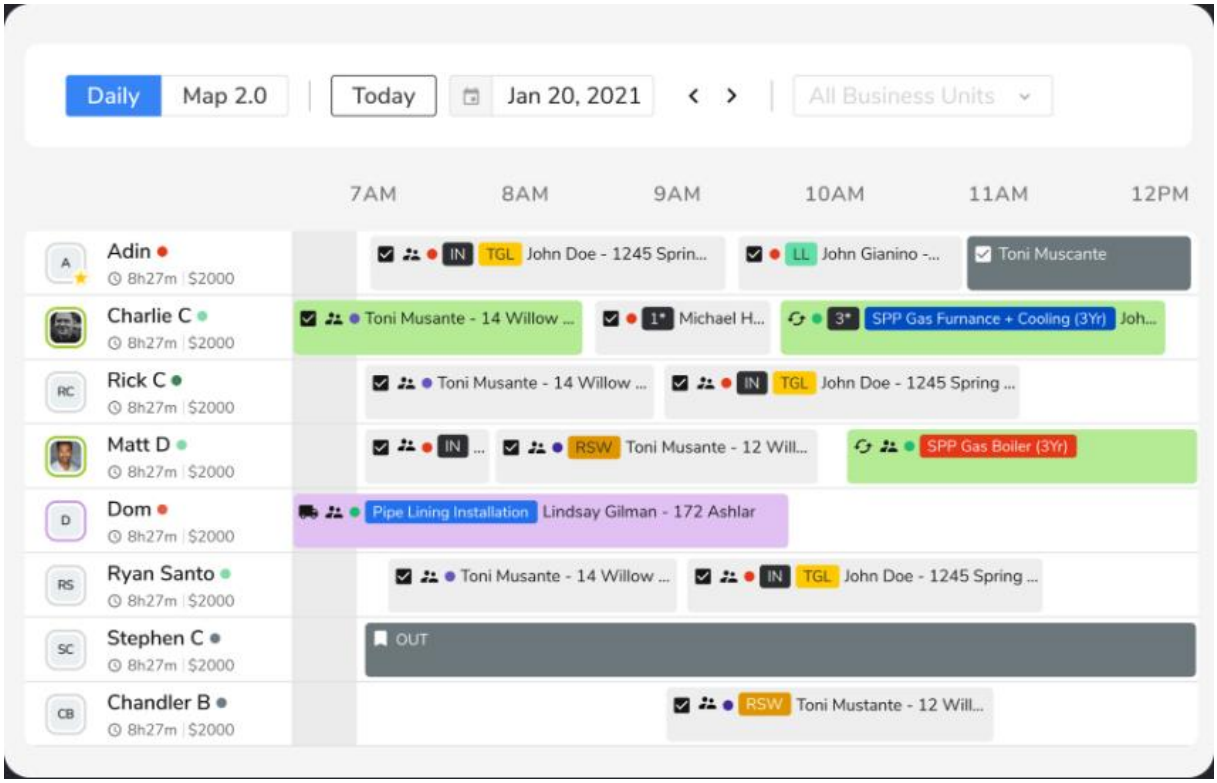


Rysunek 8 Aplikacja Zoho Inventory - moduł Zarządzanie magazynem Źródło: [3]

2.4. ServiceTitan

ServiceTitan [10] to kompleksowe oprogramowanie do zarządzania usługami w terenie, zaprojektowane specjalnie dla branży usług domowych. Oferuje szeroką gamę funkcji, które pomagają firmom zajmującym się instalacjami grzewczymi, wentylacją, klimatyzacją, hydrauliką, elektryką usprawnić ich działalność, zarządzać personelem i poprawić zadowolenie klientów. Główne cechy ServiceTitan obejmują:

1. Planowanie i wysyłanie: ServiceTitan zapewnia solidne możliwości planowania i wysyłania, umożliwiając firmom efektywne przydzielanie zadań i zarządzanie nimi. Oferuje interfejs kalendarza typu „przeciągnij i upuść”, który ułatwia planowanie, śledzenie techniczne w czasie rzeczywistym i zoptymalizowane wyznaczanie tras dla ekip serwisowych. Rysunek 9 przedstawia moduł kalendarz.

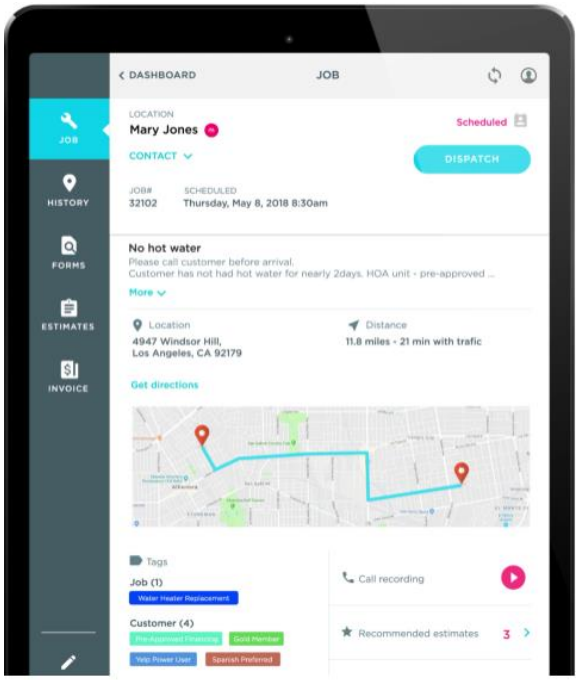


Rysunek 9 ServiceTitan – Kalendarz. Źródło: [10]

2. Zarządzanie zleceniami pracy: firmy mogą tworzyć zlecenia pracy i zarządzać nimi, śledzić postęp prac i rejestrować ważne szczegóły pracy. Umożliwia technikom dostęp do zleceń pracy na urządzeniach mobilnych, przeglądanie historii zadań i aktualizowanie statusu zadań w czasie rzeczywistym.

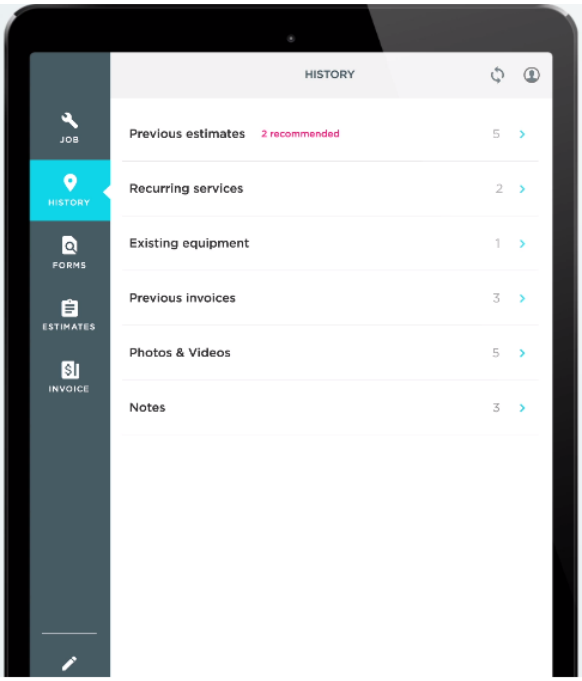
3. Zarządzanie klientami: ServiceTitan oferuje scentralizowaną bazę danych klientów, w której firmy mogą przechowywać i zarządzać informacjami o klientach. Zapewnia funkcje, takie jak profile klientów, historia usług, przypomnienia o spotkaniach i narzędzia do komunikacji z klientami, umożliwiając firmom dostarczanie spersonalizowanej i wydajnej obsługi klienta.

4. Aplikacja mobilna dla techników terenowych: umożliwia dostęp do szczegółów zlecenia, aktualizację statusu zadania, robienie zdjęć i zbieranie podpisów klientów na miejscu. Aplikacja działa w trybie offline, dzięki czemu technicy mogą nadal wykonywać swoje zadania nawet w obszarach o ograniczonej łączności. Rysunek 10 przedstawia szczegóły zlecenia prezentowane w aplikacji mobilnej.



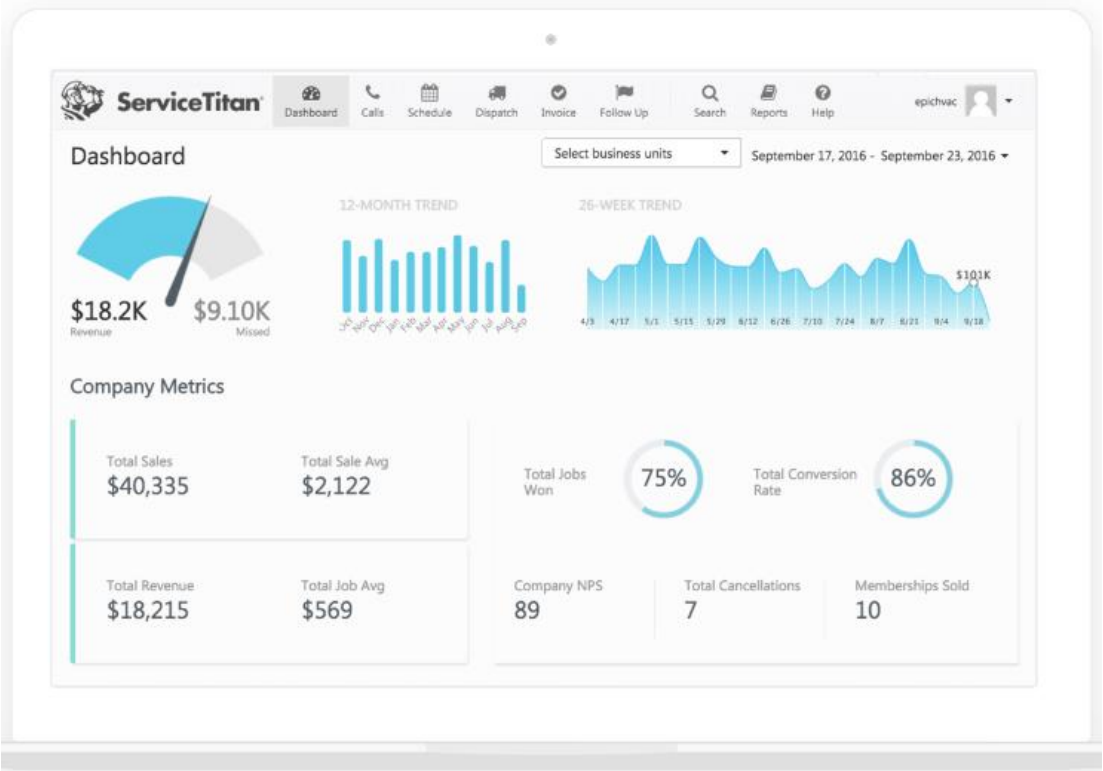
Rysunek 10 ServiceTitan – Szczegóły zlecenia. Źródło: [11]

5. Rozliczenia i płatności: Aplikacja ułatwia bezproblemowe procesy rozliczeń i płatności. Umożliwia firmom generowanie profesjonalnych faktur, zbieranie płatności w terenie i przetwarzanie płatności za pomocą różnych metod, w tym kart kredytowych, czeków i elektronicznych bramek płatniczych. Rysunek 11 przedstawia historie zlecenia prezentowane w aplikacji mobilnej.



Rysunek 11 ServiceTitan – Historia zlecenia. Źródło: [11]

6. Raportowanie i analityka: ServiceTitan oferuje solidne możliwości raportowania i analizy, aby pomóc firmom uzyskać wgląd w ich działalność. Zapewnia gotowe raporty i konfigurowalne pulpity nawigacyjne, które pozwalają firmom śledzić kluczowe wskaźniki wydajności, mierzyć produktywność techników, monitorować przychody i identyfikować obszary wymagające poprawy. Rysunek 12 przedstawia dashboard raportowy.



Rysunek 12 ServiceTitan - Dashboard raportowy. Źródło: [12]

Do najważniejszych korzyści płynących z wykorzystania aplikacji ServiceTitan należy:

1. Usprawnione operacje: automatyzując wiele procesów, ServiceTitan pomaga firmom usprawnić ich operacje i wyeliminować ręczną papierkową robotę. Poprawia wydajność planowania, zmniejsza liczbę zadań administracyjnych i usprawnia ogólne zarządzanie przepływem pracy.

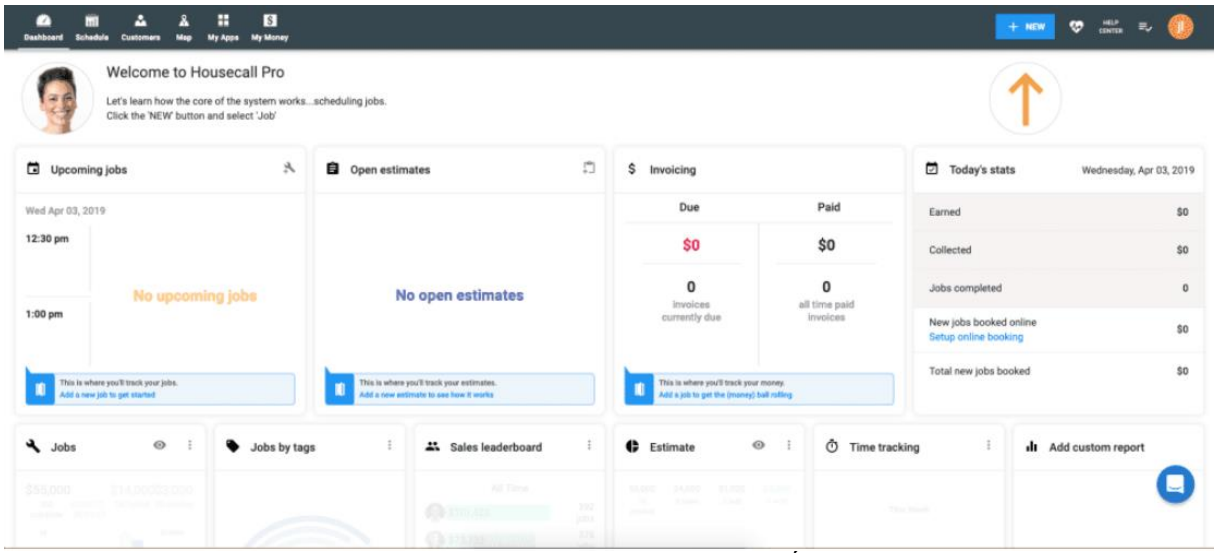
2. Udoskonalona obsługa klienta: dzięki funkcjom zarządzania klientami ServiceTitan firmy mogą zapewnić klientom wyjątkową obsługę. Możliwość dostępu do informacji o klientach, historii usług i spersonalizowanej komunikacji pomaga budować silne relacje z klientami i zwiększać ich satysfakcję.

3. Ulepszone zarządzanie pracownikami: ServiceTitan zapewnia narzędzia do zarządzania technikami pracującymi w terenie, śledzenia ich wydajności i optymalizacji ich harmonogramów. Pomaga firmom efektywnie przydzielać zadania, monitorować dostępność techników i zwiększać produktywność siły roboczej.

4. Zwiększone przychody i wydajność: Poprawiając wydajność operacyjną i optymalizując przepływy pracy, ServiceTitan pomaga firmom zwiększać przychody i rentowność. Możliwość rejestrowania dokładniejszych danych o zleceniach, generowania profesjonalnych faktur i szybkiego przetwarzania płatności przyczynia się do lepszych wyników finansowych.

2.5. HouseCall Pro

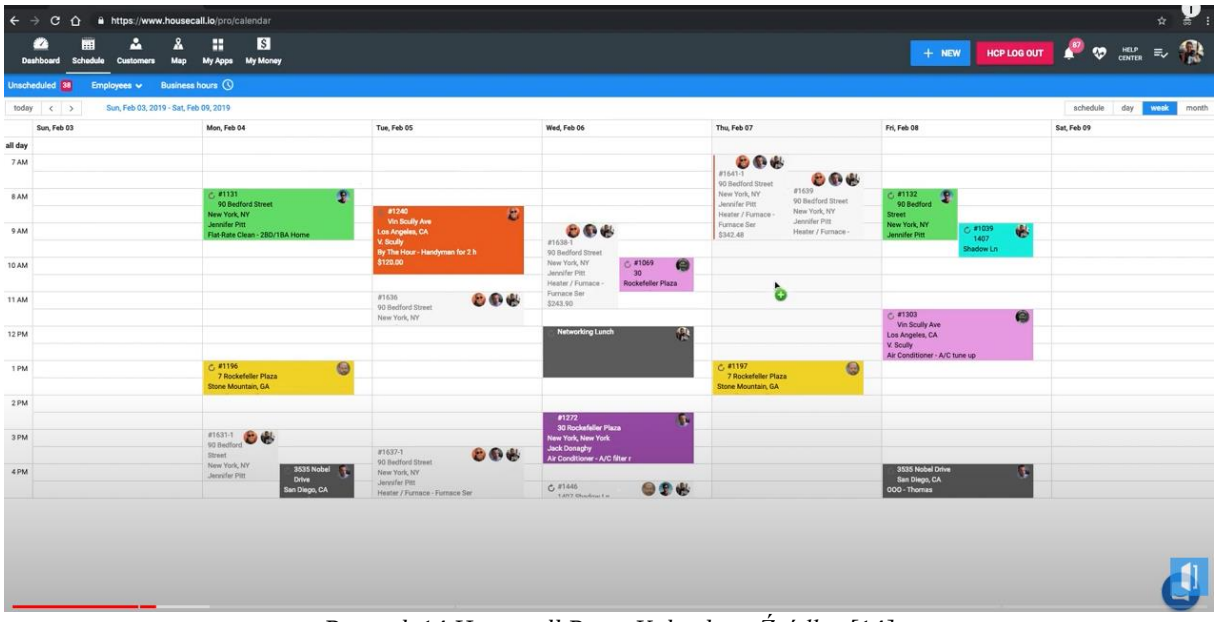
Housecall Pro [16] to oprogramowanie do zarządzania usługami terenowymi zaprojektowane w celu usprawnienia operacji biznesowych w zakresie usług domowych. Zapewnia różnorodne funkcje, które pomagają firmom zarządzać planowaniem, wysyłką, fakturowaniem, komunikacją z klientem. Rysunek 13 przedstawia stronę główną.



Rysunek 13 HouseCall Pro – Strona główna. Źródło: [13]

Housecall Pro wyróżnia się szeregiem kluczowych funkcji, które przyczyniają się do jego sukcesu i popularności. Wśród nich można wymienić:

1. Planowanie i wysyłka: Housecall Pro oferuje łatwy w użyciu interfejs planowania, który pozwala firmom łatwo zarządzać zadaniami i przydzielać je technikom pracującym w terenie. Zapewnia kalendarz typu „przeciągnij i upuść", stan pracy oznaczony kolorami oraz możliwość sprawdzenia dostępności technika w czasie rzeczywistym. Dyspozytorzy mogą skutecznie przydzielać zadania technikom na podstawie ich lokalizacji i umiejętności. Rysunek 14 przedstawia moduł kalendarza.



Rysunek 14 Housecall Pro – Kalendarz. Źródło: [14]

2. Aplikacja mobilna: pozwala technikom terenowym zarządzać swoimi zadaniami i komunikować się z biurem. Technicy mogą uzyskiwać dostęp do swoich harmonogramów, przeglądać szczegóły zadań, nawigować do lokalizacji klientów, aktualizować statusy zadań i rejestrować podpisy klientów bezpośrednio na swoich urządzeniach mobilnych. Ta wymiana informacji w czasie rzeczywistym poprawia komunikację i zmniejsza liczbę zadań administracyjnych. Rysunek 15 przedstawia przykładowe zrzuty ekranu z aplikacji mobilnej.

HouseCall Pro screenshot #2

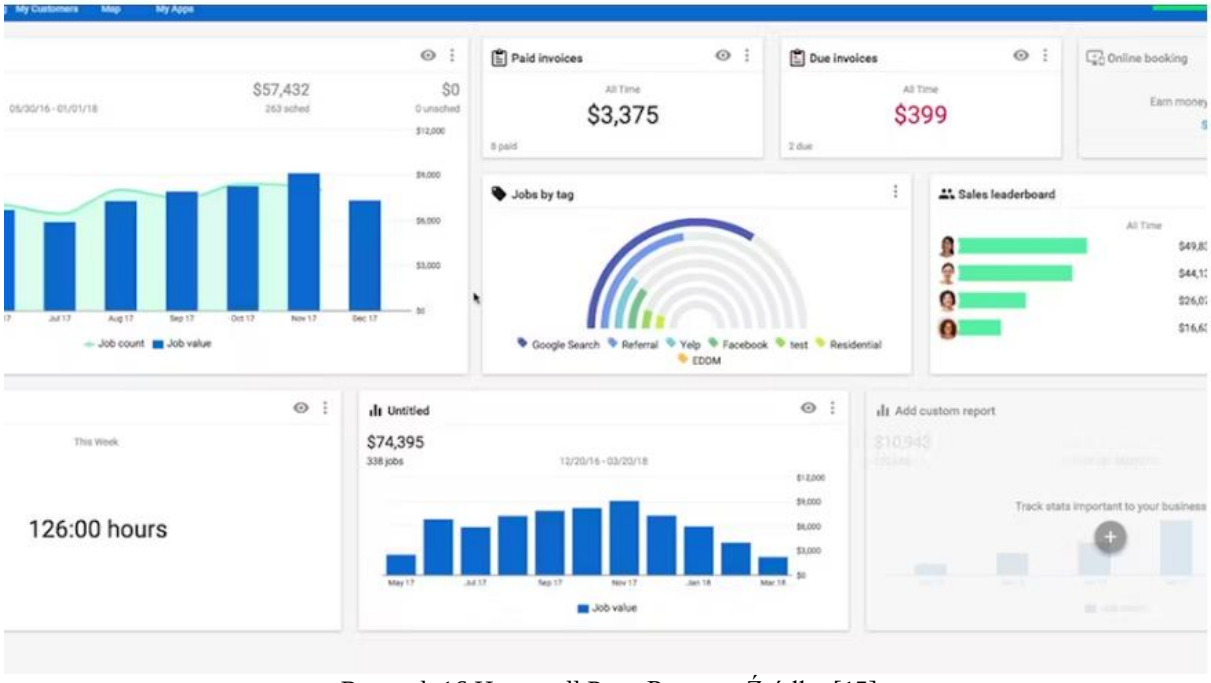
Rysunek 15 Housecall Pro - Aplikacja mobilna. Źródło: [13]

3. Rezerwacja online i zarządzanie klientami: Housecall Pro umożliwia firmom oferowanie klientom opcji rezerwacji online. Klienci mogą planować spotkania, przeglądać dostępne przedziały czasowe i otrzymywać automatyczne przypomnienia. Platforma zawiera również bazę danych klientów do przechowywania ważnych informacji o klientach, historii usług i notatek do spersonalizowanych interakcji.

4. Rozliczenia i płatności: dzięki Housecall Pro firmy mogą łatwo tworzyć profesjonalne faktury i pobierać płatności. Oprogramowanie pozwala na dostosowywanie szablonów faktur, integrację z bramkami płatniczymi oraz możliwość akceptowania różnych metod płatności, w tym kart kredytowych i płatności mobilnych. Usprawnia to proces rozliczeń i zapewnia terminowe płatności.

5. Marketing i komunikacja z klientem: Housecall Pro oferuje narzędzia do automatyzacji marketingu i komunikacji z klientem. Firmy mogą wysyłać automatyczne przypomnienia e-mailowe i SMS-owe, kontynuacje usług i spersonalizowane kampanie marketingowe. Platforma obsługuje również dwukierunkowe przesyłanie wiadomości do klientów, umożliwiając firmom kontakt z klientami w czasie rzeczywistym.

6. Raportowanie i analiza: Housecall Pro zapewnia funkcje raportowania i analizy, które pomagają firmom śledzić ich wydajność i podejmować świadome decyzje. Użytkownicy mogą generować raporty dotyczące przychodów, ukończonych zadań, opinii klientów i nie tylko. Te spostrzeżenia pozwalają firmom identyfikować obszary wymagające poprawy, optymalizować operacje i stymulować wzrost. Rysunek 16 przedstawia moduł raportowania.



Rysunek 16 Housecall Pro - Raporty. Źródło: [15]

Warto również zaznaczyć, że Housecall Pro dostarcza szereg dodatkowych możliwości, które znacząco ułatwiają zarządzanie firmą oraz zwiększają efektywność pracy. Oto niektóre z nich:

1. Usprawnione operacje: Housecall Pro automatyzuje i centralizuje wiele zadań operacyjnych, redukując ręczną pracę i upraszczając procesy. Prowadzi to do większej wydajności, mniejszej liczby błędów i zwiększonej produktywności dla firm.

2. Ulepszona obsługa klienta: udostępniając rezerwację online, przypomnienia o spotkaniach i przejrzystą komunikację, Housecall Pro pomaga firmom zapewnić lepszą obsługę klienta. Klienci korzystają z wygody rezerwacji terminów i otrzymywania aktualizacji, co prowadzi do zwiększonej satysfakcji i powtarzalności transakcji.

3. Zwiększone możliwości raportowania i analizy: Housecall Pro zapewniają firmom cenny wgląd w ich wydajność. Mogą śledzić kluczowe wskaźniki, identyfikować trendy i podejmować decyzje w oparciu o dane, aby zoptymalizować swoją działalność i stymulować rozwój firmy.

4. Funkcje rozliczeń i płatności: Housecall Pro usprawniają proces rozliczeń, zapewniając szybsze płatności i zmniejszając koszty administracyjne. Pomaga to firmom poprawić przepływy pieniężne i zwiększyć przychody.

2.6. Podsumowanie

Omówione aplikacje posiadają wiele zaawansowanych funkcjonalności i generalnie możemy podzielić je na dwa typy:

- Oprogramowanie wspomagające zarządzanie zleceniami udostępniające funkcjonalności planowania zleceń, komunikacji z pracownikami i klientami, ewidencji przebiegu prac, raportowania;
- Aplikacje pozwalające na zarządzenie stanami magazynowymi;

Wśród dostępnego oprogramowania brak jest ofert, które łączyłyby w jednej aplikacji ww. rodzaje produktów. Proponowane rozwiązanie posiada zarówno moduł odpowiedzialny za zarządzanie zleceniami jak i część przeznaczoną do ewidencji stanów magazynowych. Użytkownicy będą mogli efektywniej koordynować prace posiadając szczegółową wiedzę o istniejących zasobach magazynowych. Z kolei informacje o planowanych realizacjach pozwolą zoptymalizować planowanie zakupów elementów montażowych i narzędzi.

Dodatkową funkcjonalnością, której nie posiadają omówione systemy jest moduł zarządzania usterkami. Posiadając wiedzę o statusach i terminach napraw użytkownik może lepiej skalibrować zakupy nowych narzędzi i ew. ograniczyć koszty ich zakupów, jeśli narzędzia są możliwe do naprawy i następnie do wykorzystania w realizacji planowanych zleceń.

3. Propozycja systemu E-montażysta

W tym rozdziale została przedstawiona propozycja implementacji systemu "E-montażysta", którego celem jest obsługa firm zajmujących się montażem oraz zarządzaniem własnymi magazynami. Opisane zostały szczegóły dotyczące tego rozwiązania wraz z wymaganiami funkcjonalnymi i niefunkcjonalnymi. Proces ten rozpoczyna się od momentu zarejestrowania usługobiorcy i obejmuję różne aspekty zarządzania przedsiębiorstwem, takie jak dodanie bazy pracowników i klientów, rejestracja zleceń i ich etapów czy dodanie dostępnych narzędzi i elementów do magazynu.

W ramach rozdziału przedstawiono opis proponowanego rozwiązania systemu "E-montażysta" wraz z wymaganiami funkcjonalnymi i niefunkcjonalnymi. Dodatkowo, przedstawiono diagramy przypadków użycia oraz diagram klas. Należy jednak zaznaczyć, że w niniejszym rozdziale nie został jeszcze dokładnie omówiony podział systemu na poszczególne warstwy.

3.1. Cel

Celem systemu E-montażysta jest stworzenie kompleksowego rozwiązania informatycznego umożliwiającego efektywne zarządzanie danymi przedsiębiorstw zajmujących się montażem oraz zarządzaniem własnymi magazynami.

3.2. Założenia

System E-montażysta w swoim założeniu jest elastycznym narzędziem spełniającym zróżnicowane wymagania, dopasowując się do indywidualnych potrzeb firm. System ma na celu usprawnienie procesów magazynowych, monitorowanie przebiegu zleceń oraz optymalizacje zarządzania przechowywanych towarów. Główną korzyścią, jaką ma przynieść, jest zwiększenie efektywności operacyjnej, skrócenie czasu realizacji usługi oraz zautomatyzowanie procesów w celu zmniejszenia ryzyka błędów ludzkich. Dodatkowo przyczyni się do usprawnienia komunikacji na poziomie pracowników oraz umożliwi przechowywanie wszystkich danych dotyczących funkcjonowania przedsiębiorstwa w jednym miejscu.

3.3. Słownik

Brygadzista - pracownik odpowiedzialny za zarządzanie zespołem montażystów. Zgłasza zapotrzebowania ad hoc, pobiera i zwraca do magazynu elementy i narzędzia. Nadaje status etapowi zlecenia oraz realizuję je wraz z montażystami.

Element - przedmiot wykorzystywany przez montażystów przy realizacji zleceń. Każdy element posiada swój typ. Wydawane są z magazynu brygadziście, a po zakończeniu prac mogą wrócić na stan magazynu. Element posiada swoją historię przechowywaną w klasie WydanieZwrotElementu.

Etap Zlecenia - stanowi podstawę zlecenia. Zawiera informacje o potrzebnych elementach, narzędziach, terminach realizacji, montażystach, plikach projektowych. Może trwać maksymalnie jeden dzień roboczy. Pracownicy mogą komentować etap podczas jego trwania.

Firma - organizacja, która wykonuje zlecenia. Są w niej zatrudnieni pracownicy oraz może posiadać swój magazyn.

Handlowiec - pracownik odpowiedzialny za wprowadzanie nowych zleceń do systemu. Może dodatkowo przypisać maksymalnie dwa etapy do zlecenia. Przekazuje zlecenia do specjalisty.

Historia Zatrudnienia - klasa przechowywująca historie zatrudnienia pracownika wraz z datą rozpoczęcia i zakończenia umowy.

Kierownik Magazynu - pracownik odpowiedzialny za zarządzanie magazynem i magazynierami. Zarządza ewidencją narzędzi i elementów. Może dodawać nowy typ elementu.

Komentarz - dodatkowa notatka przypisywana do konkretnego etapu. Przechowuje informacje przez kogo i kiedy została dodana. Może zawierać zdjęcie oraz opis tekstowy. Dodatkowa funkcjonalność to możliwość oznaczenia pracownika, który dostanie powiadomienie o danym komentarzu.

Lokalizacja - opis, współrzędne i dane adresowe magazynu czy zlecenia.

Magazyn - miejsce przechowywania elementów i narzędzi. W magazynie pracują magazynierzy. W systemie może występować wiele magazynów w różnych lokalizacjach, a co za tym idzie każdy magazyn posiada swoją nazwę, godziny otwarcia oraz opis.

Magazynier - pracownik odpowiedzialny za przyjmowanie i wydawanie zasobów oraz potwierdzanie przy tym ich sprawności technicznej. Ma możliwość sprawdzenia stanu magazynu. Wprowadza również zapotrzebowanie na dany element lub narzędzie.

Menager - pracownik odpowiedzialny za ustawienia priorytetów oraz przypisanie brygadzistów do odpowiednich etapów zlecenia. Wprowadza również informacje o niedostępnościach pracowników. Dodatkowo opiniuje zapotrzebowanie ad hoc.

Montażysta - pracownik odpowiedzialny za realizację zleceń oraz zgłaszanie ewentualnych zdarzeń związanych z narzędziem lub elementem. Wprowadza również zapotrzebowanie ad hoc na dany element lub narzędzie.

Narzędzie - przedmiot wielokrotnego użytku wykorzystywany przez montażystów przy realizacji zleceń, każde narzędzie posiada swój unikalny kod QR oraz jest jakiegoś typu. Narzędzia są wydawane z magazynu brygadziście, po zakończeniu prac powinny wrócić na stan magazynu. Dla każdego narzędzia zapisywana jest jego historia użytkowania.

Niedostępność – informacja o niedostępności danego pracownika, zawiera powód oraz datę.

Pracownik – klasa zawierająca dane osobowe i dane kontaktowe, oraz login do systemu osoby zatrudnionej w firmie.

Specjalista – pracownik odpowiedzialny za podział zlecenia na etapy, dodawanie załączników oraz przypisywanie narzędzia i elementy o etapu zlecenia. Specjalista wprowadza liczbę wymaganych zdjęć jako wymagane udokumentowanie realizacji zakończenia etapu. Poza tym może podglądać elementy i narzędzia z magazynu oraz modyfikować listę narzędzi potrzebnych przy montażu. Specjalista przypisuje do etapu również listę części zewnętrznych (nie występujących w systemie) w formie PDF. Dodatkowo opiniuje zapotrzebowanie ad hoc.

Typ Narzędzia - klasa kategoryzująca narzędzia, przechowywująca dane o ilości poszczególnych typów, stanach krytycznych oraz o serwisowaniu.

Wydanie Narzędzia – klasa przechowywująca dane związane z procesem rejestracji wydania i zwrotu narzędzia.

Wydanie Zwrot Elementu - klasa zawierająca informacje o historii wykorzystania elementu.

Załącznik - dokument lub zdjęcie, który można dołączyć do zlecenia, etapu zlecenia czy komentarza.

Zapotrzebowanie ad hoc - funkcjonalność, która umożliwia brygadziście zgłoszenie pilnego zapotrzebowanie na narzędzia lub elementy.

Zdarzenie Elementu - funkcjonalność, która umożliwia rejestrację różnych zdarzeń związanych z elementami, takich jak usterka czy zaginięcie.

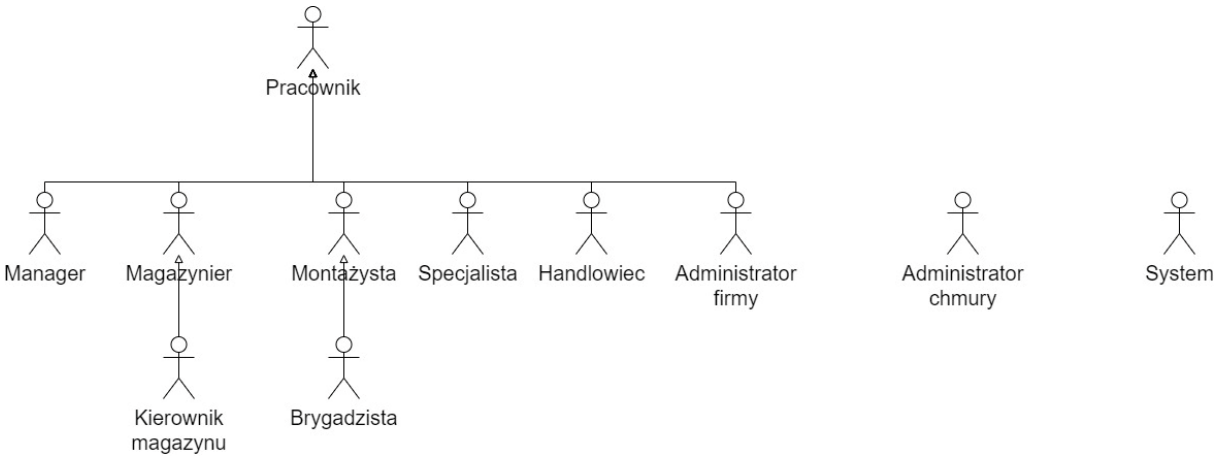
Zdarzenie Narzędzia - funkcjonalność, która umożliwia rejestrację różnych zdarzeń związanych z narzędziami, takich jak usterka czy zaginięcie.

Zlecenie - klasa główna systemu. Zlecenie jest tworzone przez handlowca. Specjalista dodaje projekt zlecenia. Manager nadaje priorytet zleceniu i przypisuje je do brygadzisty. Biorąc pod uwagę, że projekt jest rozwiązaniem chmurowym, zlecenie jest powiązane z firmą, która owe zlecenie przyjęła jak i ze zleceniodawcą, który owe zlecenie zlecił. Zlecenie posiada etapy.

Zleceniodawca - klient zlecający wykonanie prac (zlecenia).

3.4. Użytkownicy systemu

Użytkownicy systemu są podmiotami, którzy korzystają z funkcjonalności systemu i udostępnionych im zasobów w celu realizacji swoich zadań. W ramach prac zespołu analizy, wyróżniono różne typy użytkowników, zależne od ich ról i uprawnień. Każda rola użytkownika może mieć określony zakres odpowiedzialności i dostęp do konkretnych funkcji systemu. Rysunek 17 przedstawia hierarchie dziedziczenia użytkowników systemu E-montażysta.



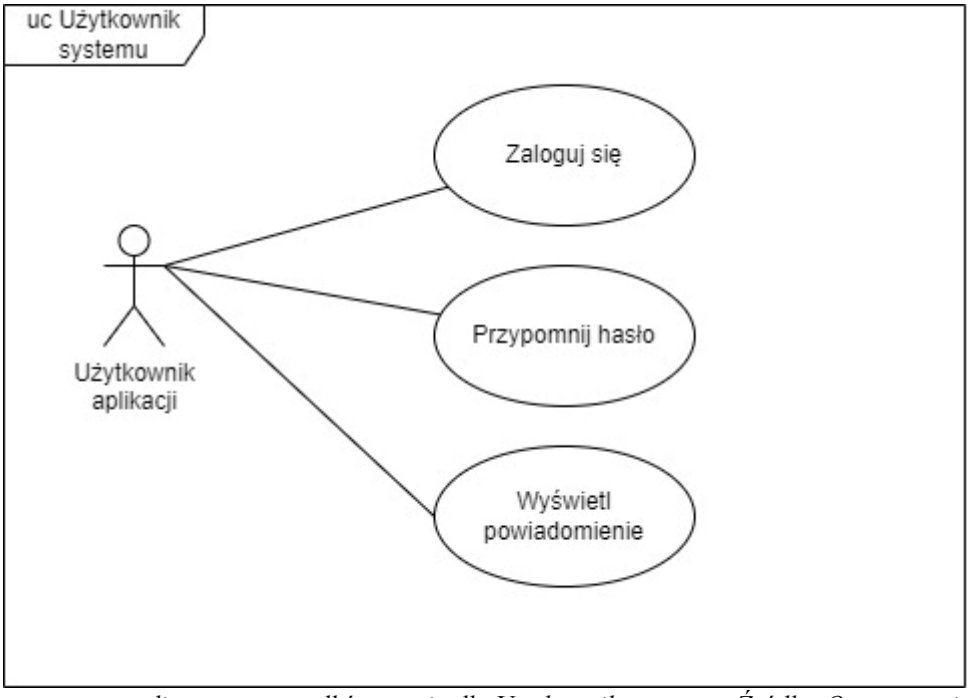
Rysunek 17 E-montażysta – Użytkownicy systemu. Źródło: Opracowanie własne

3.5. Diagram przypadków użycia

W tym podrozdziale zostaną przedstawione diagramy przypadków użycia dla systemu E-montażysta. Diagramy zostały opracowany przez zespół analizy w celu zaprezentowania głównych zadań, które mogą być wykonywane przez użytkowników systemu.

3.5.1 Użytkownik systemu

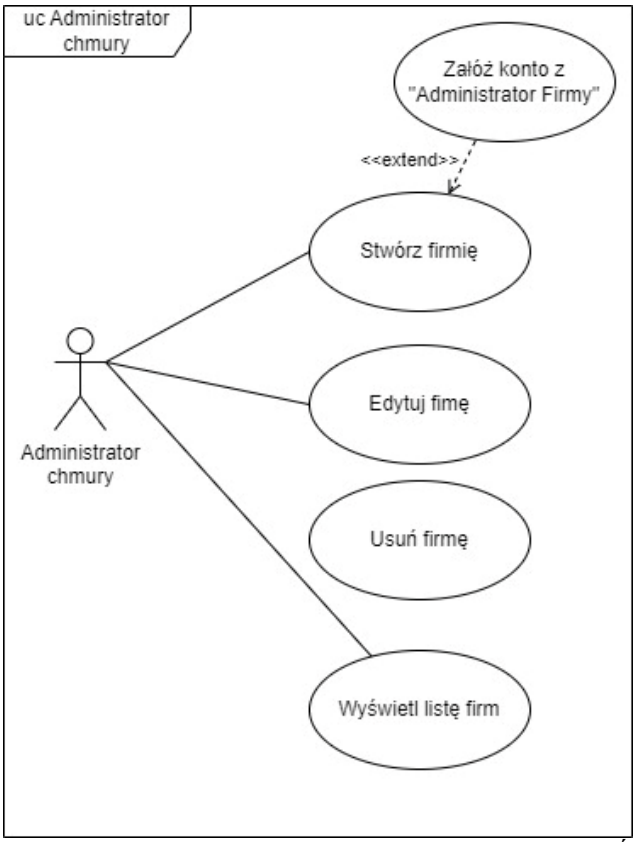
Na rysunku 18 przedstawiono przypadki użycia dla każdego użytkownika systemu.



Rysunek 18 E-montażysta – diagram przypadków użycia dla Użytkownika systemu. Źródło: Opracowanie własne

3.5.2 Administrator chmury

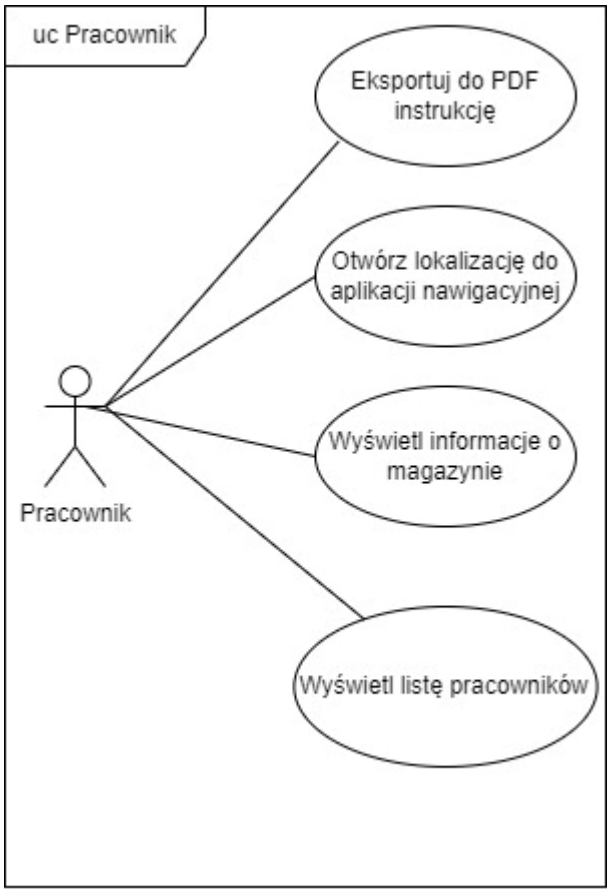
Administrator chmury odgrywa kluczową rolę w zarządzaniu firmami. Rysunek 19 przedstawia przypadki użycia dla tej roli.



Rysunek 19 E-montażysta – diagram przypadków użycia dla Administrator chmury. Źródło: Opracowanie własne

3.5.3 Pracownik

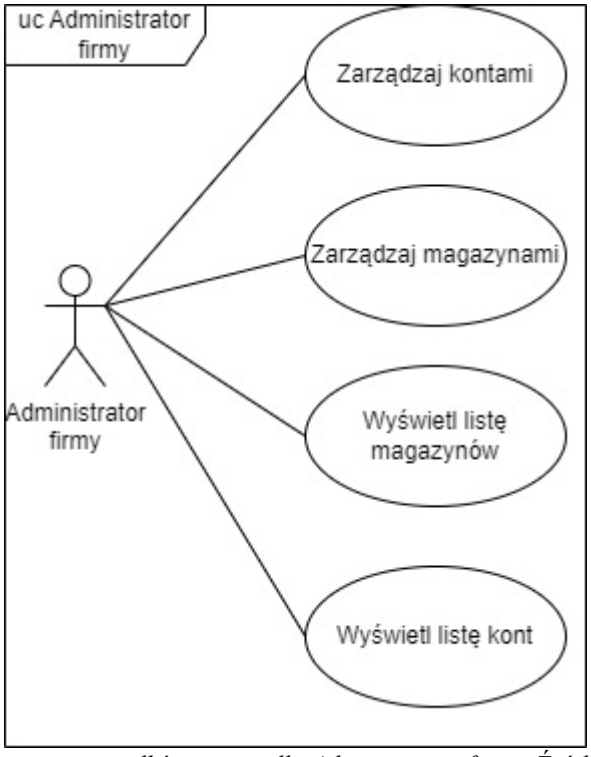
Pracownik stanowi najwyższy poziom abstrakcji w hierarchii aktorów systemu. Większość ról użytych w systemie dziedziczy po nim, co oznacza, że posiadają te same cechy i funkcjonalności, ale mają również dodane dodatkowe, specyficzne dla swoich potrzeb. Pracownik stanowi punkt wyjścia dla większości aktorów, a rysunek 20 przedstawia przypadki użycia związane z tą rolą.



Rysunek 20 E-montażysta – diagram przypadków użycia dla Pracownik. Źródło: Opracowanie własne.

3.5.4 Administrator firmy

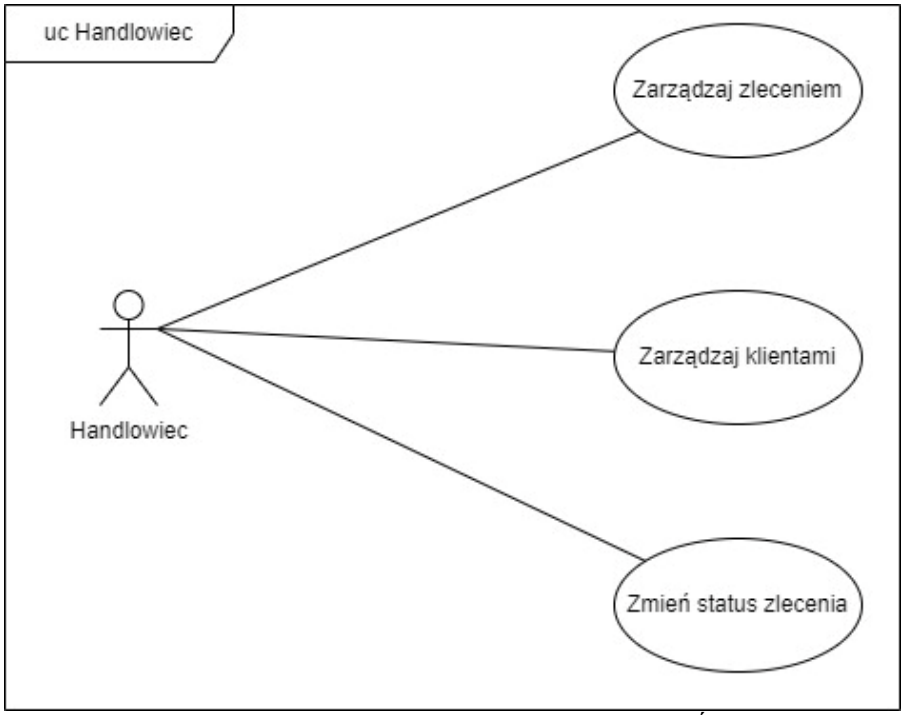
Rola Administratora firmy odgrywa kluczową funkcję w systemie, odpowiedzialną za zarządzenie kontami użytkowników oraz magazynami. Posiada pełny wgląd i kontrolę nad różnymi aspektami systemu, umożliwiając efektywne administrowanie zasobami i dostępem do danych. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 21.



Rysunek 21 E-montażysta – diagram przypadków użycia dla Administrator firmy. Źródło: Opracowanie własne.

3.5.5 Handlowiec

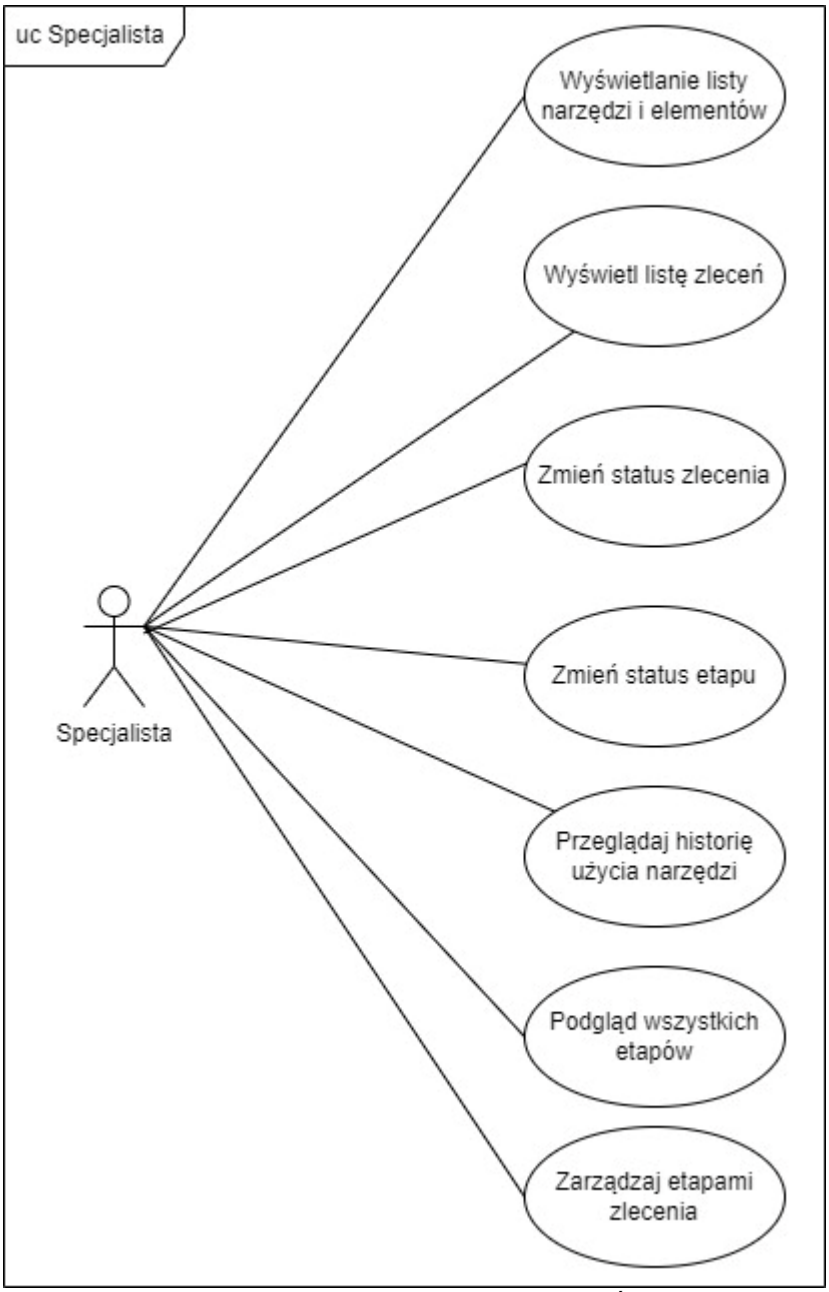
Handlowiec odgrywa istotną rolę w systemie, pełniąc funkcję pośrednika pomiędzy firmą a klientami. Odpowiedzialny jest za zarządzanie zleceniami. Współpracuje z innymi aktorami systemu, aby zaspokoić potrzeby klientów. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 22.



Rysunek 22 E-montażysta – diagram przypadków użycia dla Administrator firmy. Źródło: Opracowanie własne.

3.5.6 Specjalista

Specjalista odgrywa istotną rolę w systemie, mając za zadanie dostarczanie specjalistycznej wiedzy i umiejętności w zakresie organizacji współpracy między klientami a pracownikami. Jego odpowiedzialnością jest skuteczne zarządzanie zleceniami, w tym zmiana statusów i zarządzanie etapami. Specjalista współpracuje z innymi uczestnikami systemu, aby zapewnić wysoką jakość usług. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 23.

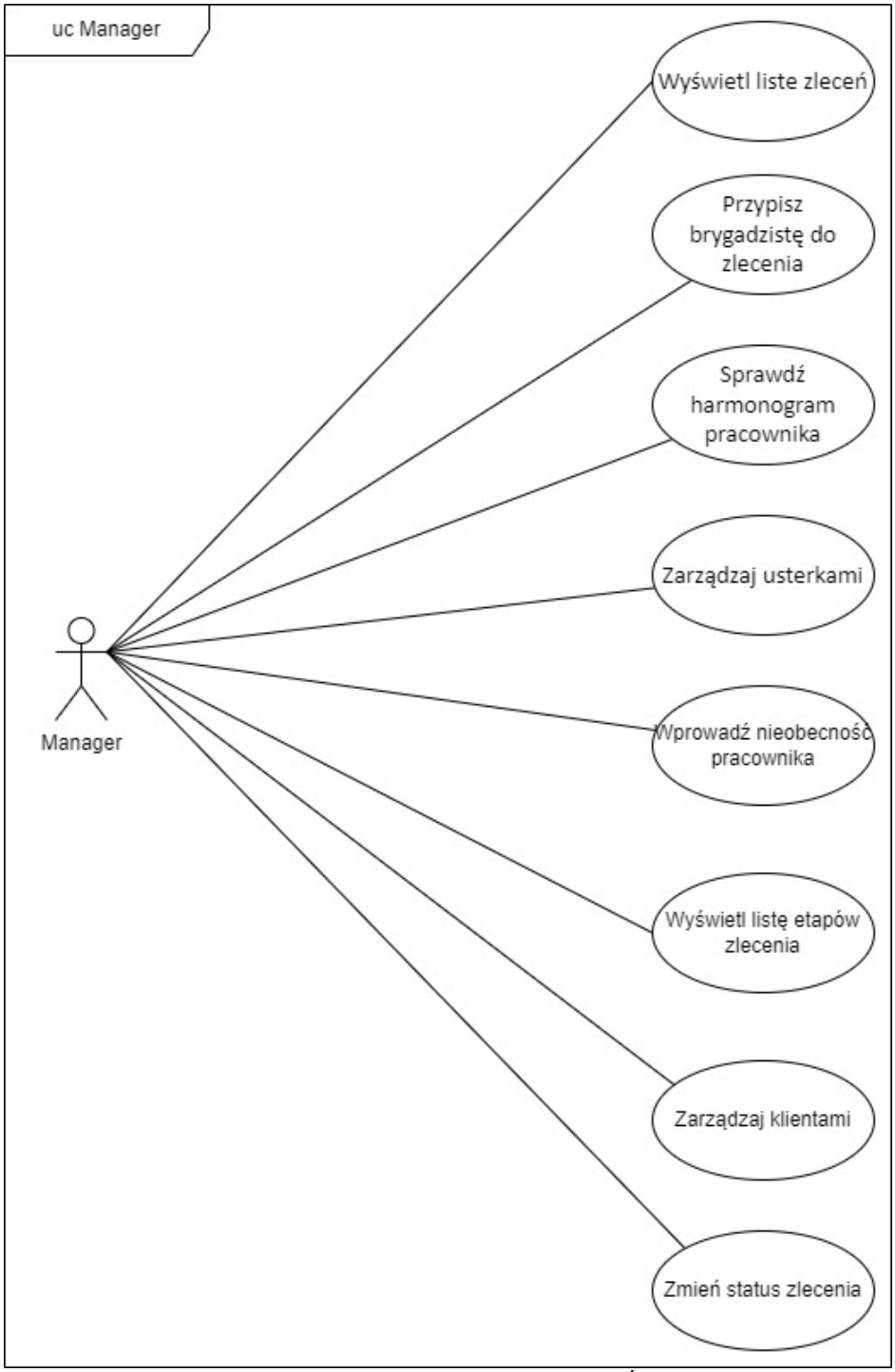


Rysunek 23 *E-montażysta – diagram przypadków użycia dla Specjalista. Źródło: Opracowanie własne.*

3.5.7 Manager

Manager pełni kluczową rolę w systemie, odpowiadając za kierowanie działaniami w ramach przedsiębiorstwa. Kieruje pracą brygadzistów, przypisując ich do odpowiednich zleceń, monitoruje nieobecności pracowników oraz zarządza usterkami i klientami. Współpracuje z innymi aktorami

systemu, aby zapewnić efektywne funkcjonowanie organizacji. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 24.

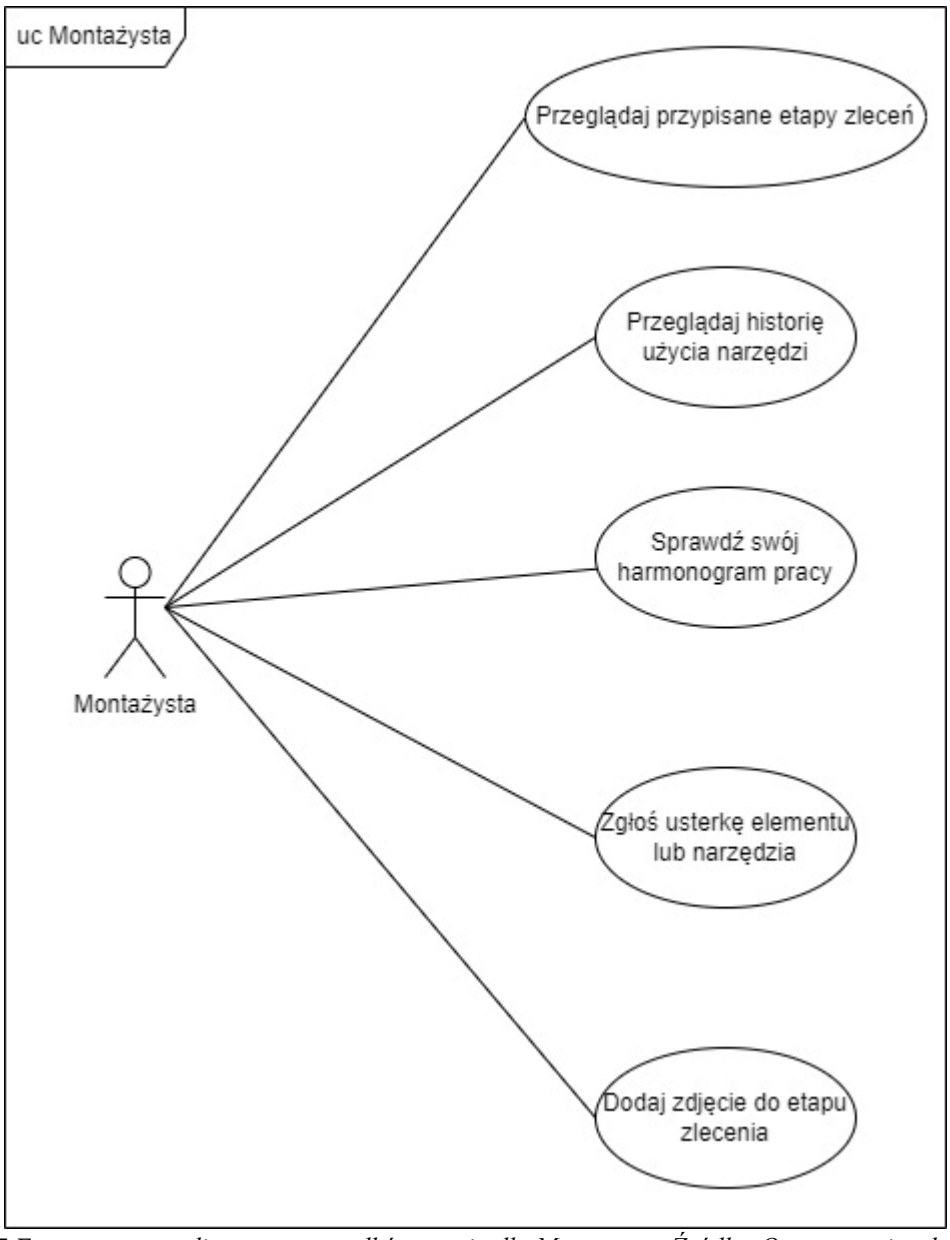


Rysunek 24 E-montażysta – diagram przypadków użycia dla Manager. Źródło: Opracowanie własne.

3.5.8 Montażysta

Montażysta pełni kluczową rolę w systemie, zajmując się realizacją etapów zleceń. Odpowiada za składanie i montaż elementów przy użyciu narzędzi zgodnie z określonymi wytycznymi. Montażysta

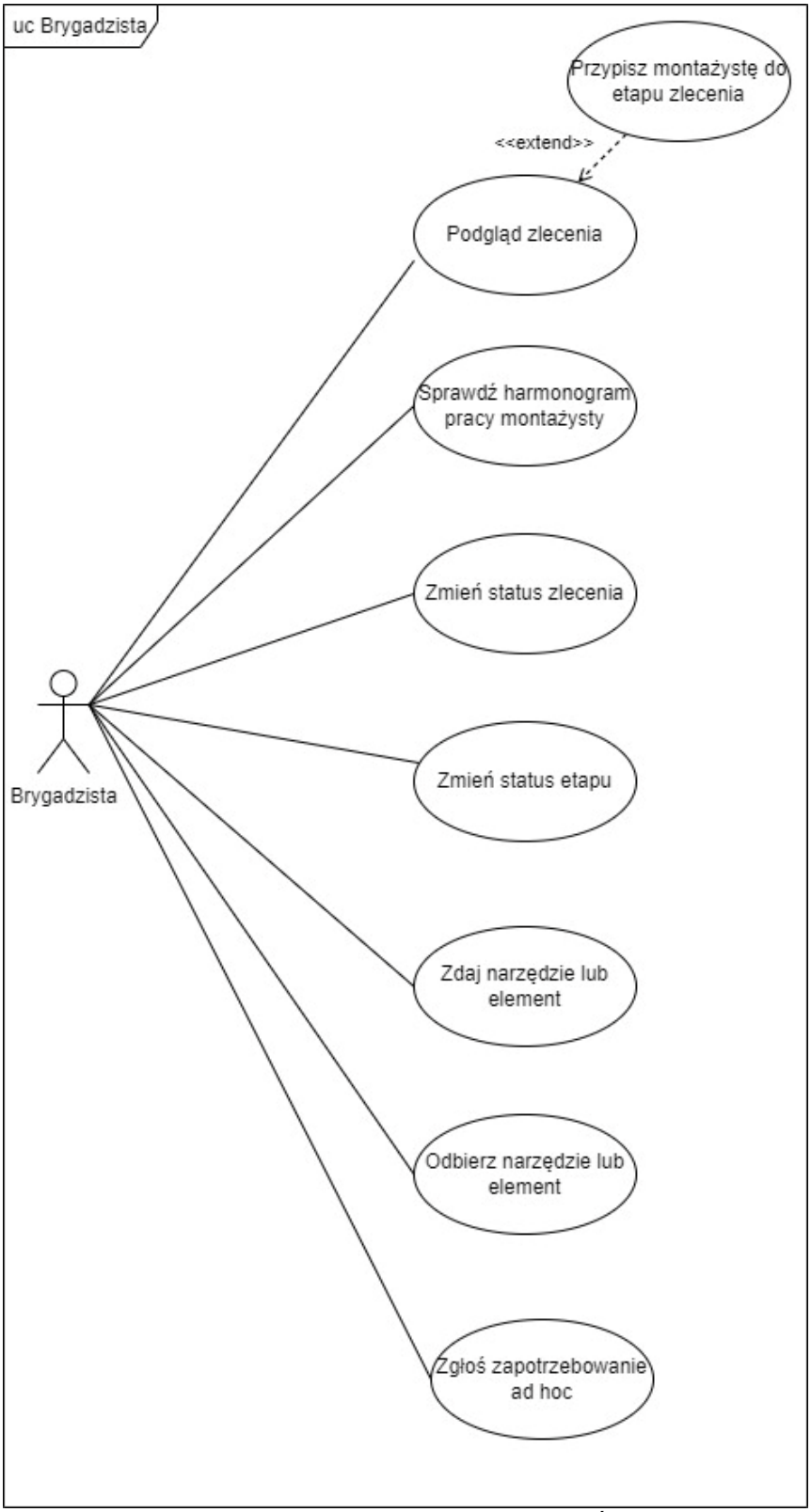
współpracuje z innymi aktorami systemu, aby zapewnić sprawne i precyzyjne wykonanie etapów zleceń. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 25.



Rysunek 25 E-montażysta – diagram przypadków użycia dla Montażysta. Źródło: Opracowanie własne.

3.5.9 Brygadzista

Brygadzista odgrywa istotną rolę w systemie, pełniąc funkcję nadzoru i koordynacji pracowników. Jest odpowiedzialny za przypisywanie zadań do odpowiednich montażystów, monitorowanie postępu prac oraz zarówno odbieranie, jak i zdawanie elementów i narzędzi w ramach zlecenia. Brygadzista współpracuje z innymi aktorami systemu, aby zapewnić efektywne działanie zespołu montażystów. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 26.

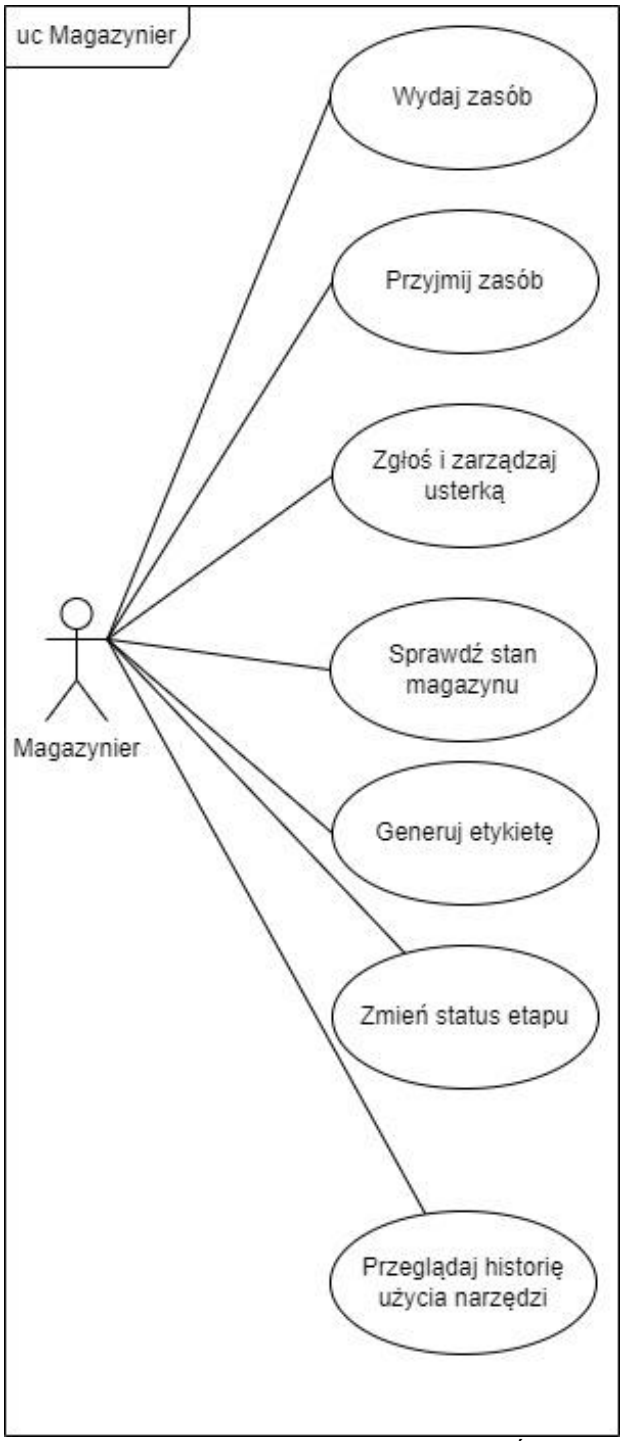


Rysunek 26 E-montażysta – diagram przypadków użycia dla Brygadzista. Źródło: Opracowanie własne.

3.5.10 Magazynier

Magazynier odgrywa istotną funkcję w systemie, zajmując się operacjami związanymi z zarządzaniem magazynem. Odpowiedzialny jest za przyjmowanie i wydawanie towarów. Ma dostęp do

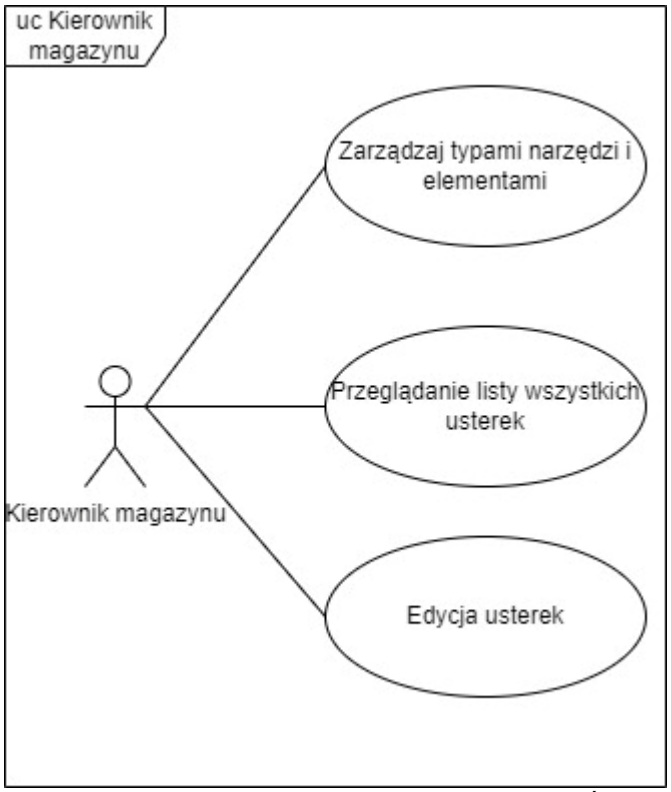
informacji dotyczących stanu magazynowego, kontroluje poziom zapasów oraz dokonuje rejestracji przychodzących i wychodzących zasobów. Współpracuje również z innymi aktorami systemu, w celu efektywnego zarządzania zapasami i realizacji zleceń i ich etapów. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 27.



Rysunek 27 E-montażysta – diagram przypadków użycia dla Magazynier. Źródło: Opracowanie własne.

3.5.11 Kierownik magazynu

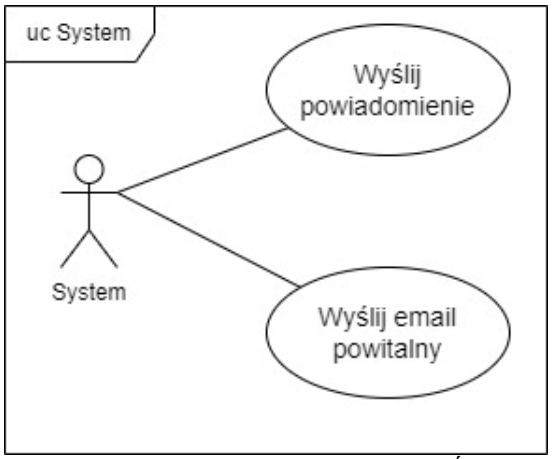
Kierownik magazynu pełni kluczową rolę w systemie, odpowiadając za efektywne zarządzanie magazynem. Jego zadaniem jest zarządzanie różnymi typami narzędzi i elementów, a także obsługa zgłaszanych usterek. Współpracuje on z innymi uczestnikami systemu w celu zapewnienia sprawnego funkcjonowania magazynu. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 28.



Rysunek 28 E-montażysta – diagram przypadków użycia dla Kierownik magazynu. Źródło: Opracowanie własne.

3.5.12 System

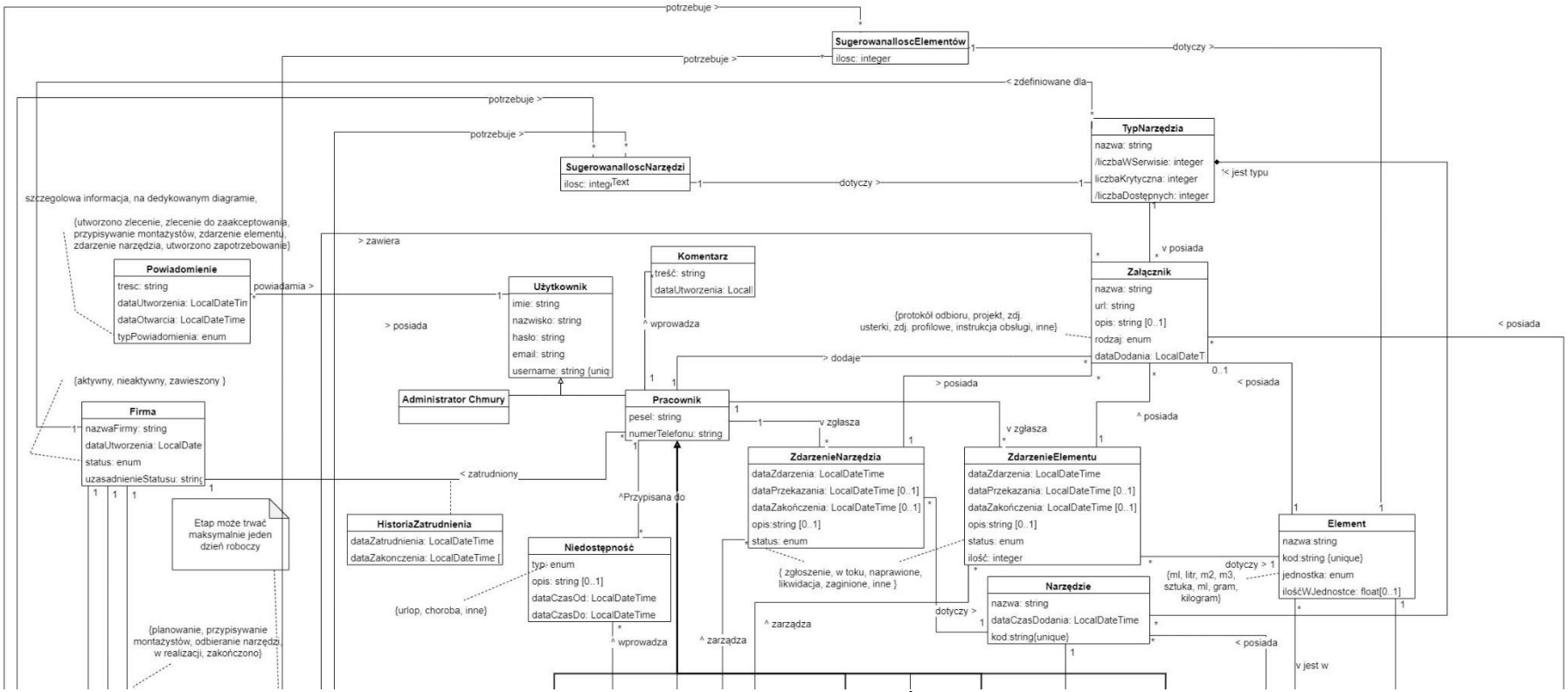
System pełni istotną funkcję w ramach całego rozwiązania, zapewniając działanie różnych usług w określonych jednostkach czasowych. Odpowiada za wysyłanie powiadanie. Przypadki użycia związane z tą rolą zostały przedstawione na rysunku 29.



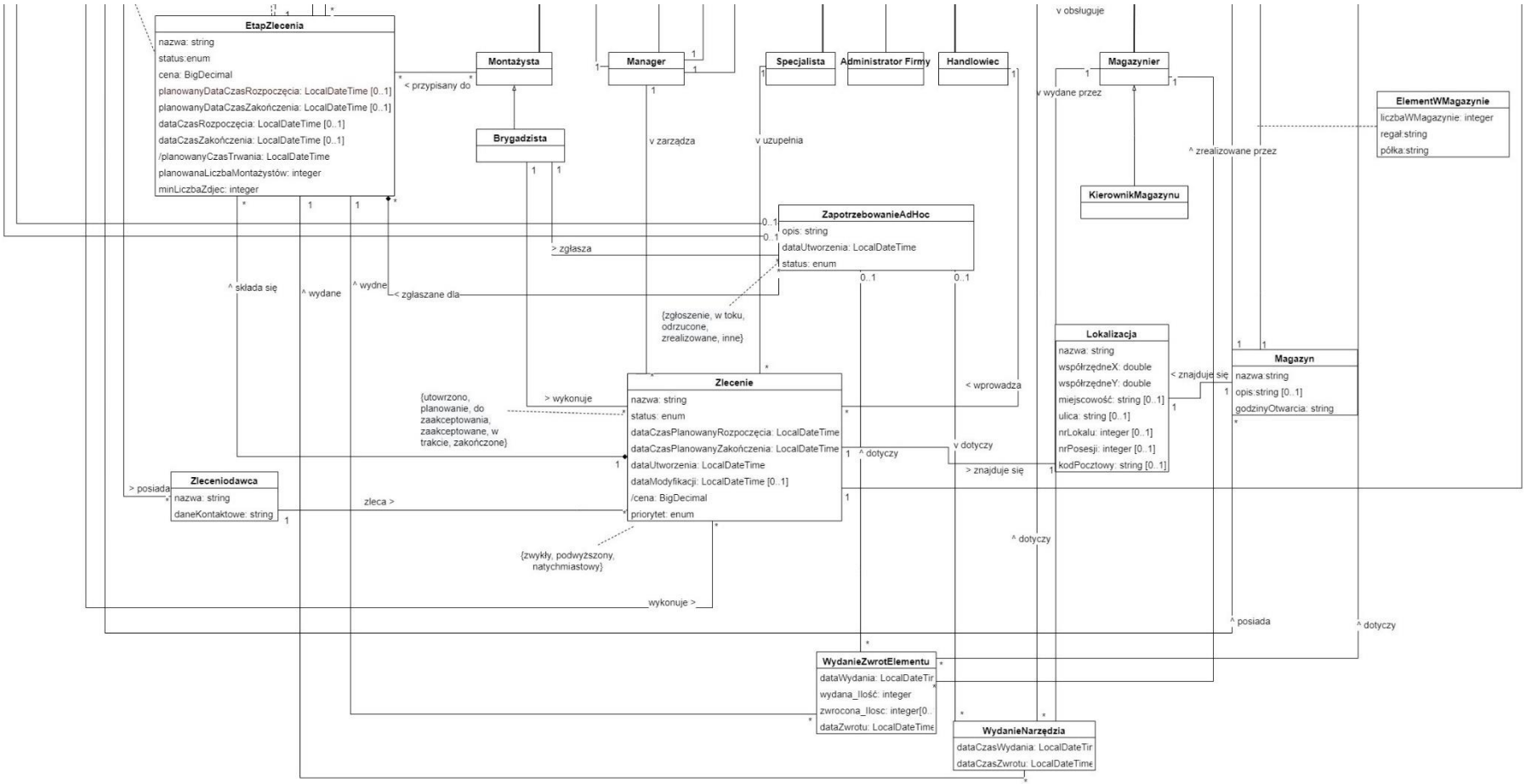
Rysunek 29 E-montażysta – diagram przypadków użycia dla System. Źródło: Opracowanie własne.

3.6. Diagram klas

W celu uporządkowaniu struktury projektu oraz zwiększenia klarowności zależności między klasami, został opracowany diagram klas. Przedstawia on hierarchię oraz relację pomiędzy poszczególnymi elementami systemu. Dzięki temu można w prosty sposób zobaczyć z jakich pól zbudowane są poszczególne klasy, jak są ze sobą powiązane oraz jakie metody i atrybuty posiadają. Było to szczególnie istotne ze względu na znaczną liczbę klas w projekcie. Rysunek 30 i 31 przedstawia górną oraz dolną część diagramu klasy.



Rysunek 30 E-montażysta – Diagram klas cześć górna. Źródło: Opracowanie własne.



Rysunek 31 E-montażysta – Diagram klas cześć dolna. Źródło: Opracowanie własne.

3.7. Moduły systemu

Podrozdział ten skupia się na przedstawieniu poszczególnych modułów systemu, które zostały zaprojektowania by skutecznie spełnić wymagania. Do najważniejszych z nich należą:

- Moduł 1: Zarządzanie firmami;
- Moduł 2: Zarządzanie kontami użytkowników;
- Moduł 3: Zarządzanie magazynami;
- Moduł 4: Obsługa zleceń;
- Moduł 5: Administracja firmy;

Wszystkie moduły systemu E-montażysta są ściśle zintegrowane i wzajemnie współpracują w celu zapewnienia pełnej funkcjonalności systemu. Każdy z nich spełnia określone zadania, tworząc spójną całość i umożliwiając realizację wymaganych funkcji systemu.

W kolejnych rozdziałach przedstawione zostaną bardziej szczegółowe opisy poszczególnych modułów oraz omówione będą ich implementacje i testowanie.

3.8. Wymagania funkcjonalne

Podczas serii spotkań zespołu analizy, przeprowadzonego w celu przeglądu potrzeb potencjalnych użytkowników, ujawniły się konkretne wymagania, które zostały przypisane do odpowiednich modułów, mających znaleźć się w systemie. W kolejnym etapie prac dokonano szczegółowego opisu tych modułów. W rezultacie powstały wymagania funkcjonalne systemu, które zostały przedstawione w niniejszym podrozdziale.

3.8.1 Moduł 1: Zarządzanie firmami

Moduł ten stanowi podstawową część systemu E-montażysta, umożliwiając efektywne zarządzanie danymi, które dotyczą firm korzystających z systemu. Jest on dostępny wyłącznie dla administratora chmury. W tabeli 2 przedstawiono szczegółowe wymagania funkcjonalne dotyczące tego modułu.

Tabela 2 Wymagania funkcjonalne, Moduł 1: Zarządzanie firmami. Źródło: Opracowanie własne

Lp.	Nazwa	Aktorzy	Opis	Ograniczenia
1.	Dodaj firmę	Administrator chmury	Pozwala na wprowadzenie nowej firmy do systemu, przy czym wymaga to jednoczesnego utworzenia konta administratora firmy.	
2.	Edytuj firmę	Administrator chmury	Pozwala na modyfikację danych zapisanych o firmie.	Firma istnieje w bazie danych
3.	Usuń firmę	Administrator chmury	Pozwala na usunięcie dodanej firmy	Firma istnieje w bazie danych

4.	Wyświetl listę firm	Administrator chmury	Pozwala na wyświetlenie listy dodanych firm.	W bazie danych istnieje co najmniej jedna firma

3.8.2 Moduł 2: Zarządzanie kontami użytkowników

Drugi moduł pełni istotną rolę w zapewnieniu bezpiecznego i efektywnego zarządzania kontami użytkowników w ramach systemu. Celem tego modułu jest umożliwienie administratorowi firmy pełnej kontroli nad kontami użytkowników oraz zapewnienie im narzędzi do wykonywania różnorodnych operacji związanych z zarządzaniem nimi. W tabeli 3 znajdują się szczegółowe wymagania dotyczące funkcjonalności zarządzania kontami użytkowników.

Tabela 3 Wymagania funkcjonalne, Moduł 2: Zarządzanie kontami użytkowników. Źródło: Opracowanie własne

Lp.	Nazwa	Aktorzy	Opis	Ograniczenia
1.	Dodaj konto	Administrator firmy	Pozwala na wprowadzenie nowego konta do systemu.	
2.	Edytuj konto	Administrator firmy	Pozwala na modyfikację danych zapisanych o koncie.	Konto istnieje w bazie danych
3.	Usuń konto	Administrator firmy	Pozwala na usunięcie dodanego konta.	Konto istnieje w bazie danych
4.	Wyświetl listę kont	Administrator firmy	Pozwala na wyświetlenie listy dodanych firm.	W bazie danych istnieje co najmniej jedno konto

3.8.3 Moduł 3: Zarządzanie magazynami

Trzeci moduł systemu, odgrywa kluczową rolę w skutecznym zarządzaniu magazynami w ramach systemu. Jego celem jest umożliwienie użytkownikom monitorowanie, organizowanie i kontrolowanie zasobów magazynowych (narzędzia i elementy) w sposób efektywny i zgodny z wymaganiami. Został zaprojektowany w taki sposób, aby usprawnić procesy magazynowe oraz zoptymalizować zarządzanie logistyką w systemie. Tabela 4 prezentuje szczegółowe wymagania funkcjonalne związane z zarządzaniem magazynami.

Tabela 4 Wymagania funkcjonalne, Moduł 3: Zarządzanie magazynami. Źródło: Opracowanie własne

Lp.	Nazwa	Aktorzy	Opis	Ograniczenia
1.	Dodaj magazyn	Administrator firmy	Pozwala na wprowadzenie nowego magazynu do systemu.	

2.	Edytuj magazyn	Administrator firmy	Pozwala na modyfikację danych zapisanych o magazynie.	Magazyn istnieje w bazie danych
3.	Usuń magazyn	Administrator firm	Pozwala na usunięcie dodanego magazynu.	Magazyn istnieje w bazie danych
4.	Wyświetl listę magazynów	Pracownik	Pozwala na wyświetlenie listy dodanych magazynów wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jeden magazyn
5.	Otwórz lokalizację do aplikacji nawigacyjnej	Pracownik	Pozwala na przekierowanie do wybranej aplikacji nawigacyjnej i wyznaczenie trasy za pomocą współrzędnych lokalizacji	
6.	Wyświetl listę elementów	Specjalista	Pozwala na wyświetlenie listy dodanych elementów w wybranym magazynie.	W bazie danych istnieje co najmniej jeden element w wybranym magazynie
7.	Wyświetl listę narzędzi	Specjalista	Pozwala na wyświetlenie listy dodanych narzędzi w wybranym magazynie.	W bazie danych istnieje co najmniej jedno narzędzie w wybranym magazynie
8.	Zgłoś usterkę narzędzia	Magazynier, Manager, Montażysta	Pozwala na raportowanie i rejestrowanie nieprawidłowości narzędzi	W bazie danych istnieje co najmniej jedno narzędzie
9.	Zgłoś usterkę elementu	Magazynier, Manager, Montażysta	Pozwala na raportowanie i rejestrowanie nieprawidłowości elementu	W bazie danych istnieje co najmniej jeden element
10.	Wyświetl listę zgłoszonych usterek	Pracownik	Pozwala na wyświetlenie listy dodanych usterek wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jedno zgłoszenie usterki
11.	Wyświetl historię narzędzia	Montażysta, Specjalista, Magazynier	Pozwala na wyświetlenie informacje na temat historii użycia wybranego narzędzia	W bazie danych istnieje co najmniej jedno narzędzie

12.	Zgłoś zapotrzebowanie ad hoc	Brygadzista	Pozwala na złożenie zlecenia na natychmiastowe dostarczenie zasobu (narzędzia lub elementu)	
13.	Zdaj narzędzie	Brygadzista	Pozwala na zwrócenie narzędzia do magazynu	
14.	Zdaj element	Brygadzista	Pozwala na zwrócenie narzędzia do magazynu	
15.	Wydaj element	Magazynier	Pozwala na wydanie elementu przypisanego do realizowanego zlecenia	Element został przypisany do zlecenia
16.	Wydaj narzędzie	Magazynier	Pozwala na wydanie narzędzia przypisanego do realizowanego zlecenia	Narzędzie zostało przypisane do zlecenia
17.	Przyjmij narzędzie	Magazynier	Pozwala na przyjęcie narzędzia	Narzędzie zostało wcześniej wydane
18.	Przyjmij element	Magazynier	Pozwala na przyjęcie elementu	Element został wcześniej wydany
19.	Dodaj narzędzie	Magazynier	Pozwala na wprowadzenie nowego narzędzie do systemu.	
20.	Usuń narzędzie	Magazynier	Pozwala na usunięcie dodanego narzędzia.	Narzędzie istnieje w bazie danych
21.	Edytuj narzędzie	Magazynier	Pozwala na modyfikację danych zapisanych o narzędziu.	Narzędzie istnieje w bazie danych
22.	Generuj etykietę	Magazynier	Pozwala na wygenerowanie i wyświetlenie kodu QR dla wybranego zasobu	Zasób istnieje w bazie danych
23.	Sprawdź stan magazynu	Magazynier	Pozwala na wyświetlenie zasobów wybranego magazynu.	W bazie danych istnieje co najmniej jeden magazyn i jeden zasób
24.	Usuń usterkę	Magazynier, Kierownik magazynu, Manager	Pozwala na usunięcie dodanej usterki.	Usterka istnieje w bazie danych

25.	Edytuj usterkę	Magazynier, Kierownik magazynu, Manager	Pozwala na modyfikację danych zapisanych o usterce.	Usterka istnieje w bazie danych
26.	Wyświetl listę typów narzędzi	Kierownik magazynu	Pozwala na wyświetlenie listy dodanych typów narzędzi wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jeden typ narzędzia
27.	Usuń typ narzędzia	Kierownik magazynu	Pozwala na usunięcie danego typu narzędzia	Typ narzędzia istnieje w bazie danych
28.	Edytuj typ narzędzia	Kierownik magazynu	Pozwala na modyfikację danych zapisanych o typie narzędzia.	Typ narzędzia istnieje w bazie danych
30.	Dodaj element	Magazynier	Pozwala na wprowadzenie nowego elementu do systemu.	
31.	Usuń element	Magazynier	Pozwala na usunięcie dodanego elementu.	Element istnieje w bazie danych
32.	Edytuj element	Magazynier	Pozwala na modyfikację danych zapisanych o elemencie.	Element istnieje w bazie danych
33.	Dodaj typ narzędzia	Kierownik magazynu	Pozwala na wprowadzenie nowego typu narzędzia do systemu.	
34.	Eksportuj do PDF instrukcję	Pracownik	Pozwala na pobranie instrukcji w formacie PDF.	
35.	Wyświetl listę typów elementu	Kierownik magazynu	Pozwala na wyświetlenie listy dodanych typów elementu wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jeden typ elementu
36.	Usuń typ elementu	Kierownik magazynu	Pozwala na usunięcie danego typu narzędzia	Typ elementu istnieje w bazie danych
37.	Edytuj typ elementu	Kierownik magazynu	Pozwala na modyfikację danych zapisanych o typie elementu.	Typ elementu istnieje w bazie danych
38.	Dodaj typ elementu	Kierownik magazynu	Pozwala na wprowadzenie nowego typu elementu do systemu.	
39.	Wyświetl listę wszystkich usterek	Kierownik magazynu	Pozwala na wyświetlenie listy wszystkich zgłoszonych usterek przez pracowników wraz z informacjami.	W bazie danych istnieje co najmniej jedna usterka

40.	Odbierz narzędzie	Brygadzista	Pozwala na oznaczenie narzędzia jako odebrane z magazynu.	W bazie danych istnieje co najmniej jedno narzędzie.
41.	Odbierz element	Brygadzista	Pozwala na oznaczenie elementu jako odebrane z magazynu.	W bazie danych istnieje co najmniej jedno element.

3.8.4 Moduł 4: Obsługa zleceń

W ramach systemu E-Montażysta zaimplementowano moduł obsługi zleceń, który odgrywa kluczową rolę w efektywnym zarządzaniu procesem realizacji zleceń oraz zapewnieniu skutecznej komunikacji między wszystkimi zaangażowanymi stronami. Ten moduł umożliwia śledzenie i kontrolę wszystkich etapów zleceń, zapewniając przejrzystość i efektywność w całym procesie. W tabeli 5 przedstawiono szczegółowe wymagania funkcjonalne dotyczące obsługi zleceń.

Tabela 5 Wymagania funkcjonalne, Moduł 4: Obsługa zleceń. Źródło: Opracowanie własne

Lp.	Nazwa	Aktorzy	Opis	Ograniczenia
1.	Wyświetl listę zleceń	Brygadzista, Specjalista, Handlowiec, Manager	Pozwala na wyświetlenie listy zleceń wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jedno zlecenie
2.	Usuń zlecenie	Brygadzista, Handlowiec	Pozwala na usunięcie dodanego zlecenia.	Zlecenie istnieje w bazie danych
3.	Edytuj zlecenie	Brygadzista, Handlowiec	Pozwala na modyfikację danych zapisanych o zleceniu.	Zlecenie istnieje w bazie danych
4.	Wyświetl listę etapów zlecenia	Specjalista, Manager, Montażysta	Pozwala na wyświetlenie listy etapów zlecenia wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jeden etap zlecenia
5.	Zmień status etapu zlecenia	Magazynier, Specjalista, Brygadzista	Pozwala na aktualizacje informacji dotyczących realizacji danego etapu zlecenia.	W bazie danych istnieje co najmniej jeden etap zlecenia
6.	Zmień status zlecenia	Specjalista, Handlowiec, Manager, Brygadzista	Pozwala na aktualizacje informacji dotyczących realizacji danego zlecenia.	W bazie danych istnieje co najmniej jedno zlecenie
7.	Usuń etap zlecenia	Specjalista	Pozwala na usunięcie dodanego etapu zlecenia.	Etap zlecenie istnieje w bazie danych

8.	Edytuj etap zlecenia	Specjalista	Pozwala na modyfikację danych zapisanych o etapie zleceniu.	Etap zlecenie istnieje w bazie danych
9.	Dodaj etap zlecenia	Specjalista	Pozwala na wprowadzenie nowego etapu zlecenia do systemu.	
10.	Dodaj zlecenie	Handlowiec	Pozwala na wprowadzenie nowego zlecenia do systemu.	
11.	Przypisz brygadzistę do zlecenia	Manager, Brygadzista	Pozwala na przypisanie konkretnego brygadzisty dla danego zlecenia.	Brygadzista oraz zlecenie istnieją w bazie danych
12.	Dodaj zdjęcie do etapu zlecenia	Montażysta	Pozwala na dołączenie graficznego materiału do dokumentacji postępu prac w ramach danego etapu zlecenia.	Etap zlecenia istnieje w bazie danych

3.8.5 Moduł 5: Administracja firmy

Moduł piąty poświęcony został profilom pracowników i klientów. Gromadzi oraz zarządza informacjami dotyczącymi zatrudnionych pracowników w organizacji. W tabeli 6 przedstawiono szczegółowe wymagania funkcjonalne dotyczące administracji firmy.

Tabela 6 Wymagania funkcjonalne, Moduł 5: Administracja firmy. Źródło: Opracowanie własne.

Lp.	Nazwa	Aktorzy	Opis	Ograniczenia
1.	Wyświetl powiadomienie	Użytkownik systemu	Pozwala na wyświetlenie powiadomień od innych pracowników wraz ze wszystkimi informacjami.	
2.	Wyświetl listę pracowników	Pracownik	Pozwala na wyświetlenie listy pracowników wraz ze wszystkimi informacjami	W bazie danych istnieje co najmniej jeden pracownik
3.	Wyświetl listę klientów	Handlowiec, Manager	Pozwala na wyświetlenie listy zleceń wraz ze wszystkimi informacjami.	W bazie danych istnieje co najmniej jeden klient
4.	Usuń klienta	Handlowiec, Manager	Pozwala na usunięcie dodanego klienta.	Klient istnieje w bazie danych
5.	Edytuj klienta	Handlowiec, Manager	Pozwala na modyfikację danych zapisanych o kliencie.	Klient istnieje w bazie danych

6.	Dodaj klienta	Handlowiec, Manager	Pozwala na wprowadzenie nowego klienta do systemu.	
7.	Sprawdź harmonogram pracownika	Manager, Brygadzista	Pozwala na uzyskanie informacji na temat planu pracy pracownika.	Pracownik istnieje w bazie danych
8.	Wprowadź nieobecność pracownika	Manager	Pozwala na rejestrację braku dostępności danego pracownika w określonym czasie.	Pracownik istnieje w bazie danych
9.	Sprawdź swój harmonogram pracy	Montażysta	Pozwala na uzyskanie informacji na temat swojego planu pracy.	
10.	Wyślij powiadomienie	System	Pozwala na przekazanie ważnej informacji do pracownika za pomocą systemu powiadomień.	
11.	Wyślij email powitalny	System	Pozwala na automatyczne wysłanie spersonalizowanej wiadomości powitalnej do nowego pracownika.	Pracownik istnieje w bazie danych

3.9. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne to właściwości dotyczące cech systemu, które nie są bezpośrednio związane z jego funkcjonalnością, ale mają wpływ w jaki sposób jest obsługiwany i jak spełnia oczekiwania użytkowników. Nie są to aspekty związane z funkcjami biznesowi, ale za to z skalowalnością, wydajnością, niezawodnością, bezpieczeństwem, dostępnością czy też łatwością rozszerzenia systemu. Wymagania niefunkcjonalne są ważne dla zapewnienia satysfakcji użytkownika oraz spełnienia określonych norm branży. Tabela 7 przedstawia szczegółowe wymagania niefunkcjonalne.

Tabela 7 Wymagania niefunkcjonalne. Źródło: Opracowanie własne.

Lp.	Aspekt	Opis
1.	Wydajność	System powinien być responsywny i dynamicznie dostosowywać się do różnych rozmiarów ekranu.
2.	Wydajność	System powinien zapewniać krótki czas odpowiedzi na żądania użytkowników, do 10 sekund.
3.	Skalowalność	System powinien być w stanie obsługiwać rosnącą liczbę użytkowników i ilość danych bez utraty wydajności.
4.	Bezpieczeństwo	System powinien zapewniać szyfrowanie danych w tym przede wszystkim haseł.
5.	Dostępność i Niezawodność	System powinien dostarczyć wysoką dostępność usługi, minimalizując ewentualne przerwy.

6.	Użyteczność	System powinien być obsługiwany w przeglądarce internetowej w języku polskim.
7.	Użyteczność	System powinien działać na systemach operacyjnych Windows oraz Android
8.	Wielofirmowość i izolacja chmurowa	System powinien umożliwiać obsługę wielu firm jednocześnie, zapewniając niezależną przestrzeń chmurową dla każdej z nich.

3.10. Scenariusze

W tym podrozdziale przedstawione są scenariusze, które opisują wybrane przypadki użycia. Scenariusze te przedstawiają konkretne sekwencje działań, które użytkownik może podjąć by zrealizować pewną interakcje z systemem. Ich celem jest zaprezentowanie funkcjonalności aplikacji w opisowych krokach.

Każdy scenariusz skupia się na opisie konkretnego przypadku użycia, od początku do końca, uwzględniając kroki podjęte przez użytkownika, oczekiwane rezultaty oraz ewentualne warunki i ograniczenia. Przebiegi te stanowią podstawę dla procesu projektowania, implementacji i testowania, umożliwiając precyzyjne określenie oczekiwanego zachowania platformy i rezultatów. Tabele 8 – 11 przedstawiają kolejno wybrane przypadki użycia: zgłoś usterkę, wprowadź informację o dostępności pracownika, zmień status etapu, sprawdź stan magazynu.

Tabela 8 E-montażysta – scenariusz przypadku użycia wprowadź informację o dostępności pracownika. Źródło: Opracowanie własne.

Nazwa przypadku użycia	
Warunki wstępne	Aktor Manager jest zalogowany w systemie, system wyświetlił listę pracowników
Warunki końcowe	
Aktor	
Przepływ zdarzeń	
Krok	**Akcja**
1	Aktor Manager rozpoczyna przypadek użycia
2	Aktor Manager wybiera pracownika z listy
3	System wyświetla szczegóły dotyczące pracownika
4	Aktor Manager wprowadza datę i godziny dostępności pracownika
5	System weryfikuje poprawność wprowadzonych danych
6a	System wyświetla prośbę o potwierdzenie wprowadzonych zmian
6aa	Aktor Manager zatwierdza wprowadzone zmiany i kończy przypadek użycia
6b	System wyświetla informację o błędnie wprowadzonych danych, wraca do kroku 3

Tabela 9 E-montażysta – scenariusz przypadku użycia zmień status etapu. Źródło: Opracowanie własne.

Nazwa przypadku użycia	Zmień status etapu zlecenia
Warunki wstępne	
Warunki końcowe	
Aktor	Brygadzista
Przepływ zdarzeń	
Krok	**Akcja**
1	Aktor Brygadzista rozpoczyna przypadek użycia
2	Aktor Brygadzista wybiera etap zlecenia
3	System wyświetla informacje o zleceniu (nazwa, aktualny status, kolejność, data rozpoczęcia /planowana data rozpoczęcia, data zakończenia/planowany czas trwania) i prosi o wybranie aktualnego statusu zlecenia
4	Aktor Brygadzista wybiera status etapu i zatwierdza wybór

Tabela 10 E-montażysta – scenariusz przypadku użycia sprawdź stan magazynu. Źródło: Opracowanie własne.

Nazwa przypadku użycia	**Sprawdź stan magazynu**
Warunki wstępne	Aktor jest zalogowany w systemie, ma rolę Magazynier
Warunki końcowe	Wyświetlona lista elementów i narzędzi na magazynie
Aktor	Magazynier
Przepływ zdarzeń	
Krok	**Akcja**
1	Magazynier uruchamia przypadek użycia
2	System wyświetla listę przypisanych magazynów
2a	System wybiera jedyny magazyn przypisany do Magazyniera i przechodzi do punktu 4
2b	System wyświetla informację o braku przypisanych magazynów i kończy przypadek użycia
3	Magazynier wybiera magazyn
4	System wyświetla listę stanów magazynowych i kończy przypadek użycia

4. Technologie i narzędzia wykorzystane w projekcie

Poniższy rozdział przedstawia technologie i narzędzia, które zostały wykorzystane w realizacji projektu. Opisane zostały główne komponenty i platformy, które odegrały kluczową rolę w tworzeniu oprogramowania. Celem tego rozdziału jest zaprezentowanie środowiska technologicznego, w którym projekt został zrealizowany oraz uzasadnienie wyboru konkretnych rozwiązań.

4.1. Java 17

Język Java według [17] to obiektowy język programowania, opracowany i zaprojektowany w 1995 roku przez firmę Sun Microsystems, obecnie zwaną Oracle Corporation. Został stworzony w oparciu o koncepcje dwóch już istniejących języków tj. Smalltalk oraz C++. Od pierwszego z nich zaczerpnął pomysł na tworzenie aplikacji, które będą działać niezależnie od platformy sprzętowej i systemu operacyjnego, dzięki użyciu maszyny wirtualnej. Dało to możliwość uruchamiania kodu na różnych platformach bez konieczności jego modyfikacji. Natomiast z języka C++ została zaczerpnięta znaczna część składni języka Java.

Zadaniem wspomnianej w powyższym akapicie maszyny wirtualnej Java (JVM) jest dostarczenie środowiska, w którym wykona się program skompilowany do kodu zrozumiałego dla danego systemu operacyjnego. Dodatkowo JVM zapewnia takie usługi jak automatyczna alokacja pamięci czy zarządzanie wyjątkami co zwiększa bezpieczeństwo i niezawodność programu. Platforma Java

Java posiada wiele bibliotek oraz frameworków umożliwiających tworzenie aplikacji internetowych, desktopowych czy mobilnych. Wpływają one na wzrost produktywności programistów, którzy za ich pomocą mogą szybko i łatwo tworzyć wydajne aplikacje. Dodatkowo dostarczają one narzędzia do rozwiązywania powszechnych problemów powstających przy pracy z językiem Java.

Programowanie obiektowe oznacza, że cały kod jest zorganizowany wokół koncepcji obiektów, czyli instancji klasy, które definiuje strukturę i zachowanie obiektu. Dziedziczenie jest podstawowym elementem podejścia obiektowego w Javie. Klasy mogą dziedziczyć po innych klasach, co pozwala na utworzenie ich hierarchii i ponowne wykorzystanie napisanego kodu. Polimorfizm to kolejny przydatny element podejścia obiektowego w Javie. Umożliwia tworzenie wielu różnych typów obiektów, które zachowują się podobnie, ale wykonują różne operacje. Zapewnia to elastyczność w tworzeniu i utrzymaniu programu.

Jedną z głównych zalet języka Java jest sposób, w jaki sposób został zaprojektowany, co przekłada się na jego bezpieczeństwo, łatwość w pisania kodu oraz niezależność od systemu operacyjnego. Często wyróżnianą wadą jest przede wszystkim wydajność. Wyniki przeprowadzonych testów pokazują, że w większości przypadków jest ona wolniejsza niż inne języki programowania. Pomimo wszystko język Java ma wiele zastosowań w różnych dziedzinach i z pewnością będzie się rozwijał wraz z postępem technologicznym.

4.2. Spring

Spring według [18] to framework do tworzenia aplikacji internetowych w języku Java. Jego głównym celem jest ułatwienie procesu złożonych systemów poprzez zapewnienie zestawu narzędzi i bibliotek. To podejście ogranicza możliwe problemy techniczne związane z tworzeniem aplikacji od podstaw. Spring składa się z wielu osobnych modułów, co umożliwia wybór tylko tych elementów, które są potrzebne w danym projekcie. Pozwala to na optymalizację rozmiaru aplikacji i uniknięcie

przeciążenia zbędnymi modułami. Spring jest powszechnie wykorzystywany przez wiele renomowanych marek, włączając w to Allegro.pl, Netflix czy Volkswagen.

Dużą zaletą Springa jest skalowalność. Dzięki temu, aplikację można łatwo dostosowywać w przypadku zmian wymagań biznesowych. Inną ważną zaletą Springa jest łatwość testowania kodu. Za sprawą wbudowanych narzędzi, można bezproblemowo pisać testy jednostkowe, integracyjne czy funkcjonalne, co pozwala na szybkie wykrycie i naprawę błędów. Spring dzięki wbudowanym mechanizmom autoryzacji i uwierzytelnienia zapewnia również wsparcie w obsłudze różnych zagrożeń bezpieczeństwa, co jest szczególnie istotne w przypadku aplikacji biznesowych przechowywujących dane osobowe. Warto również wspomnieć o tym, że Spring jest frameworkiem z otwartym kodem źródłowym, co oznacza, że jest rozwijany przez społeczność na całym świecie. Dzięki temu Spring może być stale udoskonalany do najnowszych technologii i rozwiązań.

Spring to bardzo rozbudowany framework. Zawiera w sobie wiele komponentów i bibliotek, umożliwiających pisanie zaawansowanych aplikacji. Wśród najważniejszych komponentów Springa można wymienić między innymi:

- Spring Boot – rozwiązanie bazujące na Springu, które dostarcza gotowe szablony i konfigurację umożliwiając przy tym łatwe konfigurowanie i zarządzanie zależnościami aplikacji. W ramach pracy zastosowano SpringBoot 2.7.6.
- Spring MVC - framework służący do tworzenia aplikacji webowych opartych o wzorzec MVC (Model-View-Controller). Umożliwia łatwe tworzenie kontrolerów i widoków. Zapewnia mechanizmy do obsługi formularzy, walidacji danych oraz zarządzania sesją użytkownika. Kontroler jest w stanie obsługiwać żądania dla każdej z metod HTTP, co stanowi fundament usług internetowych typu RESTful.
- Spring Security – narzędzie służące do zapewnienia bezpieczeństwa aplikacji. Umożliwia autoryzację i uwierzytelnianie użytkowników, obsługę sesji oraz zarządzanie uprawnieniami. Ponadto daje możliwość integracji z różnymi mechanizmami uwierzytelniania tj. LDAP czy OAuth.

Jak każdy inny framework, Spring posiada pewne wady i ograniczenia. Przez to, że pozwala na szybkie tworzenie i uruchamianie aplikacji, może prowadzić do nadmiernego użycia pamięci co przełoży się na dłuższy czas startu.

4.3. Swagger

Swagger według [19] to popularne narzędzie do dokumentowania API RESTful. Zapewnia standardowy format opisywania punktów końcowych interfejsu API, parametrów, odpowiedzi i metod uwierzytelniania. Dzięki Swaggerowi programiści mogą łatwo udostępniać dokumentację API innym członkom zespołu lub użytkownikom zewnętrznym. Swagger oferuje również szereg narzędzi do generowania kodu klienckiego i testowania API, co czyni go cennym narzędziem do budowania i utrzymywania interfejsów API. Wreszcie Swagger stał się powszechnie akceptowanym standardem branżowym dokumentacji API, ułatwiając programistom pracę z interfejsami API różnych dostawców. Swagger został pierwotnie stworzony przez Tony'ego Tama w 2011 roku jako projekt open source do dokumentowania i testowania interfejsów API. Szybko zyskał popularność wśród programistów i został przejęty przez SmartBear Software w 2015 roku. Później nazwa narzędzia została zmieniona na OpenAPI, ale marka Swagger jest nadal powszechnie używana w specyfikacji OpenAPI i powiązanych narzędziach.

Swagger składa się z dwóch elementów:

- Swagger Editor: Jest to internetowy edytor służący do projektowania i dokumentowania interfejsów API. Zapewnia przyjazny dla użytkownika interfejs dla programistów do definiowania

struktury ich API, w tym punktów końcowych, metod, parametrów, odpowiedzi i innych szczegółów. Umożliwia także programistom testowanie ich API poprzez wysyłanie żądań i otrzymywanie odpowiedzi.

- Swagger UI: Jest to narzędzie internetowe, które służy do generowania interaktywnej dokumentacji dla interfejsów API. Zapewnia przyjazny dla użytkownika interfejs dla programistów do eksploracji i testowania punktów końcowych API. Umożliwia także programistom generowanie klienckich zestawów SDK w różnych językach programowania, w tym Java, JavaScript, Python, Ruby i innych.

Swagger działa przy użyciu specyfikacji OpenAPI, która jest standardowym formatem definiowania interfejsów API RESTful. Specyfikacja OpenAPI zawiera zestaw reguł i wskazówek dotyczących definiowania struktury API, w tym punktów końcowych (endpointy), metod, parametrów, odpowiedzi i innych szczegółów. Swagger używa specyfikacji OpenAPI do generowania dokumentacji i klienckich zestawów SDK dla interfejsów API.

Zalety korzystania z API Swaggera:

1. Dokumentacja: Swagger API zapewnia łatwą w użyciu dokumentację API, która pomaga programistom zrozumieć i używać API.

2. Interoperacyjność: interfejs Swagger API zapewnia standardowy format dokumentacji RESTful API, ułatwiając programistom pracę z interfejsami API różnych dostawców.

3. Generowanie kodu — Swagger API oferuje szereg narzędzi do generowania kodu klienckiego i kodów pośredniczących serwera dla różnych języków programowania, co może zaoszczędzić czas i wysiłek programistów.

4. Testowanie — Swagger API zapewnia zintegrowane środowisko testowe, które umożliwia programistom łatwe i wydajne testowanie ich interfejsów API.

Wady korzystania z API Swaggera:

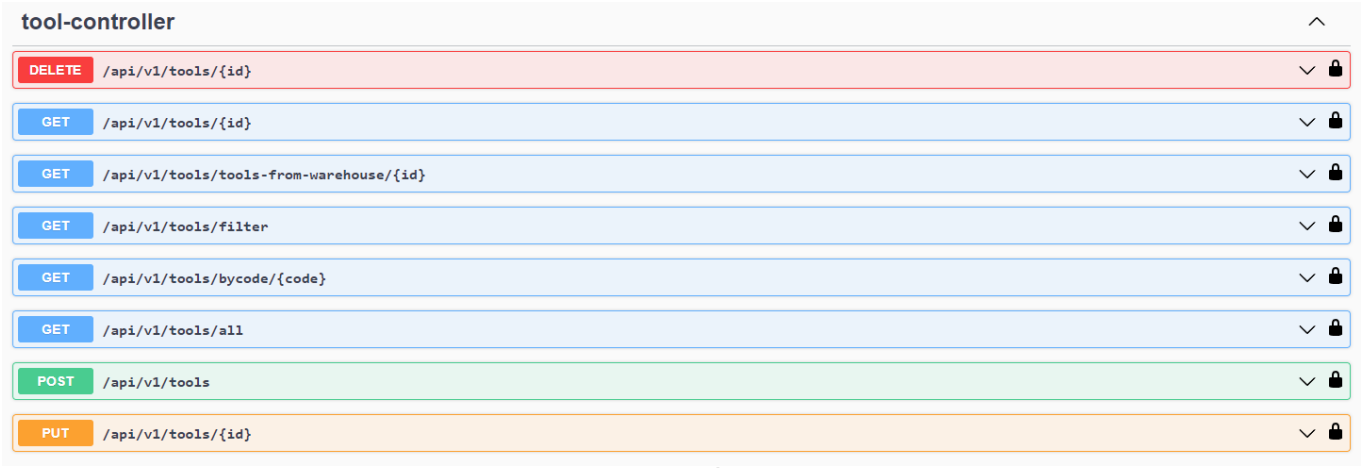
1. Krzywa uczenia się — korzystanie z interfejsu API Swagger wiąże się z krzywą uczenia się, do której przyzwyczajenie się może zająć trochę czasu i wysiłku.

2. Ograniczona obsługa interfejsów API innych niż RESTful: Interfejs API Swagger jest przeznaczony przede wszystkim dla interfejsów API RESTful i może nie być odpowiedni dla interfejsów API innych niż RESTful.

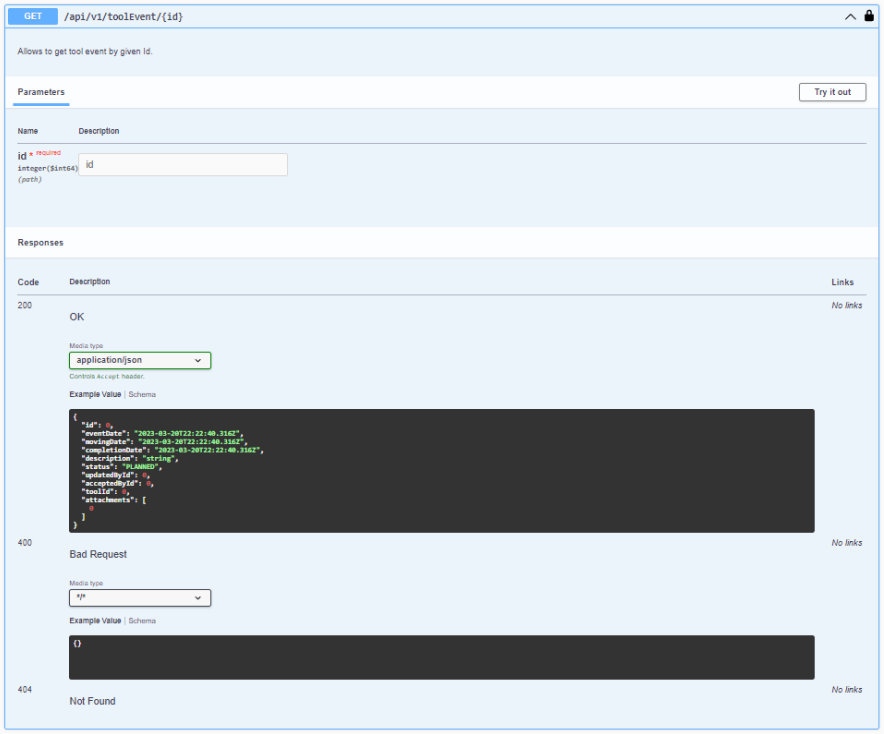
3. Ograniczona obsługa złożonych interfejsów API — interfejs API Swagger może nie być w stanie obsłużyć złożonych interfejsów API z zaawansowanymi funkcjami, takimi jak niestandardowe uwierzytelnianie lub parametry dynamiczne.

4. Konserwacja: Aktualizowanie dokumentacji Swagger API może być czasochłonne i może wymagać od programistów ręcznej aktualizacji dokumentacji przy każdej zmianie API.

Rysunek 32 przedstawia zestaw punktów końcowych (endpointów) jednej z klas aplikacji E-Montażysta, a na rysunku 33 widoczne jest opis parametrów i odpowiedzi na żądania dla jednego z punktów końcowych



Rysunek 32 E- montażysta - zestaw endpointów. Źródło: Opracowanie własne.



Rysunek 33 E- montażysta – przykład punktu końcowego. Źródło: Opracowanie własne.

4.4. Hibernate

Hibernate według [20] jest biblioteką, która umożliwia mapowanie obiektowo-relacyjne (ORM / Object Relational Mapping). Znajduję się pomiędzy tradycyjnymi obiektami języka Java a serwerami bazy danych. Opisywanie struktur w ten sposób jest możliwe poprzez konfigurację pliku XML lub adnotacje języka Java. Do najważniejszych założeń tego narzędzia należy:

• Umożliwienie komunikacji aplikacji z bazą danych, poprzez automatyczne odwzorowanie klas na tabele;

• Rozwiązanie problemu utrwalania danych na okres dłuższy niż proces aplikacji;

• Redukcja linii kodu i kosztów utrzymania w porównaniu do ręcznej obsługi trwałych danych;

• Pomoc w rozwiązaniu problemu niezgodności impedancji obiektowo-relacyjnej;

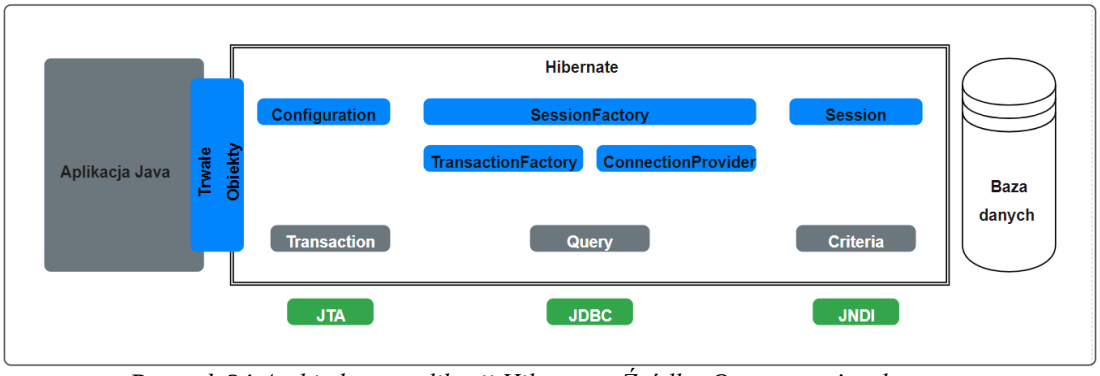
• Wysoka wydajność i mobilność;

Dodatkowo świadczy wiele funkcji, które pozwalają zoptymalizować i usprawnić pracę z bazą danych, oraz zmniejszyć udział programisty w procesie wytwarzania oprogramowania.

Hibernate powstał z inicjatywy Gavina Kinga w 2001 r. jako alternatywa dla EJB2, której pierwotnym celem było zaoferowanie lepszych możliwości utrwalania danych. W ciągu prawie dwudziestu lat od pierwszego wydania, wciąż otrzymuje poparcie społeczności i regularnie wydaje nowe wersje. Obecnie jest rozwijany przez firmę Red Hat. Usługa ta jest objęta licencją typu open source i można ją pobrać bezpłatnie. Biblioteka znalazła swoje przeznaczenie również dla platformy .NET Microsoftu i nosi nazwę NHibernate.

W języku Java istnieje interfejs API, który upraszcza proces łączenia aplikacji z zewnętrznymi, relacyjnymi systemami zarzadzania bazami danych (RDBMS / Relational Database Management System), znany pod nazwą JDBC (Java Database Connectivity). Reprezentuję zdecydowanie inne podejścia do pracy z trwałymi danymi. JDBC jest jedynie interfejsem API, więc nie wykonuje struktury mapowania obiektowo-relacyjnego. Z drugiej strony Hibernate umożliwia pisanie aplikacji Java opartych na bazach danych przy użyciu normalnej, obiektowej semantyki. Nie eliminuje to jednak potrzeby korzystania z JDBC, ponieważ Hibernate jest zbudowany na jego bazie, więc w rzeczywistości wszystkie operacje nadal oparte są o ten interfejs.

Hibernate ma architekturę warstwową, która pomaga działać bez konieczności znajomości bazowych interfejsów API. Rysunek 34 przedstawia szczegółowy widok architektury aplikacji Hibernate z jej podstawowymi klasami.



Rysunek 34 Architektura aplikacji Hibernate. Źródło: Opracowanie własne

Hibernate zawiera dane konfiguracyjne w pliku XML tj. hibernate.cfg.xml lub w pliku właściwości hibernate.properties. Klasa Configuration umożliwia określenie właściwości i mapowań dla aplikacji. Tworzy ona niezmienny interfejs SessionFactory, który jest używany do utworzenia obiektu klasy Session. Obiekt tej klasy jest głównym narzędzie do komunikacji z Hibernate, ponieważ zapewnia interfejs API, który umożliwia tworzenie, odczytywania, aktualizowania oraz usuwanie trwałych obiektów. Interfejs Transaction pozwala aplikacji na zdefiniowanie jednostki pracy, przy jednoczesnym zachowaniu abstrakcji od implementacji transakcji np. JTA czy JDBC.

HQL (Hibernate Query Language) to zorientowana obiektowo wersja SQL. Generuje niezależne zapytania do bazy danych, w taki sposób, że w zapytaniu używane są nazwy klas Java i atrybuty. Wewnętrznie Hibernate konwertuje HQL na SQL, a następnie wykonuje go na bazie danych. Interfejs Query udostępnia zorientowane obiektowo metody i umożliwia prezentacje i manipulowanie zapytaniami HQL.

Hibernate framework stanowi implementacje Java Persistence API (JPA). Wydana 11 maja 2006 r, JPA to specyfikacja określająca sposób uzyskiwania dostępu, zarządzania i utrwalania danych między obiektami języka Java a relacyjnymi bazami danych. Hibernate może być użyty na dowolnym środowisku obsługującym JPA.

Hibernate jest wciąż bardzo przydatnym narzędziem używanym w aplikacjach korporacyjnych, który ułatwia programistom języka Java sprawną komunikacje z bazami danych, bez konieczności pisania skomplikowanego kodu SQL. Nie ma potrzeby ręcznego tworzenia tabel ani ręcznego mapowania obiektów na wiersze w tabelach. Hibernate generuję to wszystko automatycznie, co zmniejsza nakład prac przy tym procesie i pozwala na skupienie się na innych aspektach aplikacji. Dodatkowo oferuje także szereg funkcji, które optymalizują i pozwalają na efektywniejsze wykorzystanie bazy danych co wiąże się z poprawą wydajności aplikacji. Dzięki swojej prostocie i elastyczności może być dostosowany do potrzeb rozmaitych projektów.

4.5. Lombok

Lombok według **Błąd! Nie można odnaleźć źródła odwołania.** to biblioteka Javy, która oferuje z estaw adnotacji umożliwiających automatyczne generowanie kodu dla wielu powtarzalnych czynności. Tworzenie getterów i setterów, konstruktorów, equals i hashCode, a nawet metod toString.

Użycie tej biblioteki pozwala na redukcja powtarzalnego kodu (np. getterów i setterów), co zwiększa szybkość programowania i wydajność tworzenia projektów. Kod jest przez to bardziej zwięzły i przejrzysty, czym pomaga w analizie w szczególności dla początkujących programistów.

Zaletą Lomboka jest również łatwość integracji, może być łatwo zintegrowany z różnymi narzędziami programistycznymi, takimi jak IDE, Maven, Gradle, Eclipse.

Podsumowując biblioteka Lombok jest popularną biblioteką programistyczną dla języka Java, która oferuje wiele narzędzi ułatwiających i przyspieszających pisanie kodu. Obejmuje w swoim zakresie to automatyczne generowanie kodu, łatwe zarządzanie adnotacjami, wstrzykiwanie zależności, obsługę wyjątków, walidację danych i wiele innych funkcjonalności. Dzięki tym mechanizmom programiści mogą pisać bardziej czytelny i zwięzły kod, co przekłada się na większą wydajność i jakość aplikacji.

4.6. Rest Api

REST API (Representational State Transfer Application Programming Interface) według **Błąd! Nie można odnaleźć źródła odwołania.** to rodzaj interfejsu programowania aplikacji, który wykorzystuje standardowe protokoły HTTP do komunikacji między aplikacjami. Opiera się on na architekturze REST, która opisuje sposoby komunikacji i działania aplikacji. Została ona opisana w pracy doktorskiej zatytułowanej "Architectural Styles and the Design of Network-based Software Architectures" amerykańskiego informatyka i naukowca Roya Fieldinga, definiuje ona zasady REST i ma duży wpływ na rozwój architektury sieciowej. Pracował on również nad projektem serwera HTTP Apache, który jest jednym z najczęściej używanych serwerów internetowych na świecie.

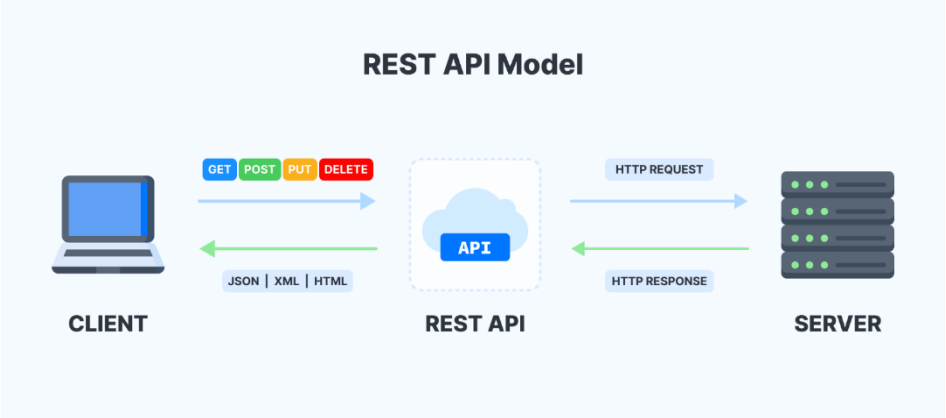
Application Programming Interface – jest to interfejs programowania aplikacji, zawiera zestaw definicji i protokołów służących do budowania i integracji oprogramowani aplikacji. Potocznie możemy nazwać umową między dostawcą danych, a użytkownikiem tych danych, która określa treść wymaganą do przesłania żądania oraz treść wymaganą jako odpowiedź na to żądanie.

Przykładowo API dla usługi pogodowej, może określać, że użytkownik podaje kod pocztowy, a dostawca odpowiada w ustalonym formacie dane o pogodzie, np. temperaturę minimalną i maksymalną

oraz szansę procentową opadów w danym dniu. API jest więc takim pośrednikiem między aplikacjami klienckimi a serwerem. Rysunek 35 ilustruję przedstawiony w tym akapicie przykład.

Przykładowy Scenariusz może wyglądać następująco:

1. Użytkownik lub aplikacja (klient) wysyła żądanie do API.
2. API odbiera żądanie, przetwarza je i przekazuje do serwera aplikacyjnego.
3. Serwer wysyła odpowiedź do API.
4. API przekazuje otrzymane informacje do klienta, który wysłał żądanie.



Rysunek 35 Przykład działania REST API. Źródło:[5]

Representational State Transfer (REST) to styl architektoniczny, który określa zasady, jakimi powinno się kierować podczas projektowania interfejsów programistycznych API. REST API powinno zostać zaprojektowane zgodnie z sześcioma regułami lub inaczej warunkami. Pierwszym z tych warunków jest jednolity interfejs, który zapewnia spójność komunikacji pomiędzy klientem a serwerem. Oznacza to, że żądania API dotyczące tego samego zasobu powinny wyglądać identycznie, niezależnie od ich źródła. Drugim warunkiem jest architektura klient-serwer, która polega na całkowitym oddzieleniu klienta od serwera. Aplikacje muszą być niezależne od siebie, a klient powinien znać URI dla żądanego zasobu i nie powinien się komunikować z aplikacją serwera w żaden inny sposób. Trzeci warunek to cacheability, czyli buforowanie zasobów po stronie klienta lub serwera. Odpowiedzi serwera powinny również zawierać informacje o tym, czy buforowanie jest dozwolone dla danego zasobu, co zwiększa wydajność po stronie klienta i skalowalność serwera. Czwartym warunkiem jest bezstanowość, czyli statelessness. Każde żądanie musi zawierać wszystkie informacje potrzebne do jego przetworzenia i jest całkowicie niezależne od innych żądań. Dane związane z żądaniami nie są przechowywane po stronie serwera. Piąty warunek to architektura wielowarstwowa (layered system), która zapewnia, że klient nie jest w stanie stwierdzić, czy komunikuje się z pośrednikiem czy bezpośrednio z serwerem. Dzięki temu zwiększa się bezpieczeństwo i można zastosować warstwę balansującą ruch pomiędzy różnymi serwerami. Wszystkie te reguły pozwalają na tworzenie API, które jest łatwe w obsłudze i zapewnia spójność w komunikacji pomiędzy klientem a serwerem.

4.7. JUnit

JUnit według **Błąd! Nie można odnaleźć źródła odwołania.** to popularna platforma testów j ednostkowych dla języka programowania Java, która umożliwia programistom pisanie i uruchamianie automatycznych testów w celu zapewnienia poprawności ich kodu. JUnit stał się branżowym

standardem testów jednostkowych w Javie i jest szeroko stosowany przez programistów i organizacje na całym świecie. Platforma posiada aktywną społeczność programistów, którzy przyczyniają się do jej rozwoju, zapewniają wsparcie i zasoby oraz dzielą się najlepszymi praktykami.

JUnit został po raz pierwszy wprowadzony w 1997 roku przez Kenta Becka i Ericha Gammę w ramach ich prac nad metodologią Extreme Programming (XP). Pierwsza wersja JUnit, JUnit 1.0, zawierała tylko kilka podstawowych instrukcji i przebieg testowy. W 2000 roku wydano JUnit 2.0 z wieloma nowymi funkcjami, w tym obsługą zestawów testów, testów sparametryzowanych i niestandardowych asercji. Ta wersja JUnit stała się jeszcze bardziej popularna wśród programistów Java i stała się branżowym standardem testów jednostkowych w Javie. W 2005 roku wydano JUnit 4.0 z istotnymi zmianami, w tym wprowadzeniem adnotacji i możliwością uruchamiania testów w dowolnej kolejności. Poprawił również organizację urządzeń testowych i wprowadził koncepcję „rozwoju sterowanego testami" (TDD), która zachęca programistów do pisania testów przed napisaniem kodu. W 2017 roku wydano JUnit 5 m.in. z obsługą wyrażeń lambda Java 8 oraz odwołań do metod, rozszerzeń i testów sparametryzowanych. JUnit 5 wprowadził również nową architekturę, która pozwala na większą elastyczność.

JUnit zapewnia prostą i elastyczną platformę do pisania i uruchamiania testów. Zawiera zestaw adnotacji, które umożliwiają programistom określanie metod testowych, urządzeń testowych i asercji. Platforma zapewnia również narzędzie do uruchamiania testów automatycznych, z których można raportować wyniki. Dostępnych jest wiele zasobów do nauki i używania JUnit, w tym dokumentacja, samouczki i przykładowy kod. JUnit jest potężnym i ważnym narzędziem dla programistów Java, którzy chcą zapewnić poprawność swojego kodu i budować wysokiej jakości oprogramowanie.

JUnit jest łatwym w użyciu frameworkiem, który upraszcza proces pisania i wykonywania testów jednostkowych w Javie. Bezproblemowo integruje się z innymi narzędziami do testowania, takimi jak Maven, Gradle i Eclipse, umożliwiając programistom łatwą integrację z ich przepływem pracy programistycznej. JUnit stał się branżowym standardem testów jednostkowych w Javie, dzięki czemu jest szeroko rozpoznawalnym i szanowanym frameworkiem. Platforma promuje stosowanie programowania sterowanego testami (TDD), które zachęca programistów do pisania testów przed napisaniem kodu, co prowadzi do bardziej niezawodnego i łatwiejszego w utrzymaniu kodu.

Z drugiej strony, JUnit został zaprojektowany specjalnie do testów jednostkowych i nie obsługuje innych rodzajów testów, takich jak testy integracyjne lub testy kompleksowe. Jest dedykowany językowi programowania Java i nie jest kompatybilny z innymi językami programowania. Może być ograniczony pod względem elastyczności w porównaniu do bardziej zaawansowanych i potężnych platform testowych. Wymaga wstępnej konfiguracji, co może być czasochłonne dla programistów niezaznajomionych z frameworkiem. Podobnie jak w przypadku każdej platformy testowej, utrzymanie zestawu testów może stać się wyzwaniem wraz z rozwojem i ewolucją kodu. Listing 1 przedstawia przykładowy test JUnit.

Listing 1 E- montażysta – przykładowy test JUnit. Źródło opracowanie własne.

```
@Test(timeout = mxWait)
public void NumEntriesIsCorrect() throws SittulationException {
     testLog = new WaterLog(windowsSizeSmall, mxEntryLarge);
     // Add entries up to, but not exceeding, the window size
     for (Integer nwrEntered 1; numEntered <= windowSizezSmall;
numEntered++) {
          // Add an entry
          testLog.addEntry(smalllEntryOne) ;
          // Confirm that rumber entered equals number reported by
the log
          assertEquaIs(numEntered, testLog.rufitries()) ; }}
```

4.8. Kotlin

Według **Błąd! Nie można odnaleźć źródła odwołania.** Kotlin jako statyczny język p rogramowania, stawiający na wieloplatformowość, został opracowany przez firmę JetBrains, a znakiem handlowym zarządza Kotlin Foundation. Został zaprojektowany do współpracy z Javą, umożliwiając łączenie obu języków w ramach jednego projektu opartego na JVM.

Inferencja typów pozwala na zwięzłą składnię i umożliwia kompilację do języka JavaScript lub kodu natywnego binarnego za pomocą kompilatora LLVM. Dzięki temu Kotlin może być uruchamiany na urządzeniach bez obsługi JVM.

Kotlin i Java różnią się od siebie pod wieloma względami. Oto najważniejsze różnice między nimi:

- Null Safety: Kotlin wymusza bezpieczną obsługę zmiennych i obiektów typu nullable, eliminując błędy związane z niezainicjalizowanymi wartościami.
- Rozszerzenia (Extension Functions): Pozwalają one na rozszerzanie klasy o nowej funkcji bez potrzeby dziedziczenia lub modyfikowania istniejącego kodu.
- Wsparcie dla korutyn (Coroutines): Kotlin oferuje mechanizm korutyn, umożliwiający wykonywanie zadań asynchronicznych bez potrzeby tworzenia osobnych wątków.
- Brak checked exceptions: W przeciwieństwie do Javy, Kotlin nie wymaga obsługi wyjątków typu checked.
- Klasy danych (Data Classes): Kotlin posiada specjalny typ klas, który ułatwia definiowanie klas używanych tylko do przechowywania danych.
- Programowanie funkcyjne: Kotlin posiada rozbudowane wsparcie dla programowania funkcyjnego, co ułatwia pisanie bardziej zwięzłego kodu.
- Zwięzłość kodu: Kotlin wprowadza skróty i uproszczenia składniowe, co prowadzi do bardziej czytelnej i zwięzłej struktury kodu.

Zastosowanie Kotlina jest bardzo szerokie, a jego zalety wynikające z charakterystycznych cech sprawiają, że znajduje on coraz większe zastosowanie w różnych technologiach tj. Android, iOS, aplikacje desktopowe i internetowe.

Kotlin znajduje zastosowanie w różnych technologiach, takich jak Android, iOS, aplikacje desktopowe i internetowe. Jego zalety wynikające z charakterystycznych cech przyczyniają się do jego rosnącej popularności wśród programistów. Oferuje rozbudowaną dokumentację oraz narzędzia ułatwiające pracę, zapewnia bezpieczeństwo przed nullami i redukuje występowanie błędów w czasie wykonania. Kotlin jest również uważany za język mniej podatny na błędy niż Java, co przekłada się na łatwiejsze utrzymanie kodu i niższe koszty związane z jego naprawą.

Pod względem idei i celów Kotlin jest podobny do języka Swift stosowanego w tworzeniu aplikacji na macOS i iOS.

4.9. Rest assured

Rest-assured **Błąd! Nie można odnaleźć źródła odwołania.** to biblioteka oparta na Javie, która z apewnia prosty i intuicyjny interfejs do interakcji z usługami internetowymi RESTful. Została zaprojektowana w celu uproszczenia procesu testowania i sprawdzania poprawności interfejsów API REST, oferując zestaw zaawansowanych i elastycznych funkcji. Rest-assured można łatwo zintegrować z popularnymi frameworkami testowymi w projektach opartych na Javie. Niektóre z kluczowych funkcji Rest-assured obejmują obsługę różnych metod HTTP, uwierzytelnianie i autoryzację, rejestrowanie

żądań i odpowiedzi oraz obsługę ładunków JSON i XML. Jest przydatnym narzędziem dla programistów i testerów, którzy chcą zautomatyzować testowanie i sprawdzanie poprawności interfejsów API REST.

Rest-assured to stosunkowo nowa biblioteka w ekosystemie Java, której pierwsze wydanie miało miejsce w 2010 roku. Początkowo biblioteka została stworzona przez Johana Haleby'ego, inżyniera oprogramowania ze Szwecji, który był sfrustrowany istniejącymi opcjami testowania usług sieciowych RESTful. Haleby rozpoczął pracę nad Rest-assured jako projekt osobisty, którego celem było stworzenie biblioteki, która uprościłaby testowanie i sprawdzanie poprawności interfejsów API REST w projektach opartych na Javie. Wypuścił pierwszą wersję Rest-assured w sierpniu 2010 roku, która szybko zyskała popularność wśród programistów i testerów.

Rest-assured jest biblioteką opartą na Javie stworzoną do testowania i sprawdzania poprawności usług internetowych RESTful. Biblioteka oferuje szereg metod i funkcji, które pozwalają programistom pisać testy dla żądań i odpowiedzi HTTP w zwięzły i czytelny sposób. Rest-assured API ma na celu uproszczenie procesu interakcji z usługami internetowymi RESTful. Zapewnia zestaw metod wysyłania żądań HTTP, takich jak GET, POST, PUT, DELETE i PATCH, a także różne opcje konfigurowania tych żądań, takie jak ustawianie nagłówków, parametrów zapytań i treści żądań. Oprócz wysyłania żądań Rest-assured oferuje również różne metody sprawdzania poprawności odpowiedzi HTTP. Te metody umożliwiają programistom sprawdzenie stanu, treści odpowiedzi, nagłówków i innych atrybutów. Biblioteka zapewnia również wsparcie dla pracy z różnymi formatami danych, takimi jak JSON i XML ułatwiając weryfikację odpowiedzi korzystających z tych formatów. Rest-assured obejmuje również funkcje zarządzania uwierzytelnianiem i autoryzacją, rejestrowania żądań, odpowiedzi oraz zarządzania plikami cookie i sesjami. Te funkcje ułatwiają testowanie złożonych scenariuszy obejmujących uwierzytelnianie i zarządzanie sesją.

Rest-assured oferuje wiele zalet dla programistów i testerów, oto kilka z nich:

1. Łatwość użycia:
   - Prosty i intuicyjny interfejs do interakcji z usługami internetowymi RESTful;
   - Szybkie i efektywne pisanie testów;
2. Kompatybilność z popularnymi frameworkami testowymi:
   - Zgodność z frameworkami takimi jak JUnit i TestNG;
   - Wygodna opcja do testowania interfejsów API REST w projektach opartych na Javie;
3. Obsługa różnych metod HTTP:
   - Elastyczność testowania scenariuszy;
   - Łatwe testowanie różnych aspektów interfejsu API REST;
4. Konfigurowanie żądań:
   - Ustawianie nagłówków, treści żądań i parametrów zapytań;
   - Łatwe testowanie różnych scenariuszy;
5. Weryfikacja odpowiedzi:
   - Różnorodne metody sprawdzania poprawności odpowiedzi http;
   - Weryfikacja kodów stanu, treści odpowiedzi i nagłówków;

Dzięki tym zaletom, programiści i testerzy mogą efektywnie testować interfejsy API REST za pomocą narzędzia Rest-assured. Rest-assured posiada również swoje wady, takie jak:

1. Ograniczona obsługa innych języków programowania:

- Rest-assured jest biblioteką opartą na Javie;

2. Wymaga znajomości języka Java:
   - Mimo prostego i intuicyjnego interfejsu, Rest-assured nadal wymaga pewnej znajomości języka Java.
3. Ograniczona obsługa innych protokołów:
   - Rest-assured jest głównie skoncentrowany na obsłudze protokołu HTTP.
4. Stroma krzywa uczenia się dla zaawansowanych funkcji:
   - Rest-assured oferuje funkcje do testowania złożonych scenariuszy, takie jak uwierzytelnianie i zarządzanie sesjami.
   - Korzystanie z tych zaawansowanych funkcji może wymagać czasu i nauki dla programistów nieznających biblioteki.
5. Potencjalne spowolnienie przy dużych zestawach testów:
   - Konfigurowanie i rozłączanie połączeń HTTP dla każdego testu może prowadzić do spowolnienia w przypadku dużych zestawów testowych.
   - Testowanie równoległe może być stosowane w celu złagodzenia tego problemu.

Mimo tych wad, Rest-assured nadal jest użytecznym narzędziem do testowania interfejsów API REST, szczególnie dla programistów pracujących w Javie i mających odpowiednią znajomość biblioteki.

Test dla metody POST oraz GET dla endpointu warehouses w klasie testowej WarehouseControllerTests aplikacji E-Montażysta przedstawia listing 2 oraz 3.

Listing 2 E – montażysta – test dla metody POST. Źródło: Opracowanie własne.

```
@Test
@Order(1)
public void post() {
     Response response = given()
          .header("Content-type", "application/json")
          .header("Authorization", "Bearer "+ AbstractTest.TOKEN)
          .and()
          .body(postBody)
          .when()
          .post(AbstractTest.BASE_PATH + " /warehouses")
          .then()
          .extract().response();
     response.getBody().prettyPeek();

     Assertions.assertEquals(201, response.statusCode());
}
```

Listing 3 E – montażysta – test dla metody GET. Źródło: Opracowanie własne.

```
@Test
@Order(4)
public void getAll() {

     Response response = given()
                .header("Authorization", "Bearer "+
AbstractTest.TOKEN)
                .contentType(ContentType.JSON)
                .when()
                .get(AbstractTest.BASE_PATH + " /warehouses/all")
                .then()
                .extract().response();

     response.getBody().prettyPeek();

     Assertions.asserTrue(response.getBodY().jsonPath().getList("$")
.size()>0);
}
```

Dzięki adnotacji zaprezentowanej na listingu 4, można określić kolejność wykonania testów zdefiniowanych w klasie testowej.

Listing 4 E – montażysta – przykład adnotacji. Źródło: Opracowanie własne.

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
```

4.10. Postman

Postman według [26] jest popularnym narzędziem do testowania API, które zostało stworzone z myślą o developerach, którzy chcą przetestować swoje API. Aplikacja oferuje wiele zalet, w tym łatwość obsługi i intuicyjny interfejs użytkownika, który umożliwia programistom szybkie tworzenie zapytań, zarządzanie nagłówkami i ciałem żądania oraz przeglądanie odpowiedzi od serwera.

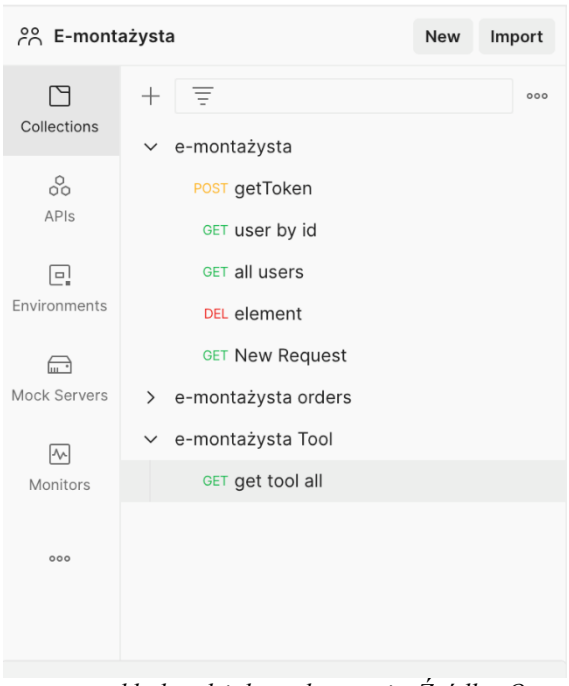
Pierwsze pomysły nad powstaniem aplikacji Postman rozpoczęły się w 2012 roku związku z problemami jakie napotkali współzałożyciele Abhinaw Astana oraz Ankit Sobti, podczas swojego stażu w Yahoo Bangalore. Zadaniem ich była praca nad różnymi API i udostępnienie ich programistom. Przy każdej aktualizacji interfejsów API musieli zaczynać od nowa i ich obecne narzędzia były trudne w obsłudze oraz nie były dostosowane do pracy w zespole. Pierwsza wersja Postmana była udostępniona jako rozszerzenie przeglądarki chrome i od samego początku wzbudziła duże zainteresowanie wśród użytkowników i nawet samej firmy Google, która zdecydowała się rozszerzenie dodać na stroną główną sklepu. I tak w 2014 roku z małego pobocznego pomysłu powstał startup założony w Indiach. Na każdym etapie rozwoju produktu ważnym elementem dla nich było słuchanie opinii i potrzeb użytkowników co przyczyniło się do dużego sukcesu aplikacji. Z strony głównej Postmana z dnia 13.02.2023 możemy wyczytać, że aktualnie jest ponad 25 milionów użytkowników tego produktu.

Jedną z największych zalet Postmana jest jego bogaty zestaw narzędzi do testowania. Oferuje on różne rodzaje testów, w tym testy jednostkowe, testy integracyjne i testy funkcjonalne, dzięki czemu można przetestować każdy aspekt swojego API. Ponadto, aplikacja umożliwia automatyzację testów za

pomocą skryptów, co pozwala na szybkie i efektywne testowanie API w sposób ciągły. Kolejną zaletą Postmana jest jego dostępność na wielu platformach, w tym na przeglądarki internetowe i różne systemy operacyjne takie jak macos, windows, linux, dzięki czemu można go używać na wielu urządzeniach. Aplikacja oferuje również wiele integracji z innymi narzędziami deweloperskimi, takimi jak GitHub, co ułatwia pracę zespołową i zarządzanie projektami.

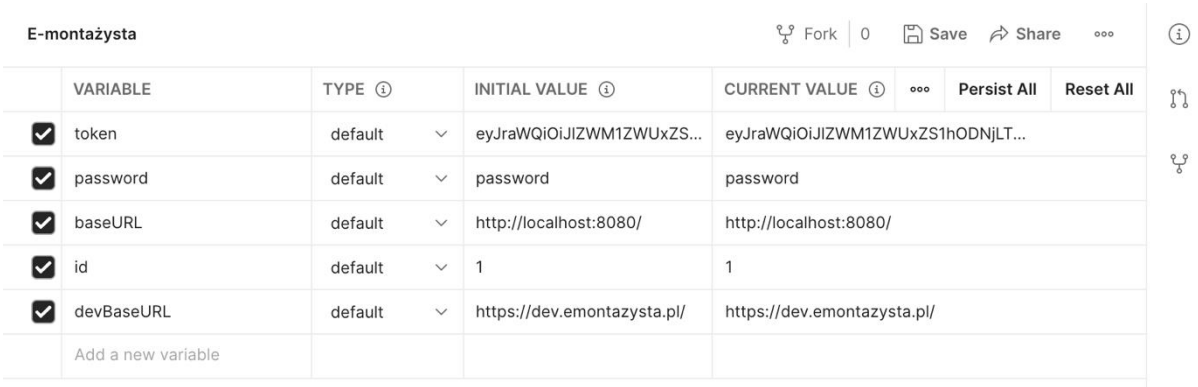
W Postmanie można tworzyć kolekcje, które pomagają w zarządzaniu grupami zapytań.

Przykładowo możemy podzielić nasze zapytania w zależności od testowanego endpointu lub kategorii, co zostało przedstawione na rysunku 36.



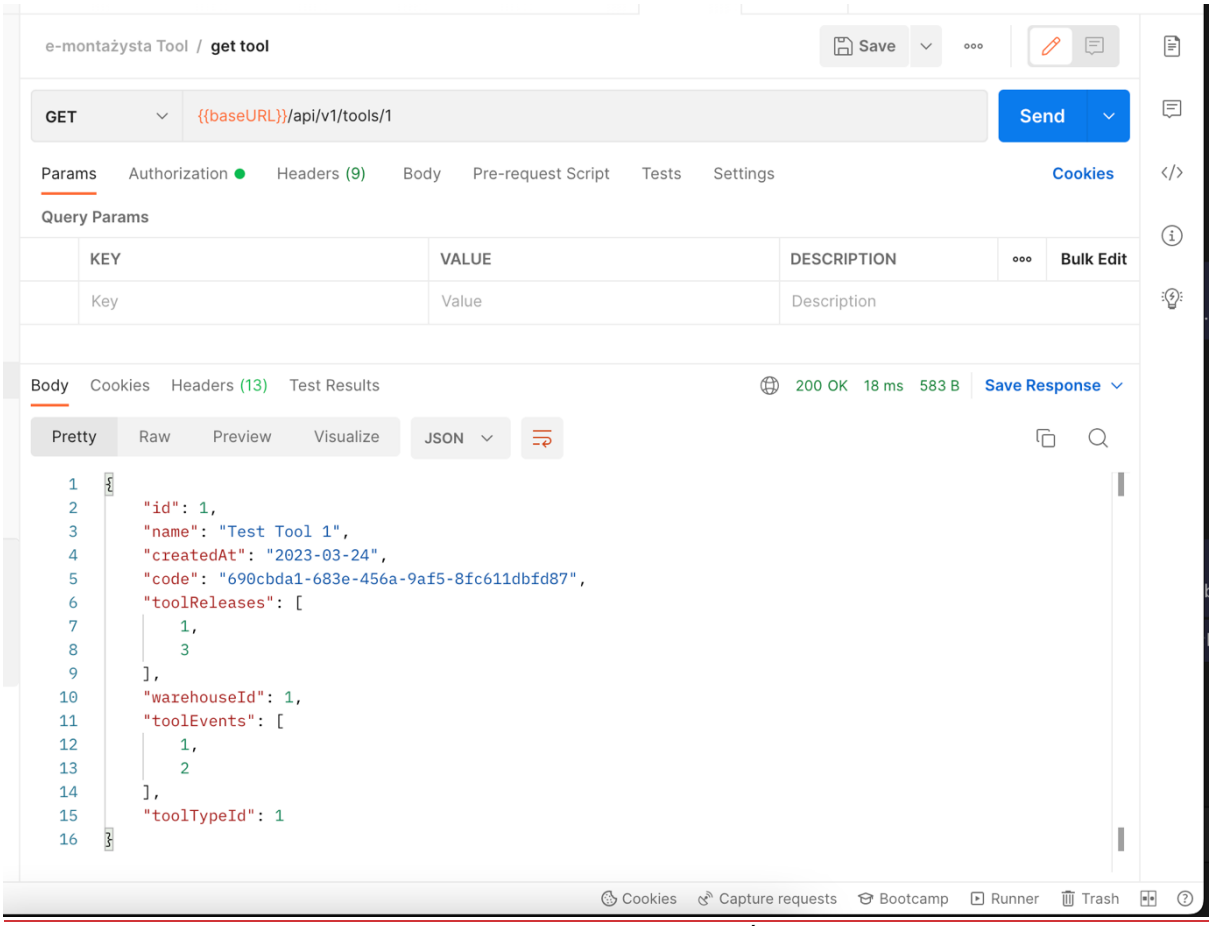
Rysunek 36 E – montażysta – przykład podziału na kategorie. Źródło: Opracowanie własne.

W Postmanie mamy również możliwość tworzenia środowisk, w których możemy ustawiać zmienne i ich wartości. W taki sposób można testować różne środowiska dev/test/prod przy tych samych kolekcjach zapytań. Rysunek 37 przedstawia użyte parametry.



Rysunek 37 E – montażysta – przykład podziału na środowiska. Źródło: Opracowanie własne

Rysunek 38 przedstawia przykład zapytania GET przy użyciu zmiennych środowiskowych oraz odpowiedź serwera.



Rysunek 38 E – montażysta – przykład zapytania GET. Źródło: Opracowanie własne.

Podsumowując, Postman to popularne narzędzie dla programistów do testowania API, które oferuje łatwy w obsłudze i intuicyjny interfejs użytkownika, pozwalający na szybkie tworzenie zapytań i zarządzanie nagłówkami oraz ciałem żądania. Historia Postmana zaczyna się w 2012 roku, kiedy to współzałożyciele zauważyli problemy z narzędziami testującymi API, co skłoniło ich do stworzenia własnego narzędzia. Pierwsza wersja Postmana była rozszerzeniem dla przeglądarki Chrome, a teraz jest dostępna na wielu platformach. Postman oferuje różne rodzaje testów, w tym testy jednostkowe, integracyjne i funkcjonalne, a także automatyzację testów za pomocą skryptów. Kolejną zaletą jest możliwość tworzenia kolekcji i środowisk, co ułatwia zarządzanie grupami zapytań oraz testowanie różnych środowisk.

4.11. Selenium

Selenium według **Błąd! Nie można odnaleźć źródła odwołania.** to szeroko stosowane narzędzie d o automatycznego testowania aplikacji internetowych, które obsługuje wiele języków programowania, takich jak Java, Python i C#. Testy Selenium API pozwalają programistom pisać skrypty symulujące działania użytkownika w aplikacji webowej i weryfikujące oczekiwane zachowanie aplikacji. Testy można uruchamiać w różnych środowiskach, w tym w trybie bezobsługowym, który umożliwia szybsze wykonywanie testów, oraz integrację z potokami ciągłej integracji/ciągłego wdrażania (CI/CD). Testy Selenium API mogą być używane w szerokim zakresie scenariuszy testowych, w tym w testach jednostkowych, testach funkcjonalnych i testach kompleksowych. Narzędzie zapewnia bogaty zestaw interfejsów API i metod, które umożliwiają programistom interakcję z elementami sieci, takimi jak przyciski, pola tekstowe i łącza, a także zarządzanie alertami, wyskakującymi okienkami i ramkami.

Selenium to platforma typu open source do automatyzacji interakcji w przeglądarce internetowej, która została po raz pierwszy opracowana przez Jasona Hugginsa w 2004 roku, kiedy pracował w ThoughtWorks. Hugginsowi powierzono zadanie przetestowania wewnętrznej aplikacji pod kątem czasu i kosztów i okazało się, że proces testowania ręcznego był czasochłonny i podatny na błędy. Aby usprawnić ten proces, stworzył program JavaScript, który mógł zautomatyzować działania użytkownika w aplikacji internetowej. Program ten ostatecznie stał się Selenium Core, który później został połączony z innym projektem o nazwie WebDriver, aby stworzyć Selenium WebDriver. Selenium WebDriver pozwala programistom zautomatyzować interakcje przeglądarki internetowej przy użyciu różnych języków programowania, w tym Java, Python i Ruby. Z biegiem lat Selenium stało się jednym z najczęściej używanych narzędzi do automatycznego testowania stron internetowych, a tysiące firm i programistów używa go do testowania swoich aplikacji internetowych. Obecnie Selenium jest nadal aktywnie rozwijane i utrzymywane przez społeczność programistów i testerów z całego świata.

Selenium WebDriver to popularne narzędzie służące do automatyzacji przeglądarek internetowych. Pozwala pisać zautomatyzowane testy, które symulują interakcje użytkownika z aplikacją internetową, takie jak klikanie przycisków, wypełnianie formularzy i nawigacja między stronami. Pisanie skryptów testowych – Po skonfigurowaniu Selenium WebDriver można pisać skrypty testowe w wybranym języku programowania. Przykładowo można na przykład symulować kliknięcie przycisku, wypełnienie formularza lub przejście do innej strony. Testy można uruchamiać przy użyciu frameworka testowego, takiego jak JUnit lub TestNG. Platforma testowa uruchomi skrypty testowe i zgłosi wszelkie błędy lub awarie, które wystąpią podczas testu.

Selenium IDE (Integrated Development Environment) to oparte na przeglądarce narzędzie do rejestrowania i odtwarzania interakcji użytkownika z aplikacją internetową. Został po raz pierwszy wprowadzony w 2006 roku jako rozszerzenie przeglądarki Firefox, a później został rozszerzony o obsługę Chrome i innych przeglądarek. IDE pozwala użytkownikom rejestrować ich interakcje z aplikacją internetową i generować skrypt, który można odtworzyć w dowolnym momencie. Może to być przydatne do szybkiego tworzenia zautomatyzowanych testów dla prostych scenariuszy, takich jak wypełnienie formularza lub kliknięcie łącza. Oprócz nagrywania i odtwarzania interakcji użytkownika, Selenium IDE zawiera inne funkcje, które można wykorzystać do tworzenia i edytowania skryptów testowych. Obejmują one:

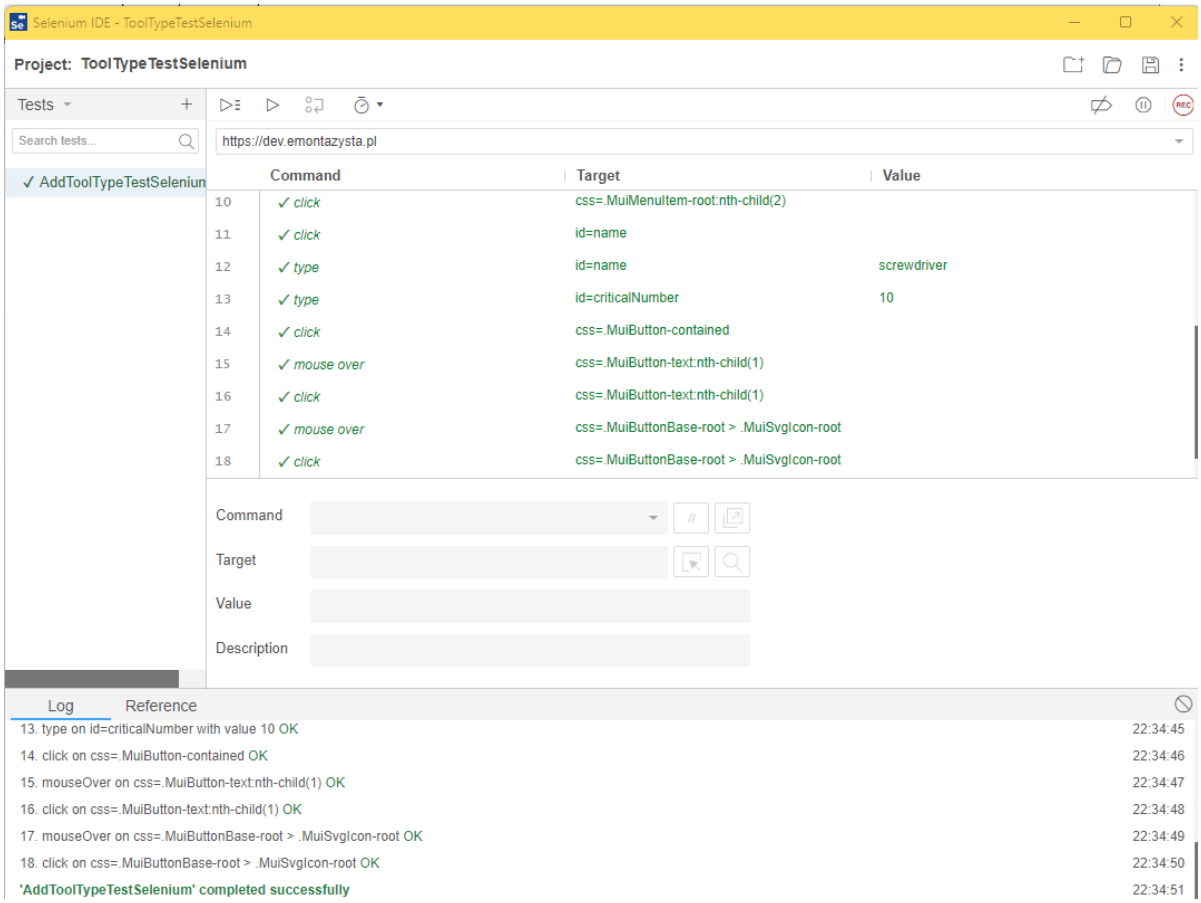
- Kontrolę elementów - przeglądanie kodu HTML i CSS strony internetowej oraz wybieranie określonych elementów do interakcji.
- Zarządzanie przypadkami testowymi –organizowanie skryptów testowych w zestawy testów i przeglądanie wyników testów.
- Parametryzację - definiowanie zmiennych i wartości wejściowych, które mogą być używane w wielu przypadkach testowych.

Selenium IDE przeszło kilka iteracji na przestrzeni lat. Obejmuje obsługę wielu przeglądarek, w tym Chrome, Firefox i Edge.

Chociaż Selenium IDE jest przydatnym narzędziem do tworzenia prostych testów automatycznych, ma ograniczoną funkcjonalność w porównaniu do Selenium WebDriver. W rezultacie wielu programistów i testerów woli używać WebDriver do bardziej złożonych scenariuszy testowych. Jednak Selenium IDE pozostaje popularnym narzędziem dla tych, którzy są nowicjuszami w automatyzacji testów lub potrzebują szybko tworzyć testy dla prostych scenariuszy. Przykład kodu w języku programowania Java dla klasy testów Selenium w aplikacji e-montażysta prezentuje listing 5. Rysunek 39 przedstawia widok testu Selenium IDE dla dodawania typu narzędzia w aplikacji e-montażysta.

Listing 5 E- montażysta - przykład kodu dla klasy testów Selenium. Źródło: Opracowanie własne.

```
public class LoginPageTest extends BasePage {
    public static WebDriver edgedriver = new EdgeDriver();
    public WebDriverWait wait = new WebDriverWait(edgedriver,
Duration.ofSeconds(5));

    @Test
    public void loginPage() throws InterruptedException {
        WebDriverManager.edgedriver().setup();

        edgedriver.get(baseUrl);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("em
ail")));
        WebElement loginElement =
edgedriver.findElement(By.name("email"));


        clickElement(loginElement);
        loginElement.sendKeys(getLogin());


wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/
/*[@id=\":r1:\"]")));
        WebElement passwordElement =
edgedriver.findElement(By.xpath("//*[@id=\":r1:\"]"));
        passwordElement.sendKeys(getPassword());

        WebElement loginButtonElement =
edgedriver.findElement(By.xpath("//*[@id=\"root\"]/div/form/div[3]/d
iv[1]/button"));
        clickElement(loginButtonElement);


        Thread.sleep(2000);

        edgedriver.close();
```



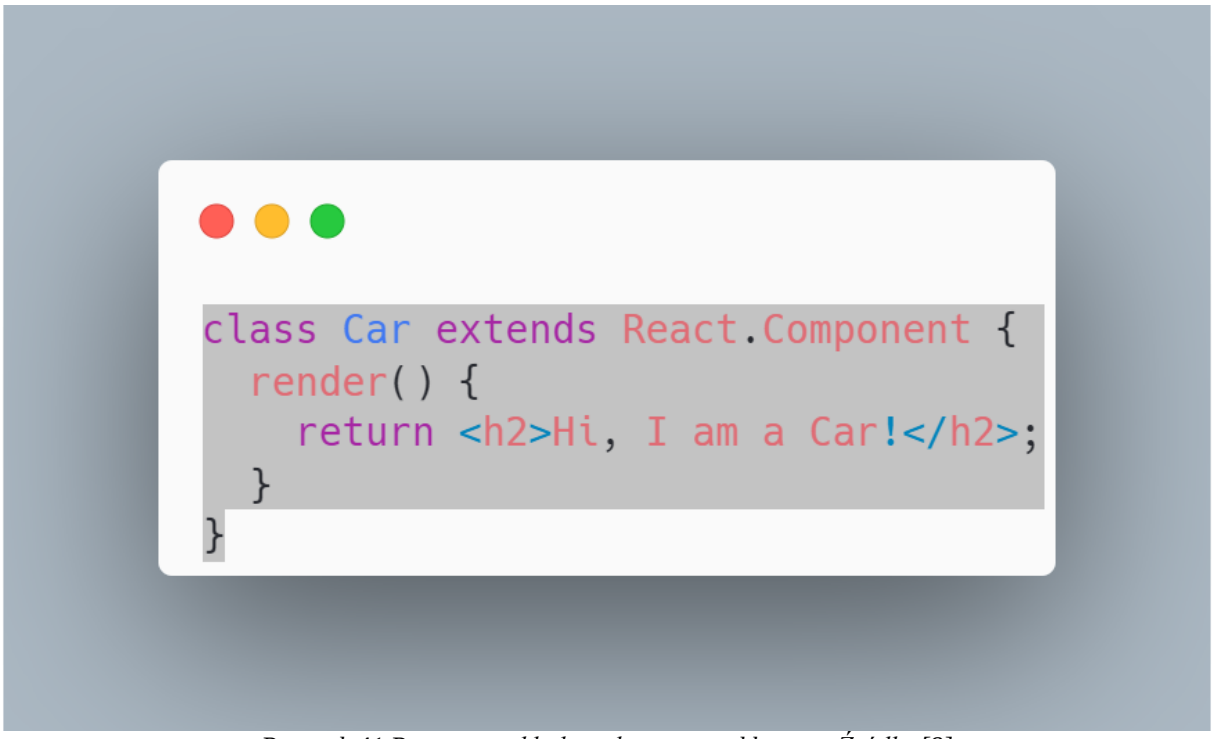
Rysunek 39 E- montażysta - widok testu Selenium IDE. Źródło: Opracowanie własne.

4.12. React

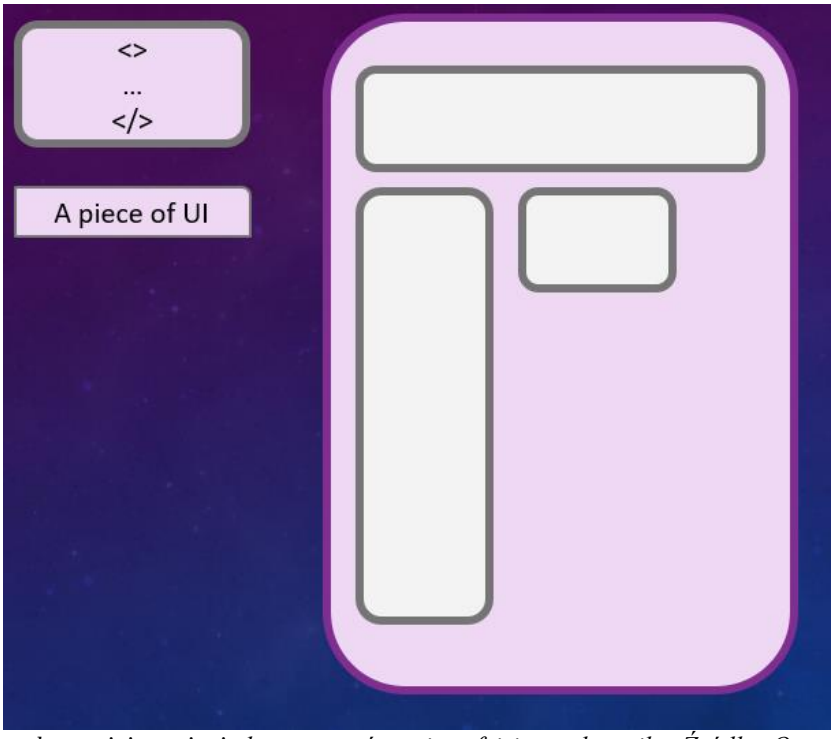
React według **Błąd! Nie można odnaleźć źródła odwołania.** to szeroko stosowana biblioteka J avaScript o otwartym kodzie źródłowym. Zapewnia potężny i wydajny sposób tworzenia interfejsów użytkownika (UI) dla aplikacji internetowych. React posiada architekturę opartą na komponentach, wirtualny DOM, składnię JSX, jednokierunkowy przepływ danych oraz tzw. hooks.

Rdzeniem architektury React jest koncepcja komponentów. Komponent to samodzielny fragment kodu wielokrotnego użytku, który reprezentuje fragment interfejsu użytkownika. React zachęca do dzielenia interfejsu użytkownika na małe, modułowe komponenty, umożliwiając programistom tworzenie złożonych interfejsów użytkownika poprzez łączenie tych komponentów. Komponenty mogą być oparte na klasach lub funkcjonalne. Rysunek 40 przedstawia przykładowy komponent funkcjonalny, natomiast na rysunku 41 znajduję się komponent klasowy. Wizualne umiejscowienie tych komponentów w interfejsie użytkownika przedstawia rysunek 42.

Rysunek 40 React - przykładowy komponent funkcjonalny. Źródło: [7]



Rysunek 41 React - przykładowy komponent klasowy. Źródło:[8]



Rysunek 42 Wizualne umiejscowienie komponentów w interfejsie użytkownika. Źródło: Opracowanie własne

Jedną z głównych korzyści płynących z użycia Reacta jest wykorzystanie wirtualnego modelu DOM. Wirtualny DOM stanowi lekką kopię rzeczywistego modelu DOM (Document Object Model), który reprezentuje strukturę strony internetowej w formie drzewa. React tworzy ten wirtualny DOM i skutecznie nim zarządza, umożliwiając efektywną aktualizację i renderowanie interfejsu użytkownika. Gdy zmienia się stan komponentu lub właściwości, React porównuje wirtualny DOM z poprzednią wersją, identyfikuje różnice i wprowadza jedynie niezbędne aktualizacje do rzeczywistego DOM. Taka strategia minimalizuje manipulacje na rzeczywistym modelu DOM, co przekłada się na lepszą wydajność.

JSX jest ważną częścią składni React. Jest to rozszerzenie składni JavaScript, które umożliwia programistom pisanie kodu podobnego do HTML w JavaScript. JSX łączy moc JavaScript z wyrazistością HTML, ułatwiając opisywanie komponentów interfejsu użytkownika. JSX nie jest rozumiany bezpośrednio przez przeglądarki, więc musi zostać przekształcony w zwykły JavaScript za pomocą narzędzia do budowania, takiego jak Babel.

React charakteryzuje się jednokierunkowym przepływem informacji. Komponenty nadrzędne przekazują dane do komponentów podrzędnych. Komponenty potomne mogą używać tych danych do renderowania interfejsu użytkownika lub wywoływania zdarzeń. Gdy komponent podrzędny musi dokonać aktualizacji, wywołuje funkcję wywołania zwrotnego przekazaną przez komponent nadrzędny. Dzięki temu zmiany danych są przewidywalne i łatwe do śledzenia, co ułatwia utrzymanie aplikacji.

React wprowadził tzw. hooki w wersji 16.8, które zrewolucjonizowały sposób, w jaki programiści zarządzają zdarzeniami stanu i cyklu życia w komponentach funkcjonalnych. Hooki to funkcje, które pozwalają programistom używać stanu i innych funkcji Reacta bez pisania komponentów opartych na klasach. Na przykład hook useState umożliwia komponentom funkcjonalnym posiadanie własnego stanu lokalnego. Hooki zapewniają czystszy, bardziej zwięzły sposób obsługi zdarzeń związanych ze stanem i cyklem życia, ułatwiając czytanie i konserwację kodu. Rysunek 30 prezentuję przykładowy hook useState.

Rysunek 43 React - Przykładowy hook useState. Źródło: [8]

4.13. TypeScript

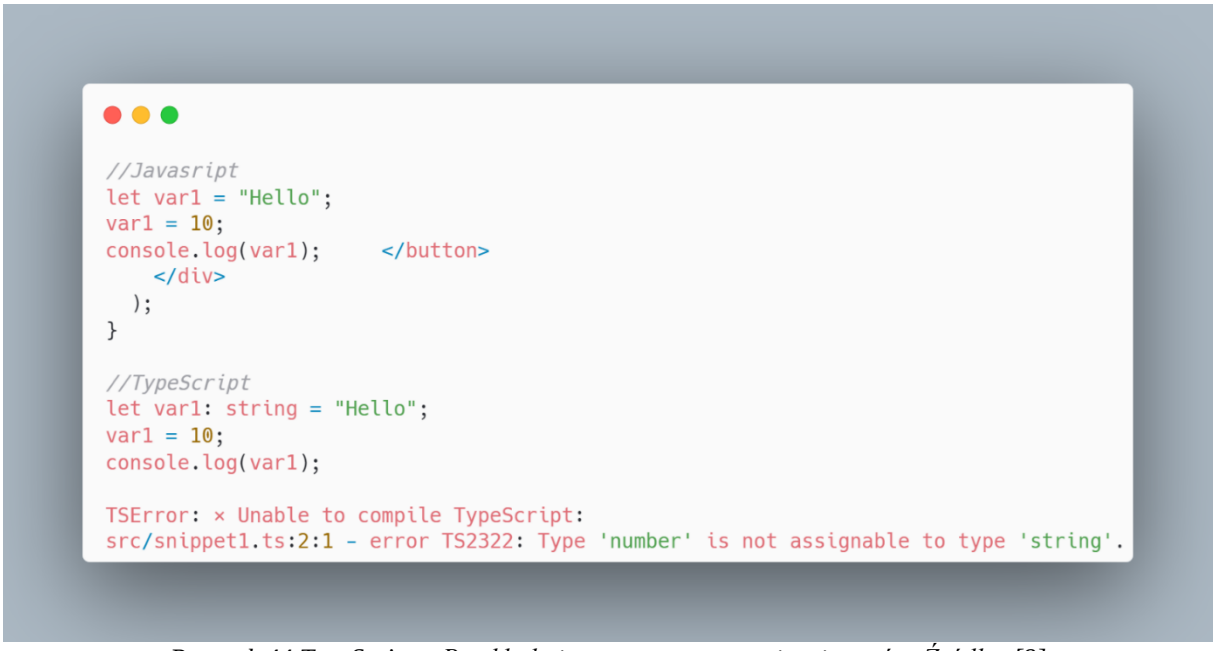
TypeScript według **Błąd! Nie można odnaleźć źródła odwołania.** to nadzbiór języka JavaScript t ypu open source opracowany i utrzymywany przez firmę Microsoft. Ma na celu ulepszenie środowiska programistycznego poprzez zapewnienie takich funkcji, jak pisanie statyczne, zaawansowane narzędzia i ulepszona skalowalność.

Jedną z głównych cech TypeScript jest pisanie statyczne (nadawanie typów zmiennym w czasie kompilacji programu). W przeciwieństwie do JavaScript, w którym zmienne mogą przechowywać wartości dowolnego typu, TypeScript umożliwia programistom określanie typów zmiennych, parametrów funkcji i zwracanych wartości. Pomaga to wychwycić błędy w czasie kompilacji, umożliwiając programistom zidentyfikowanie i naprawienie problemów przed uruchomieniem kodu. Typowanie statyczne poprawia jakość kodu, zwiększa łatwość konserwacji i zapewnia lepszą obsługę narzędzi, takich jak autouzupełnianie i nawigacja po kodzie.

TypeScript używa wnioskowania o typie do automatycznego określania typów zmiennych na podstawie ich wartości początkowych. Jednak w razie potrzeby programiści mogą również jawnie dodawać adnotacje do typów. TypeScript obsługuje szeroką gamę wbudowanych typów, w tym typy pierwotne (takie jak łańcuch, liczba i wartość logiczna), a także bardziej złożone typy, takie jak tablice, obiekty.

Inną godną uwagi cechą TypeScript jest jego kompatybilność z istniejącymi bazami kodu JavaScript. TypeScript jest blisko spokrewniony z JavaScript i jest często określany jako nadzbiór tego języka. W związku z tym każdy kod JavaScript, który jest uważany za prawidłowy, będzie również uważany za prawidłowy w TypeScript. Deweloperzy mogą stopniowo wprowadzać TypeScript do istniejącego projektu JavaScript, pozwalając im czerpać korzyści z pisania statycznego przy jednoczesnym ponownym wykorzystaniu i stopniowym ulepszaniu istniejącej bazy kodu. TypeScript umożliwia również programistom dostęp do bibliotek i frameworków JavaScript dzięki systemowi plików deklaracji, który zapewnia definicje typów dla zewnętrznego kodu JavaScript.

TypeScript zwiększa wygodę programistów dzięki obsłudze zaawansowanych narzędzi. Kompilator TypeScript, znany jako tsc, przeprowadza statyczne sprawdzanie typów i kompiluje kod do zwykłego JavaScript. Podczas procesu kompilacji kompilator dostarcza cennych informacji zwrotnych na temat możliwych błędów typu, umożliwiając programistom wykrycie ich i poprawienie na wczesnym etapie. TypeScript nie pozwoli na przypisanie typu int do typu string co przedstawia rysunek 44.



Rysunek 44 TypeScript – Przykład niepoprawnego przypisania typów. Źródło: [8]

TypeScript oferuje również ulepszoną skalowalność dla projektów na dużą skalę. Dzięki statycznemu typowaniu i adnotacjom typu, TypeScript pozwala programistom wydajniej manipulować złożonymi bazami kodu. Zapewnia lepszą organizację kodu, ułatwia nawigację po kodzie oraz ułatwia zrozumienie i utrzymanie kodu w miarę upływu czasu. System typów TypeScript pomaga uniknąć typowych błędów programistycznych i umożliwia programistom tworzenie solidniejszych i niezawodnych aplikacji.

Ponadto TypeScript dobrze integruje się z nowoczesnymi frameworkami i bibliotekami do tworzenia stron internetowych. Wiele popularnych platform, takich jak Angular, React czy Vue.js, ma oficjalne wsparcie dla TypeScript.

4.14. Figma

W projekcie zdecydowano się na użycie popularnego narzędzia Figma do projektowania interfejsów użytkownika i strony internetowej. Narzędzie to według **Błąd! Nie można odnaleźć źródła o dwołania.** działa w chmurze i umożliwia w sposób szybki i prosty projektowanie, prototypowanie oraz udostępnianie projektów. Co więcej, pozwala na równoczesną pracę wielu użytkowników w czasie rzeczywistym.

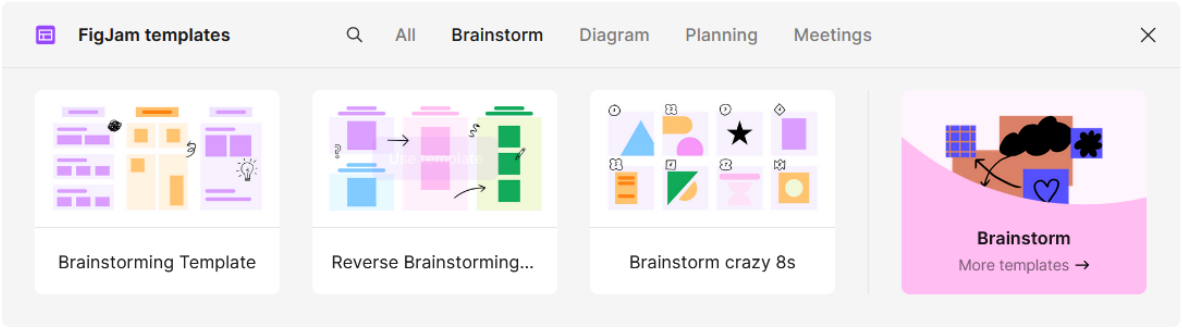
Prace nad aplikacją Figma rozpoczęły się w roku 2011 dzięki współzałożycielom Dylana Fieldowi i Evanowi Wallace'owi. Pierwsza wersja oprogramowania, która była dostępna tylko dla wybranych użytkowników, została wydana w grudniu 2015 roku. Następnie, we wrześniu 2016 roku, udostępniono pierwszą publiczną wersję dla wszystkich użytkowników. Od tamtego czasu produkt ten ciągle się rozwija i zyskuje coraz większą popularność na rynku. W roku 2022 została przedstawiona

oferta zakupu firmy przez duże amerykańskie przedsiębiorstwo informatyczne Adobe Inc. za około 20 miliardów amerykańskich dolarów (USD).

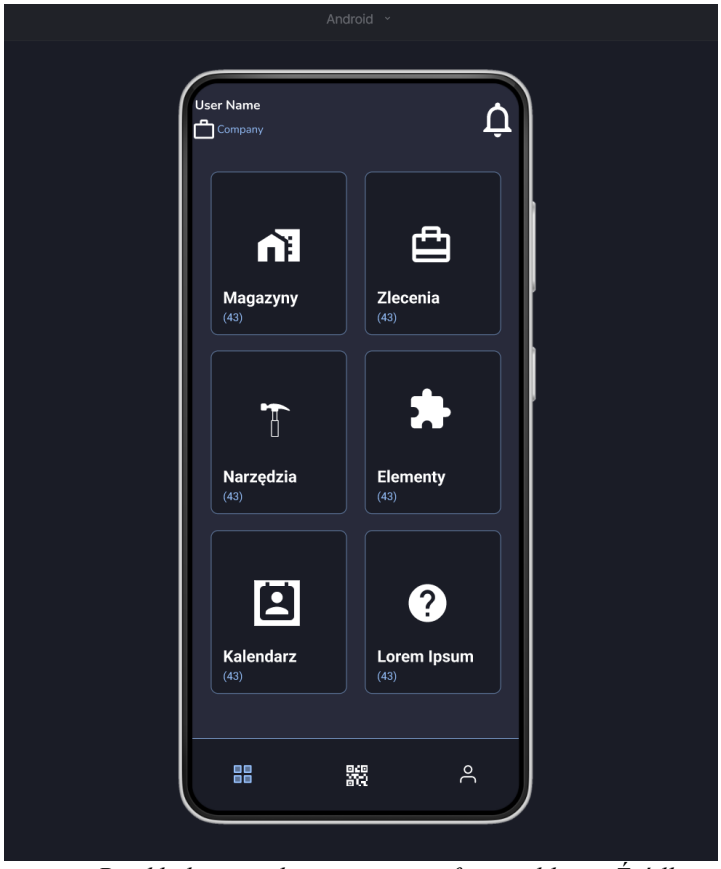
Narzędzie ma wiele zalet, które przyczyniły się do jego sukcesu wśród projektantów i nie tylko. Poniżej kilka z nich:

- Umożliwienie tworzenia interaktywnych prototypów, które można udostępniać innym osobom do przetestowania. Zaprezentowanie przejść między ekranami, animacje i inne efekty, które pomagają ze zrozumieniem projektu.
- Współpraca w czasie rzeczywistym, nad projektem może pracować jednocześnie wielu użytkowników, wszystkie zmiany są bezzwłocznie widoczne u pozostałych osób. Usprawnia i przyspiesza to współpracę zespołową.
- Działanie w chmurze, pozwala na pracę na każdym urządzeniu, które posiada przeglądarkę internetową, udostępnia to możliwość pracy nad tymi samymi projektami na kilku różnych urządzeniach bez potrzeby przenoszenia plików.
- Historyzacja zmian i udostępnianie projektów innym osobom, poprzez przesłanie linku do projektu. Użytkownicy mogą dodawać komentarze i podglądać historię zmian.
- Biblioteka zasobów, narzędzie pozwala na tworzenie bibliotek ikon, komponentów czy stylów tekstowych, ułatwia to zachowanie spójności projektów i oszczędza czas przy projektowaniu poprzez wielokrotne używanie obiektów.
- Wsparcie i aktualizacje, Figma jest stale rozwijana poprzez dodawanie nowych funkcji i ulepszanie na podstawie opinii użytkowników.
- Szablony, narzędzie posiada wiele darmowych szablonów, z różnych obszarów, diagramów, planowania, prowadzenie spotkań czy burza mózgów.

Rysunek 45 ilustruje zestaw wzorców (templates) dostępnych w aplikacji. Natomiast rysunek 46 przedstawia przykład zaprojektowanego wyglądu interfejsu aplikacji E-montażysta.



Rysunek 45 Figma- Zrzut ekranu z aplikacji. Źródło[6]



Rysunek 46 E-montażysta - Przykład zaprojektowanego interfejsu aplikacji. Źródło: opracowanie własne

Figma jest rozpowszechnionym narzędziem projektowym, które umożliwia tworzenie interfejsów, diagramów, wspiera proces planowania, a nawet służy do przeprowadzania sesji burzy mózgów. Aplikacja działa w chmurze i pozwala na bardzo prostą pracę zespołową nad projektami poprzez każde urządzenie z dostępem do Internetu i przeglądarki internetowej. Narzędzie zyskało popularność wśród projektantów dzięki funkcjom takim jak tworzenie interaktywnych prototypów, współpraca w czasie rzeczywistym, praca w chmurze, historyzacja zmian, biblioteki zasobów i regularne aktualizacje oraz skupienie na opiniach użytkowników.

4.15. Draw.io

Diagrams.net lub inaczej draw.io według **Błąd! Nie można odnaleźć źródła odwołania.** to d armowe narzędzie do tworzenia, projektowania diagramów i różnych schematów. Narzędzie jest łatwo konfigurowalne, pozwala na używanie własnych kształtów i jednocześnie posiada bardzo dużą bazę dostępnych kształtów takiach jak UML, Bootstrap, Android, iOS, Sitemap BPMN 2.0 i wiele innych co pozwala na tworzenie diagramów takich jak diagramy przepływu, diagramy sieci, schematy blokowe, diagramy UML, diagramy Venna i wiele innych. Jest narzędziem wieloplatformowym dostępnym przez przeglądarkę internetową, ale jest również dostępna wersja aplikacji do instalacji na komputerze i nie wymaga wtedy połączenia z Internetem.

DRAW.IO pozwala w czasie rzeczywistym na pracę nad diagramem wielu użytkownikom naraz, a wszystkie zmiany są zapisywane na serwerze. Poza wspólną pracą diagram można również udostępnić innym osobom tylko do podglądu, co przykładowo pozwala na śledzenie postępów innej osobie bez ryzyka, że zostaną dokonane niechciane zmiany.

Narzędzie umożliwia również importowanie i eksportowanie plików w różnych formatach, takich jak PDF, PNG, SVG i inne.

Oferuje wiele integracji z innymi narzędziami jak:

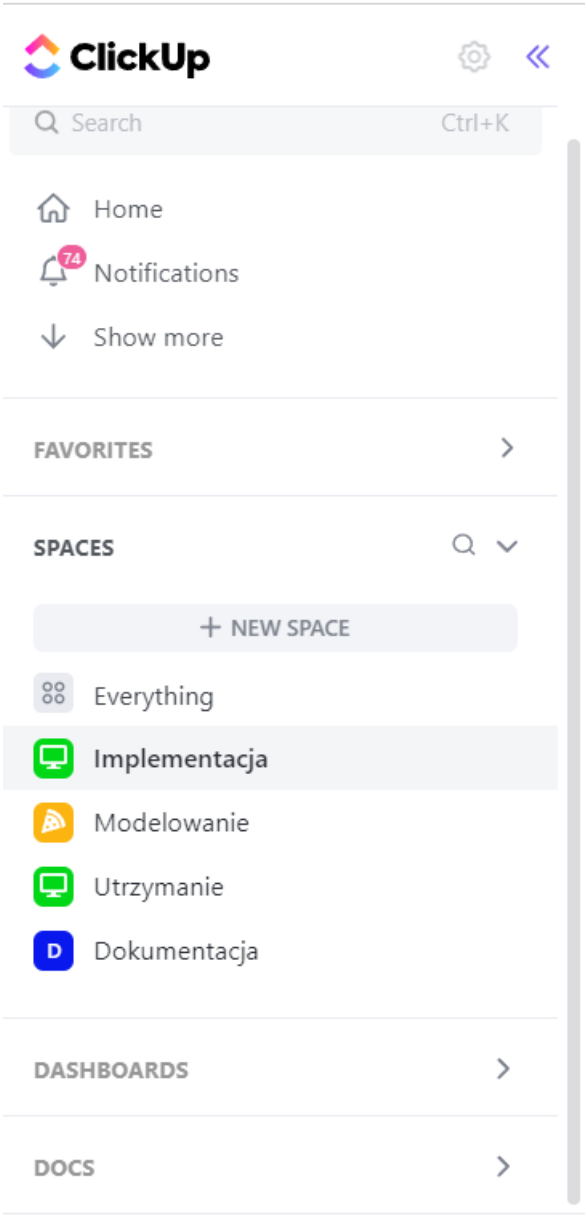
- Google Drive - Draw.io można łatwo zintegrować z Google Drive, co umożliwia przechowywanie i udostępnianie diagramów w chmurze.
- Atlassian Jira i Confluence - Draw.io jest również zintegrowany z Atlassian Jira i Confluence, co umożliwia tworzenie i udostępnianie diagramów bezpośrednio w tych narzędziach.
- GitHub - Draw.io może być również zintegrowany z GitHub, co umożliwia łatwe przechowywanie i udostępnianie diagramów w repozytoriach projektów.
- Microsoft Office - Draw.io jest zintegrowany z pakietem Microsoft Office, co umożliwia tworzenie i edycję diagramów bezpośrednio w programach Word, Excel i PowerPoint.
- WordPress - Draw.io może być zintegrowany z WordPress, co umożliwia łatwe dodawanie diagramów do stron internetowych i blogów.

I wiele innych.

Podsumowując, DRAW.IO to zaawansowane i wszechstronne narzędzie do tworzenia różnego rodzaju diagramów i schematów, oferujące wiele możliwości i opcji konfiguracji. Dzięki dużemu wyborowi dostępnych kształtów, narzędzie pozwala na tworzenie diagramów o różnej skali i złożoności, takich jak diagramy przepływu, schematy blokowe, diagramy UML i wiele innych. DRAW.IO jest również bardzo elastyczne, umożliwiając użytkownikom łatwe dostosowanie narzędzia do swoich potrzeb oraz integrację z innymi popularnymi narzędziami, co ułatwia przenoszenie danych i diagramów między różnymi aplikacjami. DRAW.IO to narzędzie idealne dla projektantów, programistów i innych osób, które potrzebują szybkiego i łatwego sposobu na tworzenie skomplikowanych diagramów i schematów.

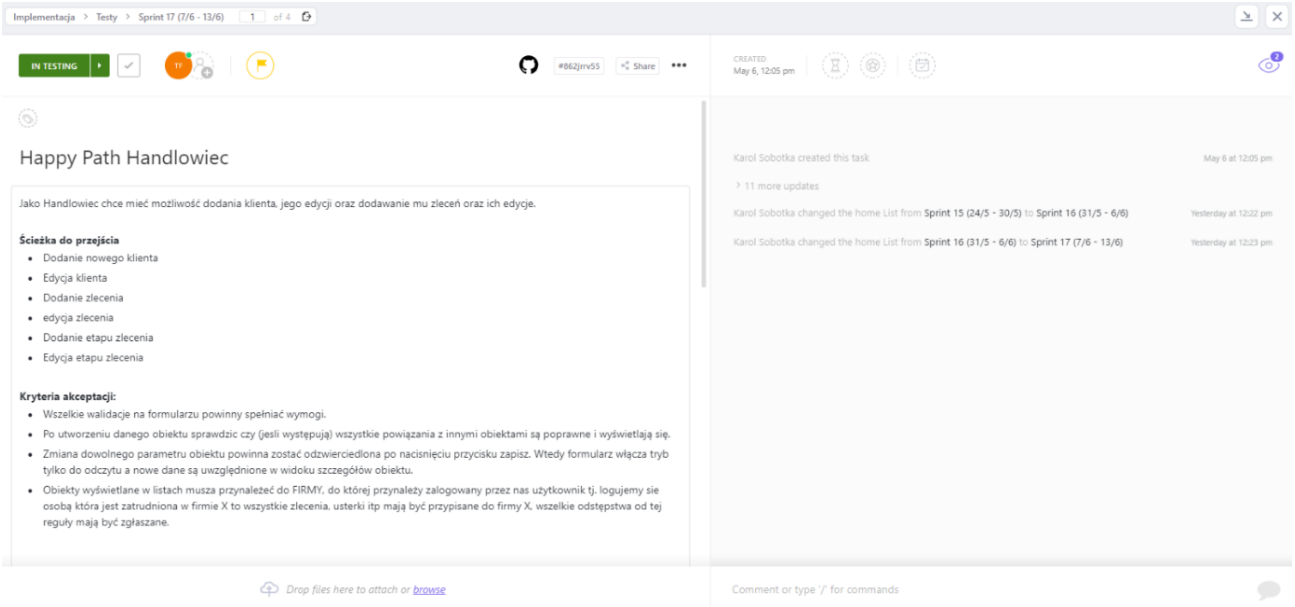
4.16. ClickUp

ClickUp to kompleksowa platforma do zarządzania projektami i współpracy zaprojektowana w celu optymalizacji wydajności zespołu i poprawy wydajności pracy. Posiada szeroką gamę funkcji i narzędzi, które pomagają zespołom organizować zadania i zarządzać nimi, skutecznie komunikować się i śledzić postępy w wielu projektach. Użytkownicy mogą tworzyć obszary robocze w celu organizowania swoich projektów i zadań. W każdym obszarze roboczym mogą tworzyć konfigurowalne listy i tablice reprezentujące różne przepływy pracy lub etapy projektu. Te listy i tablice można dostosować do konkretnych potrzeb. Rysunek 47 przedstawia przestrzeń pracy dla projektu E-montażysta.



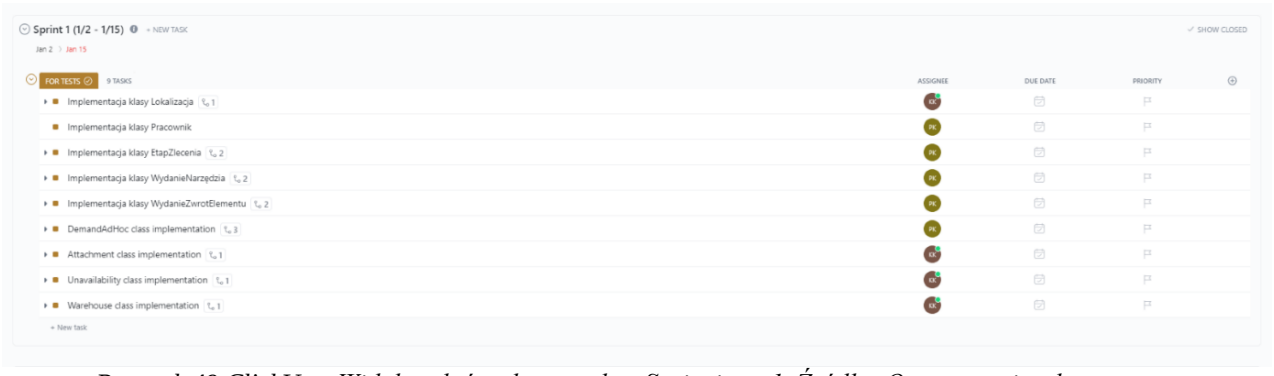
Rysunek 47 ClickUp - Przestrzeń pracy dla projektu E-montażysta. Źródło: Opracowanie własne.

Jedną z głównych cech ClickUp są możliwości zarządzania zadaniami. Użytkownicy mogą tworzyć zadania, przydzielać je członkom zespołu, ustalać terminy, dodawać opisy, dołączać pliki i ustalać priorytety na podstawie pilności lub ważności. Zadania można również organizować w hierarchie za pomocą podzadań i list kontrolnych, co pozwala użytkownikom dzielić złożone projekty na mniejsze, łatwe w zarządzaniu komponenty. Rysunek 48 przedstawia przykładowe zadanie.



Rysunek 48 ClickUp - Przykładowe zadanie. Źródło: Opracowanie własne.

ClickUp oferuje kilka widoków do przeglądania zadań i projektów. Użytkownicy mogą przełączać się między widokiem listy, widokiem tablicy, widokiem kalendarza, a nawet widokiem wykresu Gantta, który zapewnia wizualną reprezentację osi czasu projektu i zależności. Na rysunku 49 przedstawiono widok wykonanych zadań z pierwszego Sprintu.



Rysunek 49 ClickUp - Widok zadań wykonanych w Sprincie nr 1. Źródło: Opracowanie własne.

Aplikacja zapewnia funkcje śledzenia czasu, zarządzania dokumentami, wyznaczania celów i raportowania. Możliwe jest śledzenie czasu spędzonego na zadaniach, generowanie raportów do analizy wydajności zespołu, dołączanie dokumenty, zarządzanie nimi.

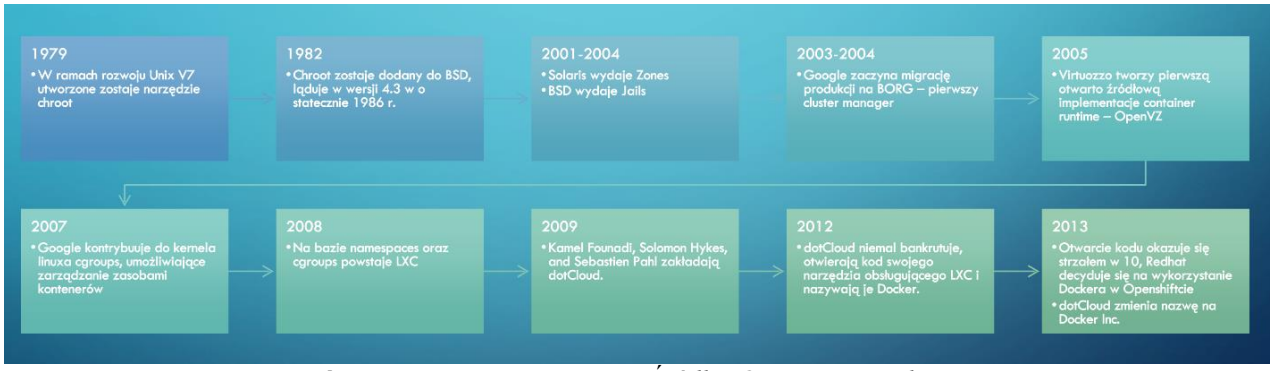
ClickUp oferuje również solidny interfejs API (Application Programming Interface), który umożliwia programistom tworzenie niestandardowych integracji, automatyzacji i rozszerzeń za pomocą ClickUp. Interfejs API zapewnia dostęp do wielu punktów końcowych, umożliwiając programistom interakcję z danymi i funkcjami ClickUp programowo.

4.17. Docker

Docker według **Błąd! Nie można odnaleźć źródła odwołania.** to narzędzie, które umożliwia d ostarczenie aplikacji wraz z jej zależnościami w lekkim, przenośnym i izolowanym kontenerze wirtualnym. Dzięki temu, oprogramowanie może być uruchamiane na większości serwerów. Wszystkie te cechy gwarantują, że oprogramowanie będzie działać zawsze w sposób niezmienny, niezależnie od

architektury, systemu operacyjnego i środowiska, w którym go uruchomi. Docker zapewnia spójność działania aplikacji, co przekłada się na większą niezawodność i możliwość łatwej migracji między różnymi środowiskami.

Rysunek 50 przedstawia historie konteneryzacji, analizując kluczowe etapy i przełomowe technologie, które przyczyniły się do narodzin i rozwoju tego nowatorskiego podejścia do wdrażania aplikacji.



Rysunek 50 Historia Konteneryzacji. Źródło: Opracowanie własne.

Jego architektura klient-serwer opiera się na kilku współpracujących ze sobą komponentach, co przedstawiono na rysunku 51. Oto opis działania Dockera:

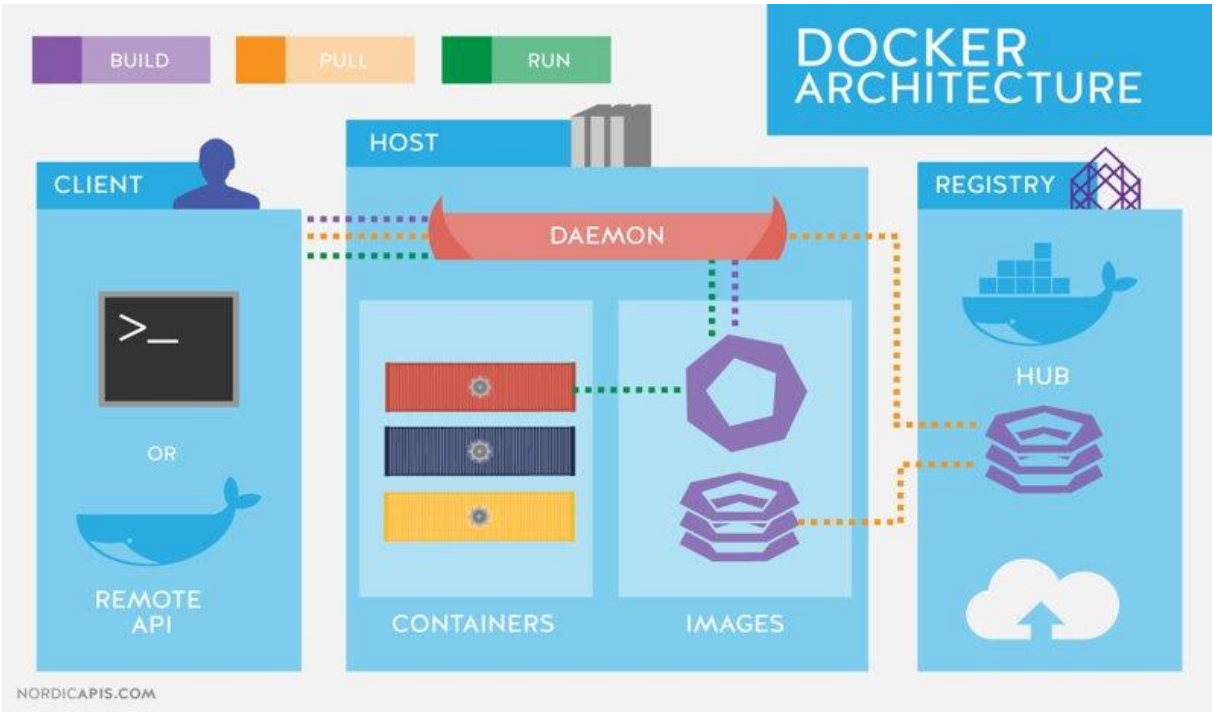
1. Docker Daemon: Główna usługa uruchamiana w systemie operacyjnym hosta. Odpowiada za budowanie, uruchamianie oraz zarządzanie kontenerami czy obrazami. Nasłuchuje i obsługuje wszystkie żądania Docker API dotyczące operacji klienta. Może komunikować się również z innymi Docker Daemon.

2. Docker Client: Narzędzie umożliwiające interakcję z Docker Daemon. Dostarcza interfejs wiersza polecen CLI, który umożliwia wydawanie poleceń związanych z tworzeniem, uruchamianiem, zarządzaniem kontenerami i obrazami Docker.

3. Obrazy Docker: Szablony tylko do odczytu, na podstawie których tworzone są kontenery z instrukcji zapisanych w pliku Dockerfile. Zawierają wszystkie zależności niezbędne do uruchomienia aplikacji.

4. Kontenery Docker: Uruchomione instancje obrazów Docker. Każdy kontener zawiera własny system operacyjny, kod aplikacji, narzędzia i biblioteki systemowe, działając w izolowanym środowisku. Pomimo częstego porównywania kontenerów Docker do maszyn wirtualnych, istnieje wiele różnic między nimi, w tym sposób działania na systemie operacyjnym hosta.

5. Rejestry Docker: Centralne repozytoria, w których przechowywane są obrazy Docker. Docker Hub jest największym dostępnym publicznie rejestrem, natomiast można również utworzyć prywatne repozytoria, które umożliwiają przechowywanie i udostępnianie obrazów wewnętrznie w organizacji.



Rysunek 51 Docker – Architektura. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***

4.18. Kubernetes

Kubernetes według **Błąd! Nie można odnaleźć źródła odwołania.** to otwarte oprogramowanie, k tórego celem jest zautomatyzowanie procesu uruchamiania, skalowania oraz zarządzania klastrami skonteneryzowanych aplikacji i usług. Początkowo rozwijany przez Google, który odegrał kluczową rolę w tworzeniu i rozwijaniu tego innowacyjnego pomysłu poprzez doświadczenie zdobyte przy budowie systemu zarządzania kontenerami o nazwie Borg. Jednakże, wraz z rozwojem Kubernetesa, pojawiło się również wiele innych dystrybucji tego rozwiązania w tym K3s, który został użyty w tym projekcie.

Poniżej przedstawiono kilka kluczowych koncepcji związanych z Kubernetsem:

1. Klastry: Zbiór maszyn roboczych umożliwiający pakowanie aplikacji i wszystkich ich zależności w izolowane jednostki. Składają się z jednego lub więcej węzłów, które połączone sieciowo, działają razem w celu uruchamiania oraz zarządzania aplikacjami kontenerowymi.

2. Serwisy: Obiekt, który definiuje stabilny punkt dostępu do jednego lub więcej podów (instancji aplikacji) w klastrze. Umożliwia użytkownikom komunikację z aplikacją bez konieczności śledzenia jej replik czy adresów IP. Serwis jest zdefiniowany w pliku yaml.

3. Wdrożenie: Proces zarządzania wydajnością, który określa pożądane zachowanie i cechy poda. Stanowi kluczowy sposób uruchamiania i utrzymywania aplikacji kontenerowych w środowisku Kubernetes. Wdrożenie umożliwia elastyczne skalowanie aplikacji w zależności od obciążenia, aktualizację na żywo, obsługę odwzorowywania dla zapewnienia wysokiej dostępności oraz deklaratywne zarządzanie, w którym można określić pożądany stan aplikacji, a Kubernetes będzie dążył do utrzymania tego stanu.

4. Skalowanie: Proces dynamicznego konfigurowania rozmiaru klastra w celu dostosowania infrastruktury do zmieniającego się obciążenia aplikacji. Pozwala na zwiększenie lub zmniejszenie

liczby podów, w zależności od potrzeb. Kubernetes oferuje dwa główne rodzaje skalowania: skalowanie pionowe (vertical scaling) i skalowanie poziome (horizontal scaling).

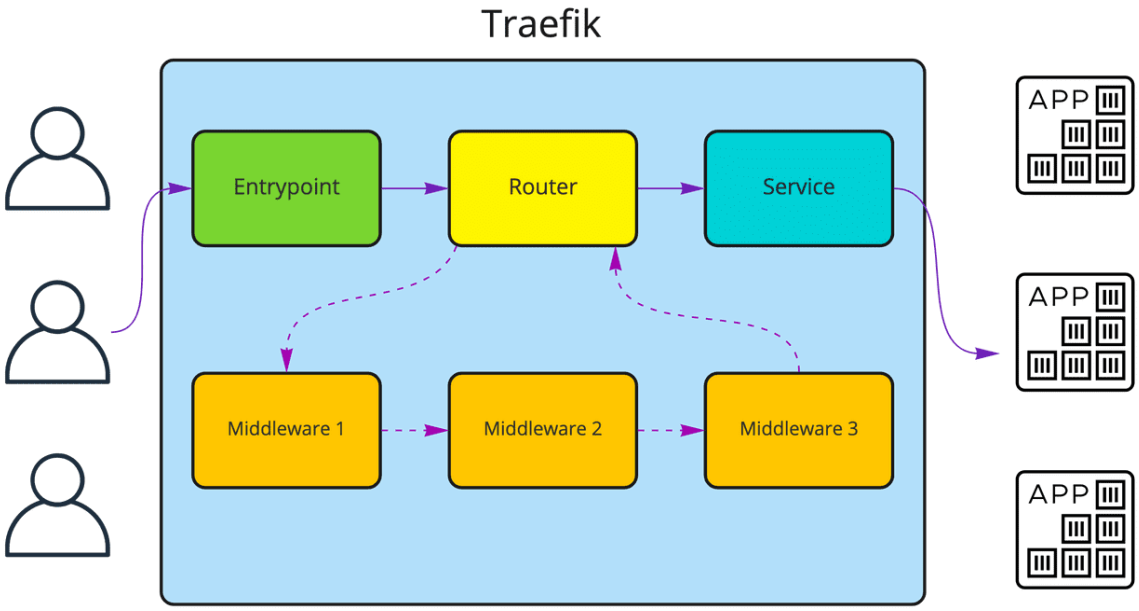
K3s to lekka wersja Kubernetes, charakteryzująca się zredukowaną złożonością, mniejszymi wymaganiami zasobowymi i prostszym procesem instalacji w porównaniu do pełnoprawnego rozwiązania. Jego architektura została starannie opracowana, aby zapewnić efektywne działanie i wydajność, ale zachowując kluczowe funkcje orkiestracji kontenerów.

4.19.Traefik

Traefik według **Błąd! Nie można odnaleźć źródła odwołania.** jest nowoczesnym narzędziem t ypu ‘edge router/reverse proxy’, którego głównym celem jest zarządzanie ruchem sieciowym wewnątrz klastra Kubernetes. Zapewnia efektywne przekierowywania żądań do odpowiednich usług. Jego architektura i funkcje są skonstruowane w taki sposób, aby automatycznie wykrywać odpowiednią konfigurację dla projektu. Traefik osiąga swoje ceele poprzez monitorowanie interfejsu API klastra Kubernetes. Narzędzie to oferuje zaawansowane funkcje routingu, algorytmów równoważenia obciążenia i obsługi protokołów szyfrowania, takich jak SSL/TLS.

Traefik rozpoczął swoją działalność jako natywny serwer proxy aplikacji w chmurze w 2016 roku. Zdobył popularność jako elastyczne narzędzie do zarządzania ruchem sieciowym i dostarczania aplikacji w kontenerach.

Traefik opiera się na czterech głównych koncepcjach: punkcie wejścia (entry point), routerze (router), oprogramowaniu pośredniczącym (middleware) oraz usłudze (service). Konfigurując Traefik, można ustawić go tak, aby nasłuchiwał na określonych punktach wejścia, takich jak porty TCP czy HTTP. Następnie, przychodzące żądania są dopasowywane do odpowiednich tras. Mogą przechodzić przez oprogramowanie pośredniczące, które może zawierać wiele funkcji, takich jak przepisywanie ścieżek czy kompresja. W efekcie żądanie trafia do skonfigurowanej usługi. W przypadku aplikacji sieciowych, serwisy są zazwyczaj mapowane na serwery sieciowe aplikacji, które obsługują żądania klientów. Rysunek 52 ilustruje sposób działania usługi Traefik.



Rysunek 52 Sposób działania usługi Traefik. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***

4.20. Github Actions

Github Actions według **Błąd! Nie można odnaleźć źródła odwołania.** to platforma u możliwiająca automatyzacje przepływów pracy w ramach serwisu GitHub. Darmowe i domyślnie dostępne dla każdego repozytorium, bez konieczności instalacji czy dodatkowej konfiguracji. Rozwiązuje wiele problemów związanych z procesem ciągłej integracji i dostarczania oprogramowania, oferując:

- Automatyzacje procesu budowania, testowania, wdrażania czy dostarczania oprogramowania;
- Elastyczność w dostosowywaniu przepływów pracy do indywidualnych potrzeb projektu;
- Integracje z wieloma popularnymi narzędziami i platformami jak Docker czy Kubernetes;
- Wirtualne środowiska testowe z różnymi wersjami oprogramowania;
- Interfejs użytkownika, w którym można śledzić postęp i wyniki przepływów pracy.

GitHub Actions opiera się na architekturze zdarzeń. Składa się z trzech podstawowych komponentów: przepływ zdarzeń, zadanie oraz krok zwanych akcją. Przepływ pracy jest podstawową jednostką organizacyjną. Definiuje sekwencję kroków do wykonania w odpowiedzi na zdarzenia w repozytorium, takie jak zmiana kodu źródłowego, utworzenie nowej gałęzi czy zgłoszenie problemu. Zadanie to niezależna jednostka pracy, która może być wykonywana równolegle lub sekwencyjnie w ramach przepływu pracy, natomiast krok to podstawowa jednostka działania w ramach zadania.

W trakcie działania przepływu zdarzeń, GitHub Actions wywołuje określone zadania, które są zdefiniowane w pliku yaml. Każde zadanie musi składać się z przynajmniej jednego kroku. Mogą to być komendy, skrypty, wywołania zewnętrznych narzędzi lub akcje. Każdy krok wykonuje określone zadanie. GitHub Actions oferuje szeroki zakres opcji konfiguracji dla poszczególnych kroków.

Podsumowując, GitHub Actions oferuje wiele korzyści dla programistów i zespołów projektowych. Dzięki integracji z platformą GitHub, możliwe jest łatwe wdrożenie i zarządzanie przepływami pracy wewnątrz repozytoriów. Elastyczność narzędzia pozwala na dostosowanie go do indywidualnych potrzeb projektu.

5. Projekt i implementacja

Zespół implementacji rozpoczął prace po zakończeniu fazy projektowania. Część członków zespołu była równocześnie zaangażowana w prace projektowe. Programiści pozostawali również w ścisłym kontakcie z testerami w celu zapewnienia płynności udoskonalania poszczególnych funkcjonalności. Wewnętrznie zespół implementacji został podzielony na mniejsze podzespoły:

1. Backend – implementacja funkcjonalności;
2. Frontend – przygotowanie interfejsu użytkownika - część webowa;
3. Android – przygotowanie aplikacji mobilnej;
4. DevOps – przygotowanie i konfiguracja środowisk deweloperskich;

W fazie początkowej implementacji, zespół zajmujący się backendem skupił się na szybkim opracowaniu niezbędnych endpointów, co przyczyniło się do ułatwienia pracy zespołowi odpowiedzialnemu za projektowanie interfejsu użytkownika. Dzięki organizacji pracy w systemie ClickUp oraz Github członkowie wszystkich podzespołów na bieżąco uzyskiwali informacje o zmianach w aplikacji.

Należy podkreślić, iż dzięki doskonałej komunikacji liderów poszczególnych podzespołów uniknięto opóźnień we wdrażaniu niektórych funkcjonalności, takich jak filtrowanie, które w większości przypadków jest realizowane po stronie serwera. Niewątpliwym atutem podziału na podzespoły był fakt, iż większość uwag dostarczanych przez zespół testerów trafiało od razu we właściwe miejsce. Zgłoszone niedociągnięcia były w krótkim odstępie czasu wnikliwie analizowane a następnie programiści przystępowali do wdrażania poprawek.

5.1. Wzorzec projektowy MVC (Model-View-Controller)

Model-View-Controller (MVC) według **Błąd! Nie można odnaleźć źródła odwołania.** to wzorzec architektury oprogramowania szeroko stosowany w tworzeniu interfejsów użytkownika i aplikacji internetowych. Zapewnia ustrukturyzowane podejście do oddzielania problemów i organizowania kodu, ułatwiając konserwację i modyfikację aplikacji w miarę upływu czasu. MVC promuje separację danych, prezentacji i logiki biznesowej, ponowne użycie i skalowalność. We wzorcu MVC możemy wyróżnić trzy elementy:

1. Model - klasy modelowe reprezentują logikę biznesową. Struktura danych jest zhermetyzowana, tak jak metody i operacje, które można wykonać na tych danych. Warstwa modelu odpowiada za pobieranie i wykorzystywanie danych pochodzących z różnych źródeł, takich jak bazy danych, interfejsy API czy pliki. W modelu zdefiniowane są również reguły biznesowe i procesy rządzące zachowaniem aplikacji. Warstwa modelu działa jako interfejs między danymi a resztą aplikacji. Na listingu 6 przedstawiono implementację przykładowej klasy Client.

Listing 6 E-montażysta - Modelowa klasa Client napisana w języku Java. Źródło: Opracowanie własne

```
@Entity
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@SQLDelete(sql = "UPDATE client SET deleted = true WHERE id=?")
```

```
@JsonIdentityInfo(generator =
ObjectIdGenerators.PropertyGenerator.class, property = "id")
public class Client {

    public Client(Long id, String name, String contactDetails,
Company company, List<Orders> orders) {
        this.id = id;
        this.name = name;
        this.contactDetails = contactDetails;
        this.company = company;
        this.orders = orders;
    }
```

2. Widok - odpowiada za warstwę prezentacyjną aplikacji. Definiuje sposób wyświetlania danych użytkownikowi i obsługuje interakcje użytkownika, takie jak kliknięcia myszą lub wprowadzanie danych z klawiatury. Warstwa wyświetlania składa się zwykle z elementów interfejsu użytkownika, takich jak przyciski, formularze i menu. Może również zawierać szablony lub języki znaczników, które definiują strukturę i układ interfejsu użytkownika. Widok pobiera dane z modelu i prezentuje je w sposób atrakcyjny wizualnie i zrozumiały dla użytkownika. Na listingu 7 przedstawiono przykładową klasę ClientDetail.

Listing 7 E-montażysta - Klasa ClientDetails napisana w języku JavaScript (React). Źródło: Opracowanie własne

```
const ClientDetails = () => {
    const params = useParams()
    const [pageMode, setPageMode] = useState<PageMode>('read')
    const { showDialog } = useContext(DialogGlobalContext)
    const formStructure = useFormStructure()
    const [initData, setInitData] =
useState(getInitValues(formStructure))

    //mutations and queries
    const addClientMutation = useAddClient()
    const editClientMutation = useEditClient((data) =>
handleOnEditSuccess(data))
    const deleteClientMutation = useDeleteClient(() =>
clientData.remove())
    const clientData = useClientData(params.id)
    //status for all mutations and queries
    const queriesStatus = useQueriesStatus([clientData],
[addClientMutation, editClientMutation, deleteClientMutation])

    const handleSubmit = (values: any) => {
        if (pageMode == 'new') addClientMutation.mutate(values)
        else if (pageMode == 'edit')
editClientMutation.mutate(values)
        else console.warn('Try to submit while read mode')
    }
```

3. Kontroler - działa jako pośrednik między modelem a widokiem. Wymaga danych wprowadzonych przez użytkownika i wyzwala odpowiednie działania w warstwie modelu lub widoku.

Obsługuje interakcje użytkownika, takie jak kliknięcia przycisków lub przesyłanie formularzy, i odpowiednio aktualizuje model. Kontroler jest odpowiedzialny za interpretację działań użytkownika i podjęcie decyzji, w jaki sposób aplikacja powinna zareagować. Koordynuje przepływ danych i aktualizacje między modelem a widokiem, zapewniając ich synchronizację. Listing 8 przedstawia jak wygląda przykładowy kod klasy kontrolera ClientControler.

Listing 8 E-montażysta - Klasa kontrolera ClientController napisana w języku Java. Źródło: Opracowanie własne

```
@RestController
@RequiredArgsConstructor
@RequestMapping(value = API_BASE_CONSTANT + "/clients", produces =
MediaType.APPLICATION_JSON_VALUE)
public class ClientController {

    private final ClientService clientService;

    @GetMapping("/all")
    @Operation(description = "Allows to get all Clients.", security
= @SecurityRequirement(name = "bearer-key"))
    public ResponseEntity <List<ClientDto>> getAll() {
        return ResponseEntity.ok().body(clientService.getAll());
    }
```

5.2. Metodologia SOLID

Zasady SOLID według **Błąd! Nie można odnaleźć źródła odwołania.** to zestaw reguł p rojektowania oprogramowania, które pomagają programistom tworzyć łatwiejszy w utrzymaniu, elastyczny i solidny kod. Zasady te, wprowadzone przez Roberta C. Martina zawierają wytyczne co do pisania czystego, skalowalnego kodu. Każda litera akronimu SOLID reprezentuje inną zasadę, a razem tworzą spójny zestaw najlepszych praktyk projektowania obiektowego.

1. Zasada pojedynczej odpowiedzialności (Single-responsibility principle - SRP)

SRP stwierdza, że klasa powinna mieć tylko jeden powód do zmiany. Każda klasa lub moduł powinien mieć jedną odpowiedzialność lub cel. Należy skoncentrować się na konkretnym zadaniu. Zmiany jednego aspektu systemu z mniejszym prawdopodobieństwem wpłyną na inne części. Ta zasada promuje kod, który jest łatwiejszy do zrozumienia, przetestowania i utrzymania. W przedstawionym listingu 9 zaprezentowany jest przykład klasy, która doskonale spełnia zasadę SRP. Klasa ta jest odpowiedzialna wyłącznie za mapowanie obiektów typu Client.

Listing 9 E-montażysta - Klasa odpowiedzialna za mapowanie obiektów typu Client. Źródło: Opracowanie własne

```
@Component
@RequiredArgsConstructor
public class ClientMapper {

    private final CompanyRepository companyRepository;
    private final OrderRepository orderRepository;

    public ClientDto toDto(Client client) {
        return ClientDto.builder()
                .id(client.getId())
```

```
                .name(client.getName())
                .contactDetails(client.getContactDetails())
                .companyId(client.getCompany() == null ? null :
client.getCompany().getId())
                .orders(client.getOrders().stream()
                        .map(Orders::getId)
                        .collect(Collectors.toList()))
                .deleted(client.isDeleted())
                .build();
    }

    public Client toEntity(ClientDto clientDto) {

        List<Orders> ordersList = new ArrayList<>();
        clientDto.getOrders().forEach(orderId ->
ordersList.add(orderRepository.findById(orderId).orElseThrow(EntityNotF
oundException::new)));

        return Client.builder()
                .id(clientDto.getId())
                .name(clientDto.getName())
                .contactDetails(clientDto.getContactDetails())
                .company(clientDto.getCompanyId() == null ? null :
companyRepository.findById(clientDto.getCompanyId()).orElseThrow(Entity
NotFoundException::new))
                .orders(ordersList)
                .build();
    }
}
```

2. Zasada otwartego/zamkniętego (Open–closed principle - OCP)

OCP stwierdza, że jednostki oprogramowania (klasy, moduły, funkcje itp.) muszą być otwarte na rozbudowę, ale zamknięte na modyfikację. Innymi słowy, zachowanie modułu musi być rozszerzalne bez modyfikowania jego kodu źródłowego. Osiąga się to za pomocą abstrakcji, interfejsów i dziedziczenia. Kierując się tą zasadą, nowe funkcjonalności mogą być dodawane bez psucia istniejącego kodu. Listing 10 przedstawia przykład klasy nadrzędnej AppUser wraz z jej rozszerzeniem klasą Employee.

Listing 10 E-montażysta - Przykład klasy nadrzędnej AppUser i jej rozszerzenia Employee. Źródło: Opracowanie własne.

```
public class AppUser implements UserDetails {

//...

}

public abstract class Employee extends AppUser {

//...

}
```

3. Zasada podstawienia Liskowa (Liskov substitution principle - LSP)

LSP stwierdza, że obiekty nadklasy muszą być wymienialne na obiekty jej podklas bez wpływu na poprawność programu. Innymi słowy, jeśli program używa klasy bazowej, musi mieć możliwość nadpisania dowolnej klasy pochodnej bez powodowania nieoczekiwanego zachowania. Naruszenie tej zasady może prowadzić do nieprawidłowej logiki programu i delikatnego kodu. Listing 11 przedstawia przykład, w którym zastosowana została zasada LSP.

Listing 11 E-montażysta - Zasada LSP. Źródło: Opracowanie własne.

```
//obiekt nadklasy
public class AppUserDto {
    private Long id;
    @NotBlank(message = "First name cannot be empty")
    @Length(min = 3, max = 32, message = "First name has to be between
2 and 32 chars")
    private String firstName;
    @NotBlank(message = "Last name cannot be empty")
    @Length(min = 2, max = 32, message = "Last name has to be between 2
and 32 chars")
    private String lastName;
    @Email(message = "Email should be in valid format")
    @NotBlank(message = "Email cannot be empty")
    private String email;
    private String password;
    @NotBlank(message = "Username cannot be empty")
    @Length(min = 3, message = "Username must contain at least 3
characters" )
    private String username;
    private Set<Role> roles;
    private boolean deleted;
}

//klasa dziedzicząca
public class EmployeeDto extends AppUserDto {

    private Long id;
    @PlPhone
    private String phone;
    @PESEL(message = "PESEL is not valid")
    @NotBlank(message = "PESEL cannot be empty")
    private String pesel;
    private List<Long> unavailabilities;
    private List<Long> notifications;
    private List<Long> employeeComments;
    private List<Long> elementEvents;
    private List<Long> employments;
    private List<Long> attachments;
    private List<Long> toolEvents;
    private String status;
    private String unavailbilityDescription;
    private LocalDateTime unavailableFrom;
    private LocalDateTime unavailableTo;
    private boolean active;
}
```

```
//kolejna klasa dziedzicząca
public class FitterDto extends EmployeeDto {

    private List<Long> workingOn;
}

//metoda z parametrem obiektu nadklasy
public interface AppUserService extends UserDetailsService {
// …
    AppUser update(Long id, AppUser user);
//…
}
//…

//implementacja ww. metody
public class AppUserServiceImpl  implements AppUserService {
//…

    @Override
    public AppUser add(AppUser user) {
        user.setUsername(user.getUsername().toLowerCase());
        log.info("Saving new user {} to the
database",user.getUsername());

user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        return appUserRepository.save(user);
    }
//…
}
//…

AppUser appUser = new AppUser(null, "Test AppUser", "Test AppUser",
"em@i.l",
                "password", "testuser", null,
Set.of(Role.CLOUD_ADMIN));

        Fitter fitter1 = new Fitter(null, "Fitter1", "Montażysta1",
"fitter1@ema.il",
                "password", "fitter1", null, Set.of(Role.FITTER),
"+48123123123",
                "92122637693", new ArrayList<>(), new ArrayList<>(),
new ArrayList<>(), new ArrayList<>(),
                new ArrayList<>(), new ArrayList<>(), new
ArrayList<>(), new ArrayList<>());
        Fitter fitter2 = new Fitter(null, "Fitter2", "Montażysta2",
"fitter2@ema.il",
                "password", "fitter2", null, Set.of(Role.FITTER),
"+48231231231",
                "81081748975", new ArrayList<>(), new ArrayList<>(),
new ArrayList<>(), new ArrayList<>(),
                new ArrayList<>(), new ArrayList<>(), new
ArrayList<>(), new ArrayList<>());
        Fitter fitter3 = new Fitter(null, "Fitter3", "Montażysta3",
"fitter1@ema.il",
```

```
                "password", "fitter3", null, Set.of(Role.FITTER),
"+48312312312",
                "03261768113", new ArrayList<>(), new ArrayList<>(),
new ArrayList<>(), new ArrayList<>(),
                new ArrayList<>(), new ArrayList<>(), new
ArrayList<>(), new ArrayList

//…

 List<AppUser> appUserList = List.of(appUser, fitter1, fitter2,
fitter3, foreman1, foreman2, warehouseman1, warehouseman2,
                warehouseManager1, warehouseManager2, specialist1,
specialist2, salesRepresentative1, salesRepresentative2,
                manager1, manager2, companyAdmin1);

// użycie metody AppUser add(AppUser user);
appUserList.forEach(appUserService::add);
```

4. Zasada segregacji interfejsów (Interface segregation principle - ISP)

Użytkownicy nie powinni być zmuszani do polegania na interfejsach, których nie używają. Zachęca do tworzenia konkretnych interfejsów dostosowanych do potrzeb poszczególnych użytkowników, zamiast dużych, monolitycznych interfejsów. Sprzyja to luźnemu powiązaniu i pozwala polegać tylko na metodach, których naprawdę są potrzebne. Listing 12 przedstawia przykład interfejsu wraz z jego implementacją.

Listing 12 E-montażysta - Interfejs i jego implementacja. Źródło: Opracowanie własne.

```
//interfejs
public interface EmploymentService {

    List<EmploymentDto> getAllEmployeeEmployments(Long employeeId);
    EmploymentDto hire(Long employeeId);
    EmploymentDto dismiss(Long employeeId);
    Optional<EmploymentDto> getCurrentEmploymentByEmployeeId(Long
employeeId);
}

//implementacja interfejsu
public class EmploymentServiceImpl implements EmploymentService {

    private final EmploymentRepository repository;
    private final EmploymentMapper employmentMapper;
    private final AppUserRepository appUserRepository;

    @Override
    public List<EmploymentDto> getAllEmployeeEmployments(Long id) {
        AppUser employee = appUserRepository.getById(id);


        return employee.getEmployments().stream()
                .map(employmentMapper::toDto)
                .collect(Collectors.toList());
    }
```

```
    @Override
    public EmploymentDto dismiss(Long employeeId) {
        Optional<EmploymentDto> dismissingCurrentEmployment =
getCurrentEmploymentByEmployeeId(employeeId);

        if(dismissingCurrentEmployment.isPresent()) {
            Long loggedUserCompanyId =
getLoggedUser().getEmployments().get(0).getCompany().getId();


if(dismissingCurrentEmployment.get().getCompanyId().equals(loggedUserCo
mpanyId)) {
                Employment employment =
employmentMapper.toEntity(dismissingCurrentEmployment.get());
                employment.setDateOfDismiss(LocalDateTime.now());
                repository.save(employment);
                return null;
            }else {
                throw new
ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        }else {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
    }

    @Override
    public EmploymentDto hire(Long employeeId) {
        //Employee for which we set employment
        AppUser employee = appUserRepository.getById(employeeId);

        //Check if user is cloud admin
        if(employee.getRoles().contains(Role.CLOUD_ADMIN)) {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }

        //Get user company
        Long employeeCompanyId =
employee.getEmployments().get(0).getCompany().getId();

        //Logged user
        Long loggedUserCompanyId =
getCurrentEmploymentByEmployeeId(getLoggedUser().getId()).get().getComp
anyId();

        //Check if employee is from company
        if(!employeeCompanyId.equals(loggedUserCompanyId)) {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }

        //Check if employee is still working

if(getCurrentEmploymentByEmployeeId(employee.getId()).isPresent()) {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
```

```
        EmploymentDto employmentDto = EmploymentDto.builder()
                .dateOfEmployment(LocalDateTime.now())
                .companyId(loggedUserCompanyId)
                .employeeId(employeeId)
                .build();

        Employment employment =
repository.save(employmentMapper.toEntity(employmentDto));
        return employmentMapper.toDto(employment);
    }

    @Override
    public Optional<EmploymentDto>
getCurrentEmploymentByEmployeeId(Long employeeId) {
        Optional<Employment> employment =
Optional.ofNullable(repository.findByEmployeeIdAndDateOfDismissIsNull(e
mployeeId));
        if (employment.isPresent())
            return
Optional.ofNullable(employmentMapper.toDto(employment.get()));
        else
            return Optional.ofNullable(null);
    }

    private AppUser getLoggedUser() {
        String username =
SecurityContextHolder.getContext().getAuthentication().getName();
        AppUser appUser = appUserRepository.findByUsername(username);
        return appUser;
    }
}
```

5. Zasada odwrócenia zależności (Dependency inversion principle - DIP)

Moduły wysokiego poziomu nie powinny zależeć od modułów niskiego poziomu; oba muszą zależeć od abstrakcji. Kładzie nacisk na rozdzielenie i zachęca do używania interfejsów lub klas abstrakcyjnych do definiowania zależności. Zastosowanie tego rozwiązania pozwala na elastyczność w wyborze implementacji i ułatwia testowanie, ponieważ zależności można łatwo zastąpić spreparowanymi obiektami. Listing 13 przedstawia przykład zastosowania zasady DIP.

Listing 13 E-montażysta – Zastosowanie zasady DIP. Źródło: Opracowanie własne.

```
//interfejs serwisowy
public interface EmploymentService {

    List<EmploymentDto> getAllEmployeeEmployments(Long employeeId);
    EmploymentDto hire(Long employeeId);
    EmploymentDto dismiss(Long employeeId);
    Optional<EmploymentDto> getCurrentEmploymentByEmployeeId(Long
employeeId);
}
```

```
//klasa kontrolera
public class EmploymentController {

    private final EmploymentService employmentService;

    @GetMapping("/for-employee/{employeeId}")
    @Operation(description = "Allows to get all Employments of given
employee.", security = @SecurityRequirement(name = "bearer-key"))
    public List<EmploymentDto> getAllEmployeeEmployments(@PathVariable
Long employeeId) {
        return employmentService.getAllEmployeeEmployments(employeeId);
    }

    @PostMapping("/{employeeId}")
    @ResponseStatus(HttpStatus.CREATED)
    @Operation(description = "Allows to hire Employee of given id.",
security = @SecurityRequirement(name = "bearer-key"))
    public EmploymentDto hire(@PathVariable Long employeeId) {
        return employmentService.hire(employeeId);
    }

    @PutMapping("/{employeeId}")
    @Operation(description = "Allows to dismiss Employee of given id.",
security = @SecurityRequirement(name = "bearer-key"))
    public EmploymentDto dismiss(@PathVariable Long employeeId) {
        return employmentService.dismiss(employeeId);
    }
}
```

5.3. Implementacja aplikacji webowej – Frontend

Do implementacji warstwy aplikacji webowej użyto język TypeScript z wykorzystaniem biblioteki React do budowy interfejsu użytkownika. Dodatkowo, użyto języka znaczników HTML i arkusza styli CSS w celu definiowania wyglądu i stylizacji stron. Aplikacja webowa została rozwijana przy użyciu oprogramowania Visual Studio Code.

W strukturze aplikacji webowej, wyróżniają się trzy główne foldery: "public", "src" i ".build". Folder "public" pełni funkcję przechowywania plików publicznych, takich jak grafiki czy ikony. Znajdujące się w nim zasoby są dostępne publicznie i mogą być ładowane bez konieczności uwierzytelnienia. Natomiast folder "src" przechowuje wszystkie pliki źródłowe, które są wykorzystywane do budowy aplikacji. Dodatkowo, utworzono w nim podfoldery, które mają na celu organizację kodu na struktury logiczne. Folder ".build" pojawia się w strukturze projektu jako rezultat procesu budowania aplikacji.

W celu zapewnienia spójności przy tworzeniu interfejsu użytkownika, utworzono reużywalne komponenty, takie jak przyciski, pola tekstowe, okna dialogowe, składane formularze lub bardziej rozbudowane struktury, takie jak tabele czy etapy zlecenia. Te wszystkie komponenty zostały wykorzystywane w różnych miejscach, aby zapewnić jednolity wygląd oraz przyspieszyć i ułatwić proces projektowania i rozwijania aplikacji, co oszczędziło czas programistom.

5.3.1 Biblioteka Material-UI (mui)

Biblioteka Material-UI (mui) **Błąd! Nie można odnaleźć źródła odwołania.**, została w ykorzystana w implementacji aplikacji webowej do dostarczania gotowych komponentów interfejsu użytkownika zgodne z zasadami wzornictwa Material Design opracowanymi przez Google. W projekcie użyto kolekcję gotowych komponentów, takich jak przyciski, formularze, nawigacja, styli i wiele innych.

Jednym z przykładów użycia biblioteki Material-UI (mui) może być wykorzystanie komponentu CheckBoxIcon do reprezentacji zaznaczonego lub niezaznaczonego pola wyboru w interfejsie użytkownika. CheckBoxIcon dostarcza ikonę graficzną, która obrazuje stan zaznaczenia elementu. Listing 14 przedstawia przykład użycia tego komponentu.

Listing 14 E-montażysta – Przykład użycia biblioteki mui, komponent CheckBoxIcon. Źródło: Opracowanie własne.

```
import CheckBoxIcon from '@mui/icons-material/CheckBox'

 //...

 const checkedIcon = <CheckBoxIcon fontSize="small" />

 //...

 <Checkbox
      icon={icon}
      checkedIcon={checkedIcon}
      style={{ marginRight: 8 }}
      checked={selected}
      id={`multiselect-${id}-opt${option.key}`}
/>
//...
```

Innym przykładem jest użycie komponentu Typography, który dostarczył narzędzi do formatowania tekstu w interfejsie użytkownika. Umożliwił utrzymanie w projekcie jednolitych i spójnych stylów typograficznych. Listing 15 przedstawia przykład użycia tego komponentu.

Listing 15 E-montażysta – Przykład użycia biblioteki mui, komponent Typography. Źródło: Opracowanie własne.

```
import { Button, Typography } from '@mui/material'

 //...

<Typography variant="h5" sx={{ mb: '10px' }}>
                        {title}
                    </Typography>
//...
```

Skorzystanie z biblioteki Material-UI przyspieszyło proces tworzenia aplikacji poprzez dostarczenie gotowych rozwiązań. Dodatkowo, umożliwiła dostosowanie komponentów do potrzeb projektu.

5.3.2 Biblioteka Emotion

Biblioteka Emotion **Błąd! Nie można odnaleźć źródła odwołania.**, została użyta w projekcie, a by zdefiniować style CSS bezpośrednio w kodzie JavaScript. Umożliwiło to lepszą kontrolę nad stylami i jego zmianami w zależności od logiki aplikacji. W przeciwieństwie do tradycyjnego podejścia, gdzie style są oddzielone od logiki w osobnych plikach CSS, Emotion umożliwia łączenie tych dwóch aspektów w jednym miejscu.

W przykładzie na listingu 16 wykorzystano bibliotekę Emotion do stylizacji komponentu CustomSelect. Wewnątrz funkcji styled, zdefiniowano obiekt styli jako argument funkcji. Wykorzystano tu składnię JavaScript, aby dynamicznie tworzyć style w zależności od wartości przekazanych przez props.

Listing 16 E-montażysta – Przykład użycia biblioteki Emotion, komponent CustomSelect. Źródło: Opracowanie własne.

```
import styled from '@emotion/styled'

 //...
const CustomSelect = styled(Select)((props: CustomSelectProps) => ({
    '& fieldset': {
        border: props.readOnly ? 'none' : '',
    },
    '& svg': {
        display: props.readOnly ? 'none' : '',
    },
    '& div': {
        cursor: props.readOnly ? 'unset' : 'pointer',
    },
}))
//...
```

5.3.3 Biblioteka Axios

Biblioteka Axios **Błąd! Nie można odnaleźć źródła odwołania.**, została wykorzystana w p rojekcie do obsługi żądań sieciowych, takich jak pobieranie czy wysyłanie danych z i do serwera. Głównym jej celem było ułatwienie procesu wykonywania zapytań http.

Na listingu 17 przedstawiono przykład użycia biblioteki Axios wraz z useQuery, który przyjmuje typ błędu AxiosError. Rozwiązanie to umożliwia kontrolowanie i reagowanie na błędy związane z żądaniami sieciowymi.

Listing 17 E-montażysta – Przykład użycia biblioteki AxiosError. Źródło: Opracowanie własne.

```
import { AxiosError } from 'axios'

 //...
const queryToolHistory = useQuery<Array<ToolHistory>, AxiosError>(
        ['tool-history'],
        async () => getToolHistory(params.toolId && params.toolId !==
'new' ? params.toolId : ''),
        {
            enabled: !!params.toolId && params.toolId !== 'new',
        },
    )
//...
```

Na listingu 18 przedstawiono przykład użycia biblioteki Axios do wykonywania żądań typu GET i POST.

Listing 18 E-montażysta – Przykład użycia biblioteki Axios. GET i POST. Źródło: Opracowanie własne.

```
import axios from 'axios'

 //...
 const reposnose = await axios(url, {
        method: 'GET',
        headers: {
            'Access-Control-Allow-Origin': 'https://o2cj2q.csb.app',
        },
//...
const response = await axios(url, {
        method: 'POST',
        headers: {
            'Access-Control-Allow-Origin': 'https://o2cj2q.csb.app',
        },
//...
```

5.3.4 Biblioteka Formik

Biblioteka Formik **Błąd! Nie można odnaleźć źródła odwołania.** została użyta w projekcie do z arządzania formularzami. Umożliwiła ich tworzenie, walidację i obsługę w interfejsie użytkownika.

Na listingu 19 widnieje komponent FormLabel, który jest zaimplementowany przy użyciu biblioteki Formik. Przez użycie formik.touched[id] sprawdza się, czy dane pole formularza zostało użyte przez użytkownika. Natomiast formik.errors[id] sprawdza, czy istnieją błędy związane z danym polem. Na tej podstawie ustala się wartość hasError, która oznacza, czy pole ma błędy.

Listing 19 E-montażysta – Przykład użycia biblioteki Formik. komponent FormLabel. Źródło: Opracowanie własne.

```
const FormLabel = (props: FormLabelProps) => {
    const { label, formik, id } = props
    const hasError = formik.touched[id] && Boolean(formik.errors[id])
    return <Typography m={2} style={{ marginBottom: hasError ? '39px' :
'' }}>{label}</Typography>
}
export default FormLabel
```

5.3.5 Biblioteka Leaflet

Biblioteka Leaflet **Błąd! Nie można odnaleźć źródła odwołania.** została wykorzystana w p rojekcie do implementacji interaktywnej mapy umożliwiając użytkownikom nawigację, powiększanie, przesuwanie i wyświetlanie informacji na podstawie znaczników na mapie. W tym przypadku dane pochodzą ze strony internetowej OpenStreetMap. Przedstawiony przykład na listingu 20 ilustruje użycie biblioteki w komponencie Map, który jest odpowiedzialny za renderowanie mapy i jej elementów.

Listing 20 E-montażysta – Przykład użycia biblioteki Leaflet. komponent Map. Źródło: Opracowanie własne.

```
import { MapContainer, TileLayer, Marker, Popup, useMap } from 'react-
leaflet'
import 'leaflet/dist/leaflet.css'
```

```
import L from 'leaflet'
//...
<MapContainer center={[coords.lat, coords.lon]} zoom={15}
scrollWheelZoom={true} style={style}>
            <TileLayer
                attribution='&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a>
        contributors'

url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            />
            <Marker icon={customIcon} position={[coords.lat,
coords.lon]}>
                <Popup>{popupText}</Popup>
            </Marker>
            <MapView coords={coords}
handleCoordinatesChange={handleCoordinatesChange} pageMode={pageMode}
/>
        </MapContainer>
//...
```

W komponencie Map, użyto komponentów dostarczonych przez bibliotekę Leaflet. TileLayer definiuje warstwę kafelków mapy, Marker reprezentuje znacznik na mapie, a Popup wyświetla dymek z treścią informacyjną.

5.3.6 Biblioteka react-qr-code

Biblioteka react-qr-code **Błąd! Nie można odnaleźć źródła odwołania.** została wykorzystana w p rojekcie do generowania kodów QR zapisanego komponencie QRCodeLabel. Listing 21 prezentuje przykład użycia tej biblioteki przedstawiając sposób tworzenia etykiety zawierającej tekst oraz odpowiadający mu kod QR.

Listing 21 E-montażysta – Przykład użycia biblioteki react-qr-code. komponent QRCodeLabel. Źródło: Opracowanie własne.

```
import QRCode from 'react-qr-code'
//...
const QRCodeLabel = ({ label, code }: LabelTypes) => {
    return (
        <Box
            bgcolor="white"
            width={500}
            textAlign="center"
            marginLeft="auto"
            marginRight="auto"
            padding="10px"
            borderRadius="5px"
        >
            <Typography
                fontWeight="bold"
                color="#15171F"
                fontSize={40}
                lineHeight={1}
                marginBottom="10px"
                marginTop={0}
                maxWidth="500px"
                style={{ wordWrap: 'break-word' }}
```

```
            > {label}
            </Typography>
            <QRCode value={code} size={192} />
        </Box> )}
export default QRCodeLabel
```

W komponencie QRCodeLabel, przekazujemy parametry label i code, które określają etykietę oraz wartość kodu QR. Następnie, użyto komponentu Box, w którym zdefiniowano różne właściwości, takie jak kolor tła, szerokość, wyśrodkowanie tekstu, marginesy czy zaokrąglenie rogów, aby dostosować wygląd etykiety QR. Na końcu, użyto komponentu QRCode i przekazano mu wartość code oraz rozmiar size jako parametry, aby wygenerować kod QR na podstawie przekazanych wartości. Rysunek 53 przedstawia przykład wygenerowanej etykiety z kodem QR dla testowego narzędzia.

Rysunek 53 E-montażysta - Przykład wygenerowanej etykiety z kodem QR. Źródło: Opracowanie własne.

5.3.7 Biblioteka jwt-decode

Biblioteka jwt_decode [54] została użyta do dekodowania tokenów JSON Web Token (JWT) w celu wyciągnięcia informacji, takich jak role użytkownika czy data wygaśnięcia. W przykładzie kodu na listingu 22, jwt_decode została wykorzystana w narzędziu utils/token.ts. Zaimplementowane zostały dwie funkcje, które wykorzystują tę bibliotekę do analizy i wyciągania informacji z tokena JWT.

Pierwsza funkcja getRolesFromToken pobiera token i dekoduje go przy użyciu jwt_decode. Następnie odczytuje pole scope ze zdekodowanego tokenu, które zawiera informacje o rolach użytkownika. Funkcja zwraca tablicę z rozdzielonymi rolami użytkownika.

Druga funkcja isExpire służy do sprawdzenia, czy token wygasł. Jeśli nie zostanie przekazany żaden token lub będzie nieważny, funkcja zwróci wartość true. W przeciwnym razie, funkcja dekoduje token za pomocą biblioteki jwt_decode i odczytuje pole exp, które zawiera informację o czasie wygaśnięcia. Następnie porównuje bieżącą datę z czasem wygaśnięcia tokenu. Jeśli bieżąca data jest większa lub równa czasowi wygaśnięcia tokenu, funkcja zwraca true, w przeciwnym razie zwraca false.

Listing 22 E-montażysta – Przykład użycia biblioteki jwt-decode. komponent QRCodeLabel. Źródło: Opracowanie własne.

```
import jwt_decode from 'jwt-decode'
//...
export const getRolesFromToken = () => {
    const token = getToken()
    if (!token) return []
    const decodedToken: DecodedTokenType = jwt_decode(token)
    const decodedUserRoles = decodedToken.scope.split(' ')
    return decodedUserRoles
}
```

```
//...
export const isExpire = (token?: string) => {
    if (!token) {
        return true
    }
    const decodedToken: DecodedTokenType = jwt_decode(token)
    const { exp } = decodedToken
    if (Date.now() >= exp * 1000) {
        return true
    }
    return false
}

//...
```

5.3.8 Biblioteka react-router

Do nawigacji w obrębie aplikacji wykorzystujemy RouterProvider z użyciem metody createBrowserRouter z biblioteki react-router [57]. Struktura stron aplikacji jest przechowywana w liście zawierającej obiekty typu PageProps, które następnie są mapowane do typu przyjmowanego przez createBrowserRouter. Listing 23 przedstawia funkcje pomocniczą PageProps.

Listing 23 E-montażysta –Funkcja pomocnicza PageProps. Źródło: Opracowanie własne.

```
type PageProps = {

      name: string

      path: string

      allowedRoles?: Array<Role>

      children?: Array<PageProps>

      component?: JSX.Element

      inNav: boolean

}
      children?: Array<PageProps>
      component?: JSX.Element
      inNav: boolean
}
```

Funkcja pomocnicza PageProps posiada następujące parametry:

- name: Nazwa wyświetlana w menu nawigacyjnym.
- path: URL, pod którym dostępna jest strona.
- allowedRoles (opcjonalny): Lista ról posiadających dostęp do strony.
- children (opcjonalny): Lista dostępnych podstron.
- component (opcjonalny): Komponent wyświetlany na stronie.
- inNav: Flaga określająca, czy strona ma być wyświetlana w menu nawigacyjnym.

Metoda browserRouterMapper służy do mapowania listy PageProps na listę RouteObject. Przyjmuje ona listę stron pages i zwraca zmapowaną listę RouteObject.

Listing 24 przedstawia przykład implementacji metody browserRouterMapper.

Listing 24 E-montażysta – Metoda mapująca listę PageProps na listę RouteObjects. Źródło: Opracowanie własne.

```
const browserRouterMapper = (pages: Array<PageProps>):
Array<RouteObject> => {
    return pages.map((page) => {
        if (page.component)
            return {
                element: <AutorizedRoute {...page} />,
                children: [
                    {
                        path: page.path,
                        element: page.component,
                        children: page.children ?
                            browserRouterMapper(page.children) :
undefined,
                    },
                ],
            }
        else return {}
    })
}
    return pages.map((page) => {
        if (page.component)
            return {
                element: <AutorizedRoute {...page} />,
                children: [
                    {
                        path: page.path,
                        element: page.component,
                        children: page.children ?
                            browserRouterMapper(page.children) :
undefined,
                    },
                ],
            }
        else return {}
    })
}
```

5.4. Implementacja aplikacji mobilnej – Android

Przy tworzeniu aplikacji mobilnej wykorzystano wzorzec projektowy MVVM **Błąd! Nie można o dnaleźć źródła odwołania.**, czyli Model – View – ViewModel oraz takie narzędzia jak Koin i Retrofit, które zostało opisane szerzej w następnych podrozdziałach. Aplikacja została napisana w Kotlinie ze względu, że jest to język dedykowany dla platformy Android, rekomendowany przez Google LLC.

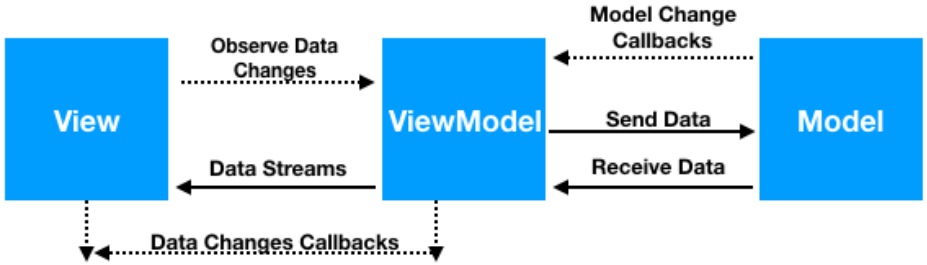
5.4.1 Wzorzec Model-View-ViewModel

Wzorzec MVVM według **Błąd! Nie można odnaleźć źródła odwołania.** jest aktualnie jednym z najbardziej popularnych wzorców projektowych wśród programistów. Zastąpił on w przeszłości używany wzorzec Model-View-Presenter.

Składa się on z trzech podstawowych składników:

- Model
- Widok
- Model Widoku

Każdy z nich służy innemu celowi, a na rysunku 54 widnieją powiązania między nimi. Użycie tego wzorca pozwoliło nam na zachowanie przejrzystości kodu i separacji ze względu na funkcjonalność.



Rysunek 54 Wzorzec architektury MVVM w Androidzie. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***

Na przedstawionym rysunku można zauważyć, że model w architekturze MVVM nie jest bezpośrednio powiązany z widokiem. Zawiera dane i logikę biznesową, ale nie zawiera informacji na którym widoku zostanie użyty. Jest on całkowicie niezależny od prezentacji i interakcji z użytkownikiem.

Widok pełni rolę odpowiedzialną za prezentację danych pochodzących z modelu oraz reagowanie na interakcje użytkownika. Jest to warstwa, która wykorzystuje powiązania danych w celu odczytania informacji z modelu i automatycznego ich odświeżania w przypadku zmiany.

Model Widoku pełni kluczową rolę jako warstwa pośrednicząca między Modelem a Widokiem. Jest odpowiedzialny za przechowywanie danych oraz logikę związaną z prezentacją i obsługą interakcji z widokiem. Komunikacja między Widokiem a Modelem Widoku odbywa się za pomocą różnych mechanizmów, w tym powiązań danych, powiązań poleceń i powiązań zdarzeń. Listing 25 przedstawia przykład kodu kompomentu ViewModel dla ekranu DashBoard.

Listing 25 E-montażysta - ViewModel w języku Kotlin dla ekranu Dashboard. Źródło: Opracowanie własne.

```
class DashboardViewModel(private val userRepository:
IUserRepository,
) : ViewModel(),
    KoinComponent,
    CoroutineScope {

    private var job: Job? = null
```

```
    private val _currentUserLiveData =
MutableLiveData<CurrentUser>()
    val currentUser: LiveData<CurrentUser> get() =
_currentUserLiveData

    private val _messageLiveData = MutableLiveData<String>()
    val messageLiveData: LiveData<String> = _messageLiveData

    private val _isLoadingLiveData = MutableLiveData<Boolean>()
    val isLoadingLiveData: LiveData<Boolean> = _isLoadingLiveData

    fun getCurrentUser() {
        job = launch {
            getCurrentUserAsync()
        }
    }

    private suspend fun getCurrentUserAsync() {
        _isLoadingLiveData.postValue(true)
        val result = userRepository.getAboutMe()
        when (result) {
            is Result.Success -> {
                val currentUser = result.data
                _currentUserLiveData.postValue(currentUser)
            }

            is Result.Error -> {
                result.exception.message?.let {
_messageLiveData.postValue(it) }
                _isLoadingLiveData.postValue(false)
            }
        }
        _isLoadingLiveData.postValue(false)
    }

    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main
}
```

5.4.2 Koin

W projekcie aplikacji mobilnej zdecydowano się na użycie lekkiego frameworka jakim jest Koin [56] do realizacji zależności (DI - Dependency Injection) w języku Kotlin. Zapewnia prosty sposób definiowania zależności między komponentami. Dzięki temu możemy deklarować moduły, w których określamy, jakie obiekty mogą być dostępne dla innych komponentów. Framework ten automatycznie tworzy i dostarcza te obiekty do wskazanych przez nas metod. Dzięki temu proces zarządzania zależnościami staje się bardziej przejrzysty i efektywny, co przekłada się na lepszą organizację kodu i łatwiejsze testowanie aplikacji. Na listingu 26 przedstawiono przykład użycia frameworka Koin do pozyskania danych o magazynach za pomocą Rest API.

Listing 26 E-montażysta - Zastosowanie framworka Koin w modelu Warehouse. Źródło: Opracowanie własne.

```
data class Warehouse(
```

```
    val companyId: Int,
    val deleted: Boolean,
    val description: String,
    val elementInWarehouses: List<Int>,
    val id: Int,
    val locationId: Int?,
    val name: String,
    val openingHours: String,
    val tools: List<Int>
){
    companion object: KoinComponent {
        val warehouseRepository: IWarehouseRepository by inject()
        suspend fun getWarehouseDetails(id: Int) : Warehouse {
            val result = warehouseRepository.getWarehouseDetails(id)
            return when(result){
                is Result.Success -> result.data
                is Result.Error -> throw result.exception
            }
        }
    }
}
```

5.4.3 Retrofit

Do komunikacji z backendem postanowiono na użycie popularnej biblioteki Retrofit [47]. Uproszcza ona komunikację między aplikacją a serwerem API, poprzez mapowanie zapytań HTTP na metody w interfejsach. Biblioteka buduje żądanie http, obsługę błędów, czy parsowanie odpowiedzi. Na listingu 27 przedstawiono użycie tej biblioteki w projekcie.

Listing 27 E-montażysta - Zastosowanie biblioteki Retrofit EventService. Źródło: Opracowanie własne.

```
interface EventService {
    @GET("/api/v1/events/filter")
    suspend fun getFilteredEvents(@Header("Authorization") token:
String): List<FilterEventDAO>

    @GET("/api/v1/events/filter")
    suspend fun getFilteredEvents(@Header("Authorization") token:
String, @QueryMap(encoded = true) payload: Map<String, String>?):
List<FilterEventDAO>

    @GET("/api/v1/toolEvent/{id}")
    suspend fun getToolEventDetails(@Header("Authorization") token:
String, @Path("id") id: Int): ToolEventDAO

    @PUT("/api/v1/toolEvent/{id}")
    suspend fun updateToolEvent(@Header("Authorization") token: String,
@Path("id") id: Int, @Body data: Element): ToolEventDAO

    @POST("/api/v1/toolEvent")
    suspend fun postToolEvent(@Header("Authorization") token: String,
@Body data: Element): ToolEventDAO

    @DELETE("/api/v1/toolEvent/{id}")
```

```
    suspend fun deleteToolEvent(@Header("Authorization") token: String,
@Path("id") id: String): ToolEventDAO

    @GET("/api/v1/elementEvent/{id}")
    suspend fun getElementEventDetails(@Header("Authorization") token:
String, @Path("id") id: String?): ElementEventDAO

    @PUT("/api/v1/elementEvent/{id}")
    suspend fun updateElementEvent(@Header("Authorization") token:
String, @Path("id") id: String, @Body data: Element): ElementEventDAO

    @POST("/api/v1/elementEvent")
    suspend fun postElementEvent(@Header("Authorization") token:
String, @Body data: Element): ElementEventDAO

    @DELETE("/api/v1/elementEvent/{id}")
    suspend fun deleteElementEvent(@Header("Authorization") token:
String, @Path("id") id: String): ElementEventDAO
}
```

5.5. Implementacja części serwerowej – Backend

Do implementacji części serwerowej zespół wykorzystał wzorzec projektowy MVC (Model-View-Controller). Logika działania oparta jest o klasy modelowe, kontrolery oraz interfejsy serwisowe. Kontrolery obsługują komunikację z użytkownikiem, serwisy zawierają logikę biznesową, a klasy modelowe hermetyzują dane. Do transferu danych wewnątrz aplikacji wykorzystano obiekty typu DTO (Data Transfer Objects). Oprogramowanie zostało zaprojektowane zgodnie z zasadami SOLID. Do komunikacji z bazą danych wykorzystano frameworki Spring Data oraz Java Persistence API (JPA). Zastosowano również Criteria API do tworzenia dynamicznych zapytań.

5.5.1 DTO

Obiekty transportu danych (DTO - Data Transfer Object) według **Błąd! Nie można odnaleźć źródła odwołania.** są integralną częścią tworzenia oprogramowania, szczególnie w dziedzinie systemów rozproszonych, zorientowanych na usługi. DTO służą jako sposób enkapsulacji i przesyłania danych między różnymi warstwami aplikacji lub ponad granicami sieci. DTO można traktować jako lekkie obiekty podobne do kontenerów, które transportują dane z jednej warstwy lub komponentu do innego bez ujawniania podstawowych szczegółów implementacji. Działają jako pośrednik między źródłem danych (takim jak baza danych lub usługa zewnętrzna) a konsumentami tych danych. Listing 28 przedstawia przykład użycia DTO dla projektu.

Listing 28 E-montażysta - Przykład DTO dla klasy AppUserDto. Źródło: Opracowanie własne.

```
public class AppUserDto {

    private Long id;
    @NotBlank(message = "First name cannot be empty")
    @Length(min = 3, max = 32, message = "First name has to be between
2 and 32 chars")
    private String firstName;
    @NotBlank(message = "Last name cannot be empty")
    @Length(min = 2, max = 32, message = "Last name has to be between 2
and 32 chars")
    private String lastName;
```

```
    @Email(message = "Email should be in valid format")
    @NotBlank(message = "Email cannot be empty")
    private String email;
    private String password;
    @NotBlank(message = "Username cannot be empty")
    @Length(min = 3, message = "Username must contain at least 3
characters" )
    private String username;
    private Set<Role> roles;
    private boolean deleted;
}
```

Głównym celem DTO jest ułatwienie przesyłania danych w formacie odpowiednim do wykorzystania przez końcowego odbiorcę. Pomagają wyeliminować zależności od określonych modeli lub struktur danych, umożliwiając rozdzielenie między nadawcą a odbiorcą. Oddzielenie jest szczególnie ważne w systemach rozproszonych, gdzie różne komponenty mogą być napisane w różnych językach programowania lub mieć różne wymagania dotyczące reprezentacji danych. Korzystanie z DTO pozwala chronić wewnętrzne struktury danych i modele domen. Hermetyzacja zapewnia elastyczność w dokonywaniu zmian w wewnętrznej implementacji bez wpływu na zewnętrzne interfejsy. Na przykład, jeśli zmieni się wewnętrzny model danych usługi, tylko odpowiednia DTO musi zostać zaktualizowana, podczas gdy klienci zewnętrzni pozostają niezmienieni. Listing 29 przedstawia przykład wykorzystania DTO w projekcie.

Listing 29 Przykład wykorzystania DTO. Źródło: Opracowanie własne.

```
//klasa DTO
public class WarehouseDto {

    private Long id;
    @NotBlank(message = "Name cannot be empty")
    private String name;
    private String description;
    @NotBlank(message = "Opening hours cannot be empty")
    private String openingHours;
    private Long companyId;
    private Long locationId;
    private List<Long> elementInWarehouses;
    private List<Long> tools;
    private boolean deleted;
}

//...

//metoda klasy serwisowej wykorzystująca DTO
public class WarehouseMapper {

//...

public WarehouseDto toDto(Warehouse warehouse) {
        return WarehouseDto.builder()
                .id(warehouse.getId())
                .name(warehouse.getName())
                .description(warehouse.getDescription())
                .openingHours(warehouse.getOpeningHours())
                .companyId(warehouse.getCompany() == null ? null :
warehouse.getCompany().getId())
```

```
                .locationId(warehouse.getLocation() == null ? null :
warehouse.getLocation().getId())

.elementInWarehouses(warehouse.getElementInWarehouses().stream()
                        .map(ElementInWarehouse::getId)
                        .collect(Collectors.toList()))
                .tools(warehouse.getTools().stream()
                        .map(Tool::getId)
                        .collect(Collectors.toList()))
                .deleted(warehouse.isDeleted())
                .build();
    }

//...

}
```

DTO ułatwiają wydajne przesyłanie danych poprzez zmniejszenie ilości szczegółów wymienianych między komponentami. Zamiast przenosić całe obiekty domeny, które mogą zawierać niepotrzebne lub poufne informacje, DTO umożliwiają selektywny i usprawniony transfer tylko niezbędnych danych. Może to znacznie poprawić wydajność i zmniejszyć obciążenie sieci, szczególnie w scenariuszach, w których przepustowość jest ograniczona lub występują opóźnienia. Listing 30 przedstawia przykładowe wykorzystanie tylko niezbędnych danych w DTO.

Listing 30 E-montażysta - Przykład wykorzystania tylko niezbędnych danych w DTO. Źródło: Opracowanie własne.

```
//pola w klasie Tool
public class Tool {

//...

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private LocalDateTime createdAt;

    private String code;

    private boolean deleted = Boolean.FALSE;

    @OneToMany(mappedBy = "tool")
    private List<ToolRelease> toolReleases;

    @ManyToOne
    private Warehouse warehouse;

    @OneToMany(mappedBy = "tool")
    private List<ToolEvent> toolEvents;

    @ManyToOne
    private ToolType toolType;
```

```
//...

}

//W DTO przekazujemy tylko niezbędne informacje
public class ToolFilterDto {

    private Long id;
    private String name;
    private String code;
    private String warehouse;
    private String toolType;
}
```

Podczas wdrażania DTO kluczowe jest przestrzeganie pewnych najlepszych praktyk. Po pierwsze, DTO powinny być proste i skupiać się wyłącznie na przesyłaniu danych. Nie mogą zawierać logiki biznesowej ani zachowań. Celem DTO jest tylko transport danych. Ponadto DTO muszą umożliwiać serializację do formatu przyjaznego dla transportu, takiego jak JSON, co pozwala na łatwą konwersję do i z ich serializowanej reprezentacji.

DTO nie należy mylić z obiektami lub encjami domeny. Podczas gdy obiekty domeny reprezentują logikę biznesową i zachowanie aplikacji, DTO skupiają się wyłącznie na przesyłaniu danych i nie powinny zawierać żadnych konkretnych metod biznesowych. Utrzymanie wyraźnego oddzielenia tych koncepcji ma kluczowe znaczenie dla skalowalnej i zrównoważonej architektury oprogramowania.

5.5.2 Criteria API

Interfejs Criteria API według **Błąd! Nie można odnaleźć źródła odwołania.** to zaawansowana f unkcja języka Java umożliwiająca tworzenie dynamicznych zapytań. Zapewnia bezpieczny i płynny interfejs API do tworzenia złożonych zapytań do baz danych bez polegania na zapytaniach SQL opartych na łańcuchach. Interfejs Criteria API jest częścią specyfikacji Java Persistence API (JPA), która jest powszechnie używana do mapowania obiektowo-relacyjnego (ORM) w aplikacjach Java.

Jedną z głównych zalet korzystania z Criteria API jest bezpieczeństwo zgodności typu. Zapytania są budowane przy użyciu klas i metod Java, co gwarantuje, że struktura zapytania jest sprawdzana w czasie kompilacji. Pomaga to wychwycić błędy na wczesnym etapie procesu programowania i zapewnia bardziej niezawodne i łatwiejsze w utrzymaniu podejście w porównaniu z zapytaniami opartymi na ciągach. Funkcjonalność zapewnia również czytelność kodu i łatwość konserwacji. Korzystając z płynnej, połączonej składni metod można zwięźle wyrażać złożone warunki zapytań. Interfejs API zapewnia m.in. metody filtrowania, sortowania, łączenia tabel, definiowania agregacji ułatwiając wyrażanie złożonej logiki zapytań w uporządkowany i zrozumiały sposób. Criteria API umożliwia również dynamiczną budowę zapytań w czasie wykonywania, co pozwala na dostosowanie warunków zapytania na podstawie danych wprowadzonych przez użytkownika lub innych czynników czasu wykonywania. Listing 31 przedstawia przykład zastosowania Criteria API w projekcie.

Listing 31 E-montażysta - Zastosowanie Criteria API. Źródło: Opracowanie własne.

```
//Przykład wykorzystania Criteria API
@Repository
public class ToolTypeCriteriaRepository {

    private final EntityManager entityManager;
    private final CriteriaBuilder criteriaBuilder;
    private final ToolTypeMapper toolTypeMapper;
```

```
    private final AuthUtils authUtils;

    public ToolTypeCriteriaRepository(EntityManager entityManager,
ToolTypeMapper toolTypeMapper, AuthUtils authUtils) {
        this.entityManager = entityManager;
        this.criteriaBuilder = entityManager.getCriteriaBuilder();
        this.toolTypeMapper = toolTypeMapper;
        this.authUtils = authUtils;

    }

    public List<ToolTypeDto> findAllWithFilters(ToolTypeSearchCriteria
toolTypeSearchCriteria ) {

        CriteriaQuery<ToolType> criteriaQuery =
criteriaBuilder.createQuery(ToolType.class);
        Root<ToolType> toolTypeRoot =
criteriaQuery.from(ToolType.class);
        Predicate predicate = getPredicate(toolTypeSearchCriteria,
toolTypeRoot);
        criteriaQuery.where(predicate);

        TypedQuery<ToolType> typedQuery =
entityManager.createQuery(criteriaQuery);
        List<ToolType> toolTypes = typedQuery.getResultList();

        return
toolTypes.stream().map(toolTypeMapper::toDto).collect(Collectors.toList
());
    }
    private Predicate getPredicate(ToolTypeSearchCriteria
toolTypeSearchCriteria, Root<ToolType> toolTypeRoot) {
        List<Predicate> predicates = new ArrayList<>();

        //Get tooltypes by name
        if(Objects.nonNull(toolTypeSearchCriteria.getName())){

predicates.add(criteriaBuilder.like(toolTypeRoot.get("name"), "%" +
toolTypeSearchCriteria.getName() + "%"));
        }

        //Get tooltypes from company

predicates.add(criteriaBuilder.equal(toolTypeRoot.get("company").get("i
d"),
                authUtils.getLoggedUserCompanyId()));

        //Get not-deleted

predicates.add(criteriaBuilder.equal(toolTypeRoot.get("deleted"),
false));

        return  criteriaBuilder.and(predicates.toArray(new
Predicate[0]));
    }
}
```

5.5.3 Resetowanie hasła

W celu implementacji funkcjonalności resetowania hasła użytkownika systemu, proces został podzielony na dwa etapy, zapewniające bezpieczne i skuteczne odzyskiwanie dostępu.

Etap 1: Inicjowanie resetowania hasła

1. Użytkownik, który nie jest zalogowany, klikając odpowiedni przycisk, inicjuje proces resetowania hasła.

2. Następnie użytkownik jest proszony o podanie swojej nazwy, którą wykorzystuje do logowania.

3. Podana informacja trafia do serwera, który wyszukuje w bazie danych przypisany do niej adres e-mail.

4. Generowany jest jednorazowy token, który jest zapisywany w rekordzie użytkownika.

5. System tworzy wiadomość e-mail zawierającą link, w którym umieszczony jest wspomniany token, a następnie wysyła tę wiadomość na skrzynkę pocztową użytkownika.

Etap 2: Ustawianie nowego hasła

1. Użytkownik, otrzymując wiadomość e-mail, klikając w link, przechodzi do drugiego etapu resetowania hasła.

2. Na stronie docelowej użytkownik zostaje poproszony o podanie nowego hasła.

3. Na serwer, wraz z nowym hasłem, przekazywany jest również wcześniej otrzymany token.

4. Serwer odszukuje rekord użytkownika, który zawiera ten token.

5. Wartość hasła w rekordzie użytkownika zostaje zaktualizowana na nowe hasło podane przez użytkownika.

6. W ten sposób użytkownik może zalogować się za pomocą nowo utworzonego hasła i odzyskać pełny dostęp do systemu.

Dzięki podziałowi procesu resetowania hasła na dwa etapy, użytkownik jest chroniony przed nieautoryzowanym dostępem do swojego konta, a resetowanie hasła odbywa się w sposób bezpieczny i kontrolowany.

5.5.4 Usuwanie nietrwałe

Usuwanie nietrwałe znane jest również pod określeniem usuwania wirtualnego. Jest to technika, której używa się w systemach informatycznych, aby zamiast natychmiastowego usuwania rekordów z bazy danych, oznaczać je jako usunięte. Dzięki temu obiekt w rzeczywistości pozostaje dostępny w systemie i nadal można na nim wykonywać operacje.

W przeciwieństwie do trwałego usuwania rekordów, wprowadza się dodatkowe pole w strukturze danych, którego wartość zależy od tego, czy obiekt został usunięty. Pole to jest ustawiane na wartość "true" lub "false". Dzięki temu rozwiązaniu usunięte rekordy będą niewidoczne w większości operacji wyszukiwania, jednak wciąż będzie możliwe odwołanie się do nich za pomocą określonych operacji.

Usuwanie nietrwałe niesie za sobą możliwości:

- Przywracania danych – w przypadku przypadkowego usunięcia danych lub potrzeby odzyskania informacji, oznaczenie rekordu zamiast jego usuwania umożliwia szybkie przywrócenie danych.

- Śledzenia operacji - przechowując obiekty “usunięte” istnieje możliwość monitorowania kto, kiedy i dlaczego dokonał usunięcia informacji.
- Ochrony przed nieodwracalnymi błędami - zapobiega przypadkowym i nieodwracalnym usunięciom cennych informacji lub naruszenia integralności danych

Implementując tę funkcjonalność, do struktury danych dodane zostało pole “deleted” typu boolean. Każdy tworzony obiekt inicjalizuje te pole wartością “false“. Informacja ta została dodana do modeli przesyłanych przez serwer do warstwy wizualnej. W przypadku usunięcia rekordu, wartość pola “deleted” zmieniana jest na “true“.

Oprócz ww. korzyści zastosowanie funkcjonalności wiąże się również z pewnymi wyzwaniami:

- Implementacja funkcji usuwania nietrwałego zwiększa złożoność logiki zarządzania danymi. Wymaga dodatkowych kontroli i warunków, aby upewnić się, że usunięte rekordy są prawidłowo obsługiwane w zapytaniach, filtrach i logice biznesowej.
- Implementacja funkcji usuwania nietrwałego zwiększa złożoność logiki zarządzania danymi. Wymaga dodatkowych kontroli i warunków, aby upewnić się, że usunięte rekordy są prawidłowo obsługiwane w zapytaniach, filtrach i logice biznesowej.
- Zwiększone wymagania dotyczące pamięci z uwagi na brak rzeczywistego usunięcia rekordów. Problem ten może wystąpić w szczególności w przypadku aplikacji z dużą ilością danych.
- Usuwanie nietrwałe może wpływać na wydajność zapytań i operacji na bazie danych. Na przykład zapytania muszą zawierać warunki odfiltrowywania tymczasowo usuniętych rekordów, co może wpływać na plany wykonywania zapytań i spowalniać wydajność, zwłaszcza w przypadkach, gdy zaangażowane są duże zestawy danych.
- Może wystąpić również problem z integralnością danych. Na przykład, jeśli między tabelami istnieją ograniczenia klucza obcego, usunięcie rekordu w jednej tabeli może spowodować konflikt z metodą usuwania nietrwałego w innej tabeli. Może to prowadzić do niespójności i potencjalnie komplikować procesy konserwacji i czyszczenia danych.
- W zależności od charakteru aplikacji i wymagań prawnych, które należy spełnić, funkcja usuwania nietrwałego może nie być wystarczająca do celów audytu i przechowywania danych. Niektóre przepisy mogą wymagać całkowitego usunięcia danych, a nie tylko oznaczenia ich jako usuniętych.

5.5.5 Zapisywanie plików

System został rozbudowany o funkcjonalność przechowywania i zarządzania przesłanymi przez użytkowników plikami, co ma na celu ułatwienie efektywnej komunikacji oraz wymiany informacji. Ten dodatek umożliwia centralne gromadzenie niezbędnych dokumentów w ramach jednego systemu. Decyzja o wprowadzeniu tej funkcjonalności wynikała z potrzeby dodawania różnych rodzajów dokumentów do aplikacji, takich jak potwierdzenia odbioru instalacji, instrukcje montażu czy zdjęcia zakończonych realizacji.

Podczas opracowywania rozwiązania rozważono dwie możliwości implementacji przechowywania plików: w bazie danych lub w strukturze plików projektu. Po przeprowadzeniu analizy, zdecydowano się ostatecznie na przechowywanie przesłanych plików w strukturze plików projektu. Głównym powodem tej decyzji była łatwość wyświetlania zdjęć profilowych użytkowników systemu. Dzięki temu podejściu, aplikacja webowa może bezpośrednio odwoływać się do każdego

zdjęcia za pomocą bezwzględnej ścieżki do pliku. W warstwie serwerowej został utworzony specjalny folder, który służy do przechowywania tego rodzaju dokumentów.

Celem zagwarantowania unikalności nazw plików, przy użyciu klasy BCryptPasswordEncoder z biblioteki Spring Security, tworzony jest hash liczby sekund od dnia 1 stycznia 1970 roku. Implementacja metody enkodującej tworząc hash dodaje do niego losową wartość tekstową. Wykorzystanie tego rozwiązania zapewnia także bezpieczeństwo utrudniając możliwość przewidzenia nazwy przechowywanych dokumentów. Następnie hash poddawany jest czyszczeniu ze znaków specjalnych, które nie powinny znajdować się w nazwie plików. Metoda ta gwarantuje jednoznaczność nazw i minimalizuje ryzyko wystąpienia sytuacji, w których dwie różne informacje są zapisywane pod tą samą nazwą. Implementacja funkcjonalności została przedstawiona w przykładzie kodu zamieszczonym na listingu 32. Listing 33 przedstawia wykorzystanie klasy BCryptPasswordEncoder z biblioteki Spring Security.

Listing 32 E-montażysta – Implementacja funkcjonalności zapisywanie do pliku. Źródło: Opracowanie własne.

```
@Override
public String save(byte[] content, String fileName) throws IOException
  String fileExtension = getFileExtension(fileName).orElseThrow();
  String uniqueName = getUniqueFileName();
  Path newFile = Paths.get(RESOURCES_DIR + uniqueName + "." +
fileExtension).normalize();
  Files.createDirectories(newFile.getParent());
  Files.write(newFile, content);
  return newFile.toAbsolutePath().toString();
}

//...


private String getUniqueFileName() {
  String currentDate = String.valueOf(System.currentTimeMillis());
  return
cleanSpecialCharacters(bCryptPasswordEncoder.encode(currentDate));
}

private Optional<String> getFileExtension(String filename) {
  return Optional.ofNullable(filename)
  .filter(f -> f.contains("."))
  .map(f -> f.substring(filename.lastIndexOf(".") + 1));
}

private String cleanSpecialCharacters(String fileName) {
    return fileName.replaceAll("[\\\\/:*?\"<>|]", "");
}
```

Listing 33 E-montażysta - Wykorzystanie klasy BCryptPasswordEncoder z biblioteki Spring Security. Źródło: ***Błąd! Nie można odnaleźć źródła odwołania.***

```
public String encode(CharSequence rawPassword) {
    if (rawPassword == null) {
    throw new IllegalArgumentException("rawPassword cannot be null");
    } else {
        String salt = this.getSalt();
        return BCrypt.hashpw(rawPassword.toString(), salt);
    }
```

```
}



private String getSalt() {
  return this.random != null ?
BCrypt.gensalt(this.version.getVersion(), this.strength, this.random) :
BCrypt.gensalt(this.version.getVersion(), this.strength);
}
```

5.5.6 Komunikacja z bazą danych

Do komunikacji z bazą danych w projekcie zostały użyte frameworki Spring Data oraz Java Persistence API (JPA), które zapewniają wygodne i wydajne sposoby interakcji z bazami danych.

Spring Data według **Błąd! Nie można odnaleźć źródła odwołania.** jest częścią większego n arzędzia jakim jest Spring Framework i ma na celu uproszczenie dostępu do bazy danych i zmniejszenie kodu szablonowego poprzez zapewnienie warstwy abstrakcji wysokiego poziomu. Obsługuje wiele technologii baz danych. Dzięki Spring Data programiści mogą definiować repozytoria i wykorzystywać predefiniowane metody do wykonywania typowych operacji na bazie danych bez konieczności pisania niestandardowych zapytań SQL lub kodu dostępu do danych.

JPA definiuje strukturę mapowania obiektowo-relacyjnego (ORM). Zapewnia standardowy sposób mapowania obiektów Java na tabele bazy danych i upraszcza interakcję między aplikacjami Java a relacyjnymi bazami danych. Hibernate jako implementacja JPA obsługuje podstawowe operacje na bazie danych i umożliwia pracę z obiektami Java zamiast bezpośredniego zajmowania się instrukcjami SQL. Listingi 34-36 przedstawiają kolejno przykład interfejsu JpaRepisitory, wykorzystanie adnotacji @Entity i @Id do mapowania klas i atrybutów oraz wykorzystanie adnotacji @OneToMany, @ManyToOne do mapowania asocjacji.

Listing 34 E-montażysta - Przykład interfejsu JpaRepisitory. Źródło: Opracowanie własne.

```
@Repository
public interface AppUserRepository extends JpaRepository<AppUser,Long>
{

    Optional<AppUser> findByIdAndDeletedIsFalse(Long id);
    AppUser findByUsername(String username);
    AppUser findByResetPasswordToken(String resetPasswordToken);

    List<AppUser> findAllByDeletedIsFalse();
    List<AppUser> findAllByRolesNotContaining(Role role);
    List<AppUser> findAllByIdIn(List<Long> listOfIds);
    List<AppUser> findAllByRolesContaining(Role role);

}
```

Listing 35 E-montażysta - Wykorzystanie adnotacji @Entity i @Id do mapowania klas i atrybutów. Źródło: Opracowanie własne.

```
@Entity
public class Tool {

//...
```

```
@Id
    private Long id;

    private String name;

    private LocalDateTime createdAt;

    private String code;

    private boolean deleted = Boolean.FALSE;

//...

}
```

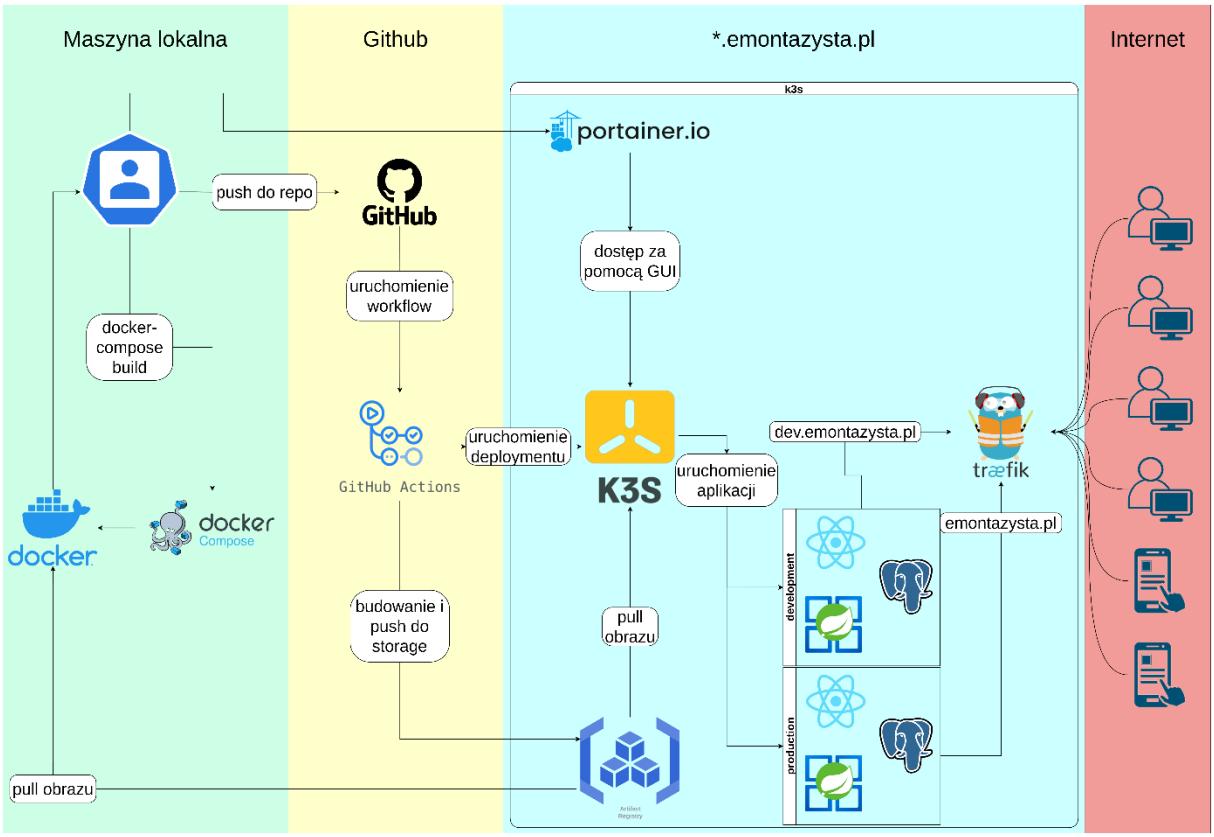
Listing 36 E-montażysta - Wykorzystanie adnotacji @OneToMany, @ManyToOne do mapowania asocjacji. Źródło: Opracowanie własne.

```
@Entity
public class Tool {

//...

@OneToMany(mappedBy = "tool")
    private List<ToolRelease> toolReleases;

    @ManyToOne
    private Warehouse warehouse;

    @OneToMany(mappedBy = "tool")
    private List<ToolEvent> toolEvents;

    @ManyToOne
    private ToolType toolType;
}
```

5.6. Implementacja części DevOps

Głównym celem sekcji odpowiedzialnej za narzędzia DevOps było dostarczenie kompletnego potoku (*pipeline'u*), który spełniałby podstawowe zasady Ciągłej Integracji (*CI*) i Ciągłego Dostarczania (CD). Miał on zapewnić udogodnienia na każdym etapie procesu tworzenia oprogramowania, czyli od rozwoju, przez integrację, testowanie, aż do wdrażania aplikacji. Jego implementacja miała na celu poprawę doświadczenia programisty oraz zwiększenie jakości dostarczanego kodu, jak również przyspieszenie procesu wdrażania aplikacji.

W celu jego implementacji wykorzystano różne narzędzia i technologie, takie jak Docker, Kubernetes (konkretnie k3s), Rejestr Docker, Traefik oraz Akcje Github. W tym rozdziale skupiono się na analizie wykorzystanych rozwiązań z podziałem na etapy. Rysunek 55, prezentujący graficznie potok, jest kluczowym elementem umożliwiającym zrozumienie architektury.



Rysunek 55 E-montażysta Przepływ zmian w projekcie Źródło: Opracowanie własne.

5.6.1 Lokalne środowisko developerskie

Aby zapewnić deweloperom jak najlepsze doświadczenia i umożliwić im skoncentrowanie się na pisaniu kodu, zastosowany został Docker w sekcjach aplikacji webowej oraz serwerowej. Dla każdej warstwy przygotowane został osobne pliki Dockerfile, które zawierały manifesty do budowania obrazów. Umożliwiło to członkom zespołu uruchamianie odizolowanych instancji poszczególnych warstw aplikacji na swoich środowiskach lokalnych. W manifestach zastało wykorzystane podejście wieloetapowe oraz wielowarstwowe konstruowania obrazów. Każdy plik Dockerfile został podzielony na etapy: *builder*, *dev-envs*, *prepare-production* oraz *production-build*. Poniżej przeanalizowany każdy ze wspomnianych etapów na podstawie załączonego kodu.

1. Etap *builder*:
    - Ustawienie katalogu roboczego: Na początku tego etapu, konfigurowany jest katalog, w którym Docker będzie operował podczas budowania obrazu aplikacji.
    - Skopiowanie potrzebnych plików projektu: Następnie, niezbędne pliki projektu, takie jak pliki źródłowe i narzędzia do zarządzania zależnościami, są kopiowane do katalogu roboczego.
    - Budowanie aplikacji: W tym etapie uruchamiany jest proces budowania aplikacji. Może to obejmować kompilację kodu, instalację zależności i inne niezbędne czynności. Wynikiem tego procesu jest obraz aplikacji zbudowany dla lokalnego środowiska deweloperskiego. Następny etap to dev-*envs*, który wprowadza obsługę nowej technologii od Docker Inc. o nazwie Dev Environments. Pozwala ona programistą tworzyć i zarządzać spójnymi, zdecentralizowanymi środowiskami deweloperskimi przechowywanymi w kontenerach. Jednakże, ze względu na dodatkową warstwę złożoności oraz wymagane kompetencje, programiści nie wykorzystywali tej funkcjonalności.

2. Etap *dev-envs*:
    - Obsługa Docker Dev Environments: Ten etap wprowadza obsługę narzędzia Docker Dev Environments, które pozwala na tworzenie i zarządzanie spójnymi środowiskami deweloperskimi przechowywanymi w kontenerach. Jednak w opisanym przypadku zespół DevOps zdecydował się nie wykorzystywać tej funkcjonalności ze względu na dodatkową złożoność i wymagane kompetencje.

3. Etap *prepare-production*:
    - Tworzenie odpowiednich katalogów: Na początku tego etapu tworzone są docelowe katalogi potrzebne do działania aplikacji w produkcji.
    - Rozpakowywanie pliku JAR: Plik JAR, który został wygenerowany w poprzednim etapie, jest rozpakowywany w celu uzyskania niezbędnych zasobów aplikacji.

4. Etap *production-build*:

    - Tworzenie specjalnego użytkownika: W tym etapie tworzony jest specjalny użytkownik, który będzie uruchamiał proces aplikacji w kontenerze. Zdecydowaliśmy się na to rozwiązanie głównie ze względu na zasadę minimalnych uprawnień.
    - Tworzenie struktury katalogów: Tworzona jest struktura katalogów, która będzie wykorzystywana do przechowywania załączników lub innych potrzebnych danych.
    - Kopiowanie odpowiednich katalogów: Katalogi "lib", "META-INF" i "classes" z etapu *prepare-production* są kopiowane oddzielnie przy użyciu komendy COPY.

    - Ustawienie portu i punktu wejścia: Na koniec ustawiany jest port, z którego aplikacja zostanie wystawiona w kontenerze, oraz ustalony zostaje punkt wejścia jako proces Javy z odpowiednimi parametrami wskazującymi na główną klasę aplikacji.

Struktura manifestu, przedstawiona na listingu 37, zapewniła separację między zależnościami a aplikacją, co pozwoliło nam optymalnie wykorzystać przechowywanie w pamięci podręcznej warstw w Dockerze i poskutkowało skróceniem czasu budowania obrazów oraz zmniejszeniem ich rozmiaru w rejestrze.

Listing 37 E-montażysta – Dockerfile. Źródło: Opracowanie własne.

```
#ETAP I
FROM maven:3.8.5-eclipse-temurin-17 as builder
WORKDIR /workspace/app
COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src
RUN mvn install --no-transfer-progress

# ETAP II
FROM builder as dev-envs
CMD ["mvn", "spring-boot:run"]

# ETAP III
FROM builder as prepare-production
RUN mkdir -p target/dependency
WORKDIR /workspace/app/target/dependency
```

```
RUN jar -xf ../*.jar

# ETAP IV
FROM eclipse-temurin:17-jre-alpine as production-build
RUN addgroup -S spring && adduser -S spring -G spring
RUN mkdir /files
RUN chown spring:spring /files
USER spring:spring
VOLUME /tmp /files
ARG DEPENDENCY=/workspace/app/target/dependency
COPY --from=prepare-production ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=prepare-production ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=prepare-production ${DEPENDENCY}/BOOT-INF/classes /app
EXPOSE 8080
ENTRYPOINT ["java","-
cp","app:app/lib/*","com.emontazysta.EMontazystaApplication"]
```

W ramach środowiska lokalnego zostało zapewniony także plik docker-compose, który umożliwiał łatwe budowanie oraz uruchomienie wymaganych przez aplikację komponentów, tj. frontendu, backendu, czy bazy danych Postgres. Listing 38 przedstawia zawartość pliku docker-compose.yaml.

Listing 38 E-montażysta – docker-compose.yaml. Źródło: Opracowanie własne.

```
services:
  backend:
    build:
      context: ../Java/E-montazysta
      target: dev-envs
    restart: always
    secrets:
      - db-password
    ports:
      - 8080:8080
    environment:
      DB_URL: jdbc:postgresql://db:5432/emontazystadb
      DB_USER: postgres
      DB_PASSWORD: password
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - react-spring
      - spring-postgres
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres
    restart: always
    secrets:
      - db-password
    ports:
      - 5432:5432
    volumes:
      - db-data:/var/lib/postgresql/data
```

```
    networks:
      - spring-postgres
      - pgadmin-postgres
    environment:
      - POSTGRES_DB=emontazystadb
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres", "-d",
"emontazystadb"]
      interval: 3s
      timeout: 5s
      retries: 10
      start_period: 3s
    expose:
      - 5432

  frontend:
    build:
      context: ../React
      target: dev-envs
    ports:
      - 3000:3000
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - react-spring
    depends_on:
      - backend
    expose:
      - 3306
      - 33060

  db-web:
    image: dpage/pgadmin4
    secrets:
      - db-connection
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@emontazysta.pl
      - PGADMIN_DEFAULT_PASSWORD=password
      - PGADMIN_CONFIG_MAX_LOGIN_ATTEMPTS=0
    ports:
      - 5050:80
    networks:
      - pgadmin-postgres
    volumes:
      - db-web-data:/var/lib/pgadmin
      - ../db/connection.json:/pgadmin4/servers.json
    depends_on:
      db:
        condition: service_healthy

volumes:
  db-data: {}
  db-web-data: {}
secrets:
  db-password:
```

```
    file: ../db/password.txt
  db-connection:
    file: ../db/connection.json
networks:
  react-spring: {}
  spring-postgres: {}
  pgadmin-postgres: {}
  backend:
    build:
      context: ../Java/E-montazysta
      target: dev-envs
    restart: always
    secrets:
      - db-password
    ports:
      - 8080:8080
    environment:
      DB_URL: jdbc:postgresql://db:5432/emontazystadb
      DB_USER: postgres
      DB_PASSWORD: password
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - react-spring
      - spring-postgres
    depends_on:
      db:
        condition: service_healthy
  db:
    image: postgres
    restart: always
    secrets:
      - db-password
    ports:
      - 5432:5432
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - spring-postgres
      - pgadmin-postgres
    environment:
      - POSTGRES_DB=emontazystadb
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres", "-d",
"emontazystadb"]
      interval: 3s
      timeout: 5s
      retries: 10
      start_period: 3s
    expose:
      - 5432
  frontend:
    build:
      context: ../React
      target: dev-envs
```

```
    ports:
      - 3000:3000
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - react-spring
    depends_on:
      - backend
    expose:
      - 3306
      - 33060
  db-web:
    image: dpage/pgadmin4
    secrets:
      - db-connection
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@emontazysta.pl
      - PGADMIN_DEFAULT_PASSWORD=password
      - PGADMIN_CONFIG_MAX_LOGIN_ATTEMPTS=0
    ports:
      - 5050:80
    networks:
      - pgadmin-postgres
    volumes:
      - db-web-data:/var/lib/pgadmin
      - ../db/connection.json:/pgadmin4/servers.json
    depends_on:
      db:
        condition: service_healthy
volumes:
  db-data: {}
  db-web-data: {}
secrets:
  db-password:
    file: ../db/password.txt
  db-connection:
    file: ../db/connection.json
networks:
  react-spring: {}
  spring-postgres: {}
  pgadmin-postgres: {}
```

5.6.2 System kontroli wersji

W zakresie systemu kontroli wersji zespół zdecydował się skorzystać z narzędzi dostarczonych przez platformę GitHub, a konkretnie z narzędzia GitHub Actions. W celu zapewnienia spójności i jakości w procesie budowania aplikacji opracowano dedykowany przepływ pracy dla każdej warstwy aplikacji. Te sekwencje operacji przeprowadzają weryfikację poprawności budowy oraz wykonują testy jednostkowe aplikacji. Przepływy pracy są wyzwalane w odpowiedzi na określone zdarzenia w repozytorium, takie jak utworzenie *pull request* lub scalenie z gałęzią master, oraz tworzenie tagów. W ramach tych przepływów pracy wykorzystywane są wcześniej wspomniane pliki Dockerfile w celu zbudowania obrazu aplikacji i przesłania go do rejestru Docker, który jest używany przez nasz orkiestrator. Listing 39 przedstawia przykładowy kod reprezentujący te przepływy pracy.

Listing 39 E-montażysta - Przykładowy kod reprezentujący przepływ pracy. Źródło: Opracowanie własne.

```
name: Release&Publish Java Dev

on:
  push:
    branches:
      - 'master'
    paths:
      - 'Java/**'
  pull_request:
    branches:
      - 'master'
    paths:
    - 'Java/**'
  workflow_dispatch:
jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        uses: actions/checkout@v2
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          registry: ${{ secrets.DOCKER_REGISTRY }}
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      -
        name: Build Docker image
        uses: docker/build-push-action@v2

        with:
          file: Java/E-montazysta/Dockerfile
          context: Java/E-montazysta
          tags: |
            ${{ secrets.DOCKER_REGISTRY }}/emontazysta/backend:latest
            ${{ secrets.DOCKER_REGISTRY }}/emontazysta/backend:${{
github.sha }}
          push: true
          cache-from: type=gha
          cache-to: type=gha,mode=max

  publish:
    needs: [release]
    runs-on: ubuntu-latest
    steps:
      -
        name: Kubernetes Set Context
        uses: Azure/k8s-set-context@v3.0
        with:
```

```
          cluster-type: generic
          method: service-account
          k8s-url: ${{ secrets.KUBERNETES_URL }}
          k8s-secret: ${{ secrets.KUBERNETES_SECRET_DEV }}
      -
        name: Checkout source code
        uses: actions/checkout@v3
      -
        name: Deploy to the Kubernetes cluster
        uses: azure/k8s-deploy@v4.9
        with:
          namespace: development
          manifests: |
            Java/E-montazysta/.build/backend-pvc.yaml
            Java/E-montazysta/.build/deployment.yml
            Java/E-montazysta/.build/ingressroute_dev.yml
            Java/E-montazysta/.build/middleware.yml
            Java/E-montazysta/.build/serverstransport.yml
            Java/E-montazysta/.build/service.yml
            Java/E-montazysta/.build/postgres-deployment.yaml
            Java/E-montazysta/.build/postgres-service.yaml
            Java/E-montazysta/.build/postgres-pvc.yaml
          images: |
            ${{ secrets.DOCKER_REGISTRY }}/emontazysta/backend:${{
github.sha }}
          imagepullsecrets: |
            regcred
          strategy: basic
          action: deploy
          token: ${{ github.token }}
```

Jak widać, w ramach przepływu pracy stosowane jest budowanie za pomocą BuildKit. Następnie, przepływ pracy autoryzuje się w rejestrze i inicjuje budowanie aplikacji. Artefaktem wynikowym tej operacji jest przesłany do rejestru obraz. W kolejnym kroku przepływ pracy autoryzuje się w naszym klastrze k3s i wdraża serwerową aplikację na przestrzeni nazw o nazwie *development*. Do tego celu wykorzystywane są pliki znajdujące się w katalogu .build, które zawierają konfigurację kolejno kubernetesowego *persistent volume claim, deployment'u, service'u* oraz traefikowego *ingress routu, middleware'u* i *server* transportu. Wszelkie dane wrażliwe przechowujemy w ramach mechanizmu secretów repozytorium również zapewnianego przez Github.

Aby zapewnić ciągłą gotowość do wdrożenia najnowszej działającej wersji na środowisko produkcyjne, zespół zdecydował się zrezygnować z modelu *gitflow* na rzecz *trunk-based development*. Ta strategia znacznie upraszcza proces budowania potoku (pipeline'u). Po przesłaniu zmian na gałąź *feature*, członek zespołu oczekuje, aż przepływ pracy zweryfikuje poprawność budowania i przeprowadzi testy jednostkowe. Następnie, po uzyskaniu co najmniej jednej zgody od innego członka zespołu oraz poprawnej weryfikacji od narzędzi automatycznych, wprowadza swoje zmiany do gałęzi *master*. Po scaleniu aplikacja jest budowana i wdrażana na środowisko *development*, gdzie jest testowana przez zespół testowy. W przypadku wydania nowej wersji aplikacji stosowane są gitowe tagi w formacie <nazwa warstwy>/v<numer wersji>. Numeracja wersji jest prowadzona zgodnie z zasadami semantycznego wersjonowania. W przypadku wykrycia błędów stosowana jest strategia *fix forward*.

Zapewnienie skutecznego systemu kontroli wersji i procesu CI/CD jest kluczowe dla utrzymania wysokiej jakości i ciągłego rozwoju aplikacji. Wybranie narzędzi takich jak GitHub Actions i strategii *trunk-based development* pozwalało zespołowi programistów skoncentrować się na efektywnym pisaniu kodu, jednocześnie zapewniając automatyzację procesu budowania, testowania i wdrażania aplikacji.

5.6.3 Środowiska zdalne

Zaprojektowane zostało również zdalne środowisko integracyjne i produkcyjne dla projektu. mplementacja tych środowisk opiera się na wykorzystaniu przestrzeni nazw w jednowęzłowym klastrze k3s, który został uruchomiony na serwerze VPS. Przy podejmowaniu decyzji rozważaliśmy różne technologie, takie jak Docker Swarm, jednak doszliśmy do wniosku, że nie spełnia on naszych wymagań dotyczących izolacji sieciowej środowisk, co przełożyłoby się na niepotrzebne skomplikowanie naszego procesu CI/CD. Rozważaliśmy także Kubernetes, Microk8s i Nomad, ale ostatecznie zdecydowaliśmy się na k3s ze względu na prostotę oraz fakt, że spełniał wszystkie nasze potrzeby związane z projektem.

Aby rozwiązać kwestie ruchu sieciowego zarówno do, jak i wewnątrz klastra, zdecydowaliśmy się użyć narzędzia Traefik. Wybraliśmy je, ponieważ jest to proste i natywne rozwiązanie dla orkiestratorów. Dzięki niemu zaimplementowaliśmy mechanizmy takie jak przekierowanie ruchu z subdomen na usługi w odpowiednich przestrzeniach nazw, a także automatyczne zarządzanie certyfikatami domeny za pomocą protokołu ACME. Na listingu 40 można zobaczyć przykładową konfigurację IngressRoute dla warstwy serwerowej na środowisku *development*.

Listing 40 E-montażysta - konfiguracja IngressRoute dla warstwy serwerowej. Źródło: Opracowanie własne

```
---
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: backend
spec:
  entryPoints:
    - web
    - websecure
  routes:
  - kind: Rule
    match: Host(`dev.emontazysta.pl`) && PathPrefix(`/api/v1`,
`/swagger-ui`, `/v3/api-docs`)
    middlewares:
    - name: backend-add-index-swagger
      namespace: development
    priority: 2
    services:
    - kind: Service
      name: backend
      namespace: development
      port: 80
```

Na orkiestratorze k3s umieszczona została również instancja Docker Registry, która umożliwiała łatwe zarządzanie obrazami aplikacji, oraz instancję Portainera, który wystawiał intuicyjny interfejs graficzny dostępny przez przeglądarkę.

Aby przekazywać parametry połączenia do zewnętrznych komponentów, takich jak baza danych czy serwer SMTP, wykorzystano funkcjonalność przechowywania poufnych danych dostępną w Kubernetes. Następnie wstrzykiwano te informacje do kontenerów poprzez ustawienie odpowiednich zmiennych lokalnych. Listing 41 przedstawia manifest dla tego rodzaju przechowywania.

Listing 41 E-montażysta –Przykład przechowywania poufnych danych. Źródło: Opracowanie własne

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: db-credentials
type: Opaque
data:
  username: dXN1cm5hbWU=
  password: cGFzc3dvcmQ=
```

Przestawiona metodyka pozwoliła bezpiecznie przechowywać poufne informacje, jednocześnie umożliwiając dostęp do nich wewnątrz kontenerów aplikacji.

Zapewnienie zdalnych środowisk integracyjnych i produkcyjnych oraz odpowiednie zarządzanie ruchem sieciowym i poufnymi danymi były kluczowe dla skutecznego wdrożenia i funkcjonowania naszego potoku CI/CD. Wybór k3s i Traefik umożliwił nam osiągnięcie prostego i efektywnego rozwiązania tych aspektów.

5.6.4 Podsumowanie

Wdrożenie ciągłej integracji i dostarczania (CI/CD) ułatwiło wyłapywanie błędów oraz możliwość ich rozwiązywania na jak najwcześniejszym etapie. Dzięki automatycznym testom jednostkowym i weryfikacji poprawności budowania, byliśmy w stanie szybko zidentyfikować potencjalne problemy i przeciwdziałać im jeszcze przed wdrożeniem aplikacji.

Implementacja środowisk zdalnych przy użyciu k3s pozwoliła nam zapewnić ich stabilne działanie. Dzięki izolacji i odpowiedniemu zarządzaniu zasobami, nasze środowiska były niezależne od siebie, co minimalizowało ryzyko wpływu jednego środowiska na inne.

W trakcie trwania projektu zastało przeprowadzone ponad 200 zdalnych procesów budowania i 100 wdrożeń, które odbywały się niemal transparentnie dla programistów. Dzięki zautomatyzowanym procesom i integracji z narzędziami kontroli wersji, procedura budowania i wdrażania była szybka i bezproblemowa. Implementacja CI/CD umożliwiła nam również separację środowisk i testowanie na nich różnych funkcjonalności. Dzięki temu mogliśmy bezpiecznie eksperymentować z nowymi funkcjami i testować je na wydzielonych środowiskach, unikając potencjalnych konfliktów czy błędów w działającym kodzie.

Podsumowując, wdrożenie CI/CD przyczyniło się do wykrywania błędów na wczesnych etapach, zapewniło stabilne działanie zdalnych środowisk, umożliwiło liczne budowanie i wdrażanie bez przeszkód oraz pozwoliło na separację środowisk i testowanie na nich różnych funkcjonalności. Był to kluczowy element naszego projektu, który przyczynił się do osiągnięcia wysokiej jakości i efektywności procesu dostarczania oprogramowania.

Projekt stanowi przykład skutecznego wykorzystania narzędzi DevOps, które przyczyniły się do zwiększenia wydajności, jakości i bezpieczeństwa naszej aplikacji.

6. Interfejs użytkownika

W tym rozdziale skupiono się na opisie implementacji interfejsu użytkownika dla aplikacji webowych i mobilnych. Stanowi on powiązanie między użytkownikiem a aplikacją, wpływając bezpośrednio na ich satysfakcję oraz zachęcenie potencjalnych klientów. Zarówno aplikacja webowe, jak i mobilna odgrywa kluczową rolę w projekcie, przez co niezwykle istotne było zapewnienie intuicyjnego i estetycznego interfejsu. Szczególną uwagę zwrócono na klarowność, czytelność i łatwość obsługi aplikacji. Dlatego tak ważne było, aby aplikacja spełniała szereg kluczowych cechy:

1. Odpowiednia kolorystyka – Kolory zostały starannie dobrane tak żeby pasowały do kontekstu aplikacji oraz były przyjemna dla oka użytkownika. Dzięki temu, interfejs staje się przyjazny w użyciu, co przekłada się na lepsze wrażenia użytkownika.

2. Intuicyjna w obsłudze – Proste menu główne zostało zaprojektowane tak żeby zminimalizować wysiłek poznawczy wymagany od użytkowników. Poprzez logiczne rozmieszczenie opcji, można łatwo nawigować po aplikacji.

3. Spójna - Wszystkie elementy interfejsu zostały zaprojektowane z dbałością o jednolitość styl. Użyto powtarzalnych wzorców, takich jak typografia, układ elementów czy ikony, aby stworzyć spójny wygląd i wrażenie całości. Dzięki temu użytkownikowi przyjemniej jest korzystać z aplikacji, ponieważ wszystko działa w sposób przewidywalny.

4. Responsywna i skalowalna - Interfejs aplikacji został zaprojektowany z myślą o różnych urządzeniach i rozmiarach ich ekranów. Dzięki temu aplikacja może być wygodnie używana na komputerach, tabletach czy smartfonach, a elementy interfejsu dostosowują się elastycznie do ich rozmiarów. Skalowalność interfejsu umożliwia optymalne wykorzystanie przestrzeni ekranu, zapewniając użytkownikom ciągłą czytelność obsługi bez względu na to z jakiego urządzenia korzystają.

6.1. Aplikacja webowa

Aplikacja webowa zapewnia dostęp do funkcjonalności poprzez przeglądarkę internetową. Użytkownicy mogą łatwo korzystać z aplikacji na różnych urządzeniach, takich jak komputery, tablety czy smartfony, o ile posiadają dostęp do połączenia internetowego.

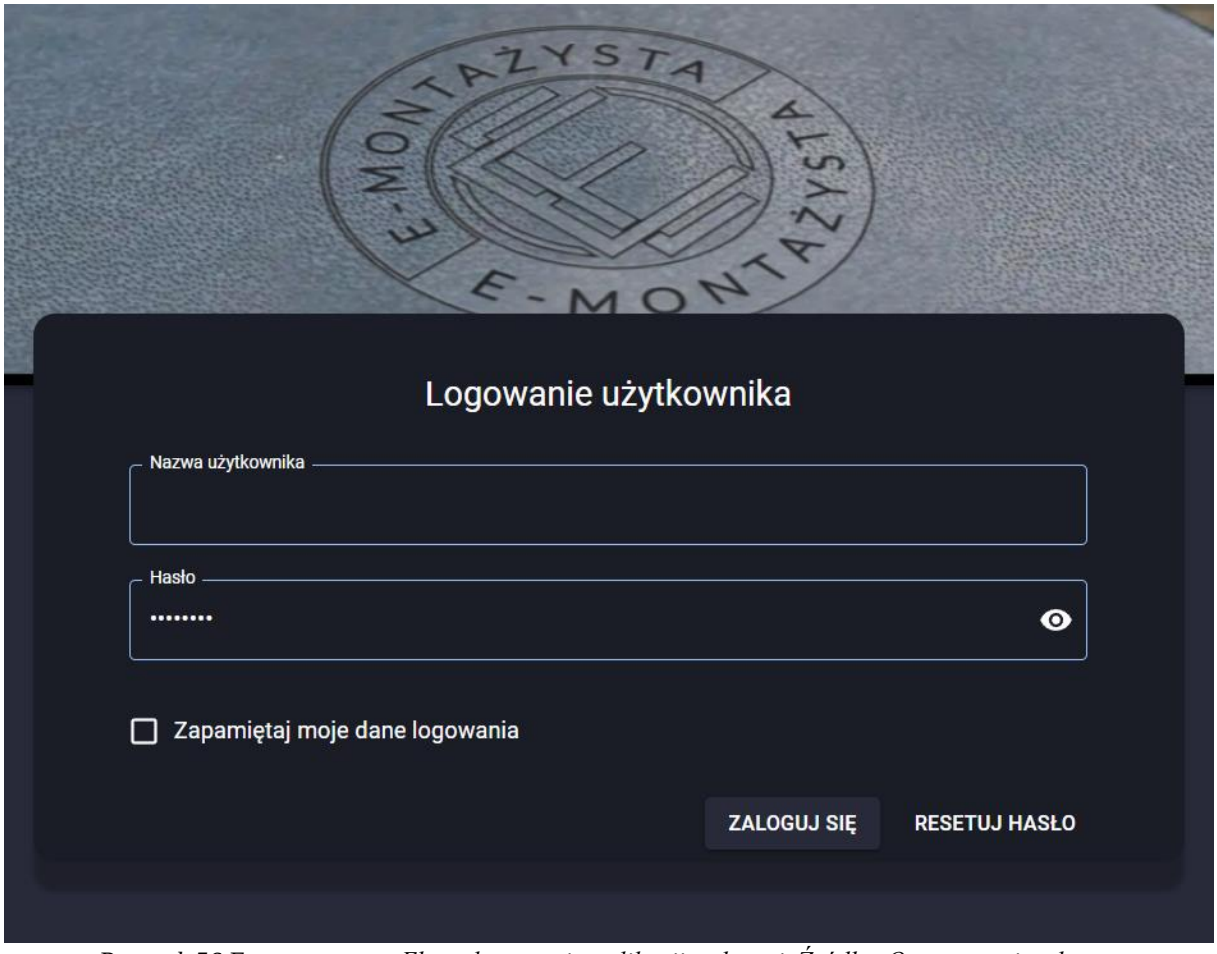
W następnych podrozdziałach skupiono się na przedstawieniu różnych aspektów interfejsu użytkownika dla aplikacji webowej, uwzględniając specyfikę systemów do zarządzania firmą monterską.

6.1.1 Ekran logowania

Aby uzyskać dostęp do aplikacji, użytkownik musi zalogować się za pomocą swojej nazwy użytkownika oraz hasła. Rysunek 56 przedstawia wymienione pola oraz dostępne opcje dodatkowe:

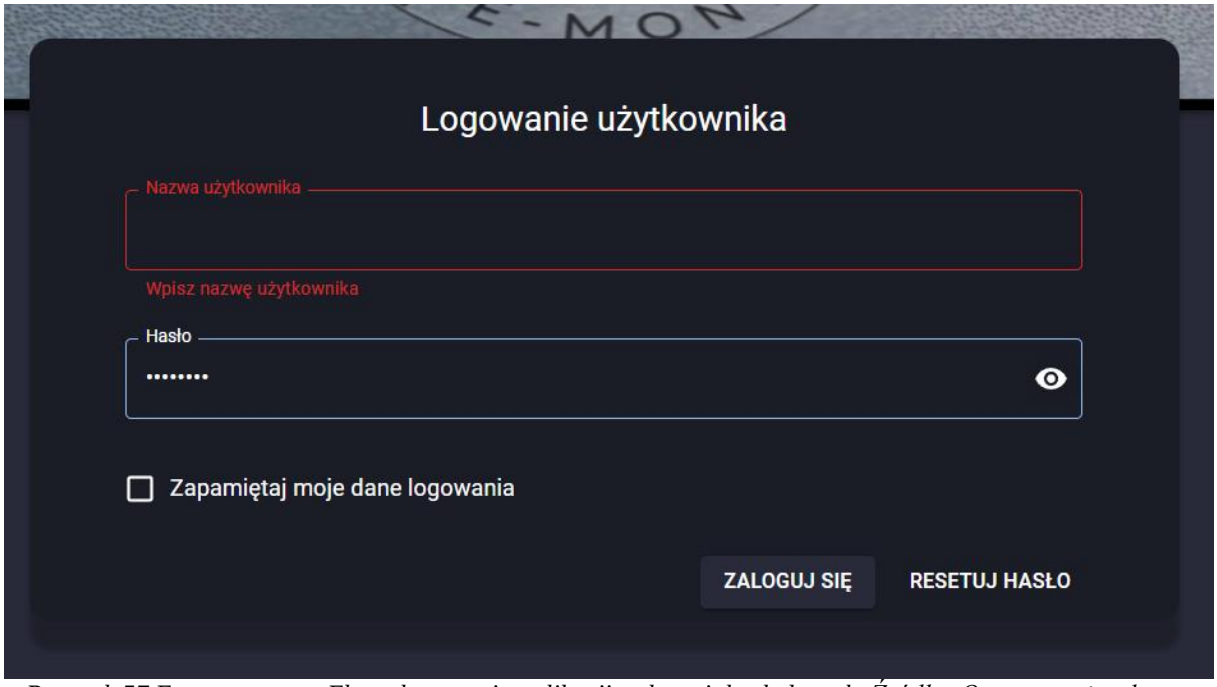
1. Nazwa użytkownika – Pole wymagane, jest unikalnym identyfikatorem użytkownika w systemie.
2. Hasło – Pole wymagane, jest poufnym kodem dostępu, który powinien być znany tylko użytkownikowi.
3. Zapamiętaj moje dane logowania – Opcja ta pozwala użytkownikowi zapisać nazwę użytkownika i hasło na urządzeniu, tak aby w przyszłości nie musiał ich wpisywać ponownie przy kolejnym logowaniu.

4. Resetuj hasło - W celu odzyskania dostępu w przypadku zapomnienia hasła, użytkownik ma możliwość skorzystania z opcji resetu hasła.



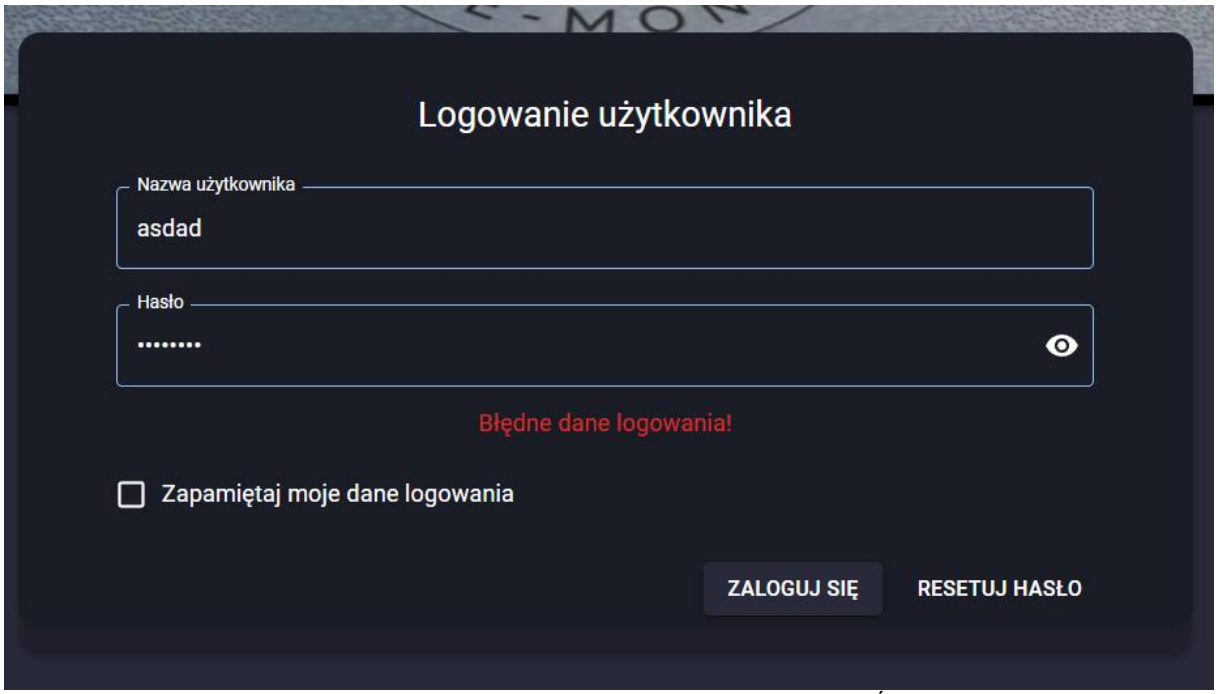
Rysunek 56 E-montażysta – Ekran logowania aplikacji webowej. Źródło: Opracowanie własne.

W przypadku niewprowadzenia któregoś z wymaganych pól, użytkownik zobaczy komunikat przedstawiony na rysunku 57.



Rysunek 57 E-montażysta – Ekran logowania aplikacji webowej, brak danych. Źródło: Opracowanie własne.

W przypadku podania nieprawidłowych danych logowania pojawi się generyczna informacja przedstawiona na rysunku 58.



Rysunek 58 E-montażysta – Ekran logowania aplikacji webowej, błędne dane. Źródło: Opracowanie własne.

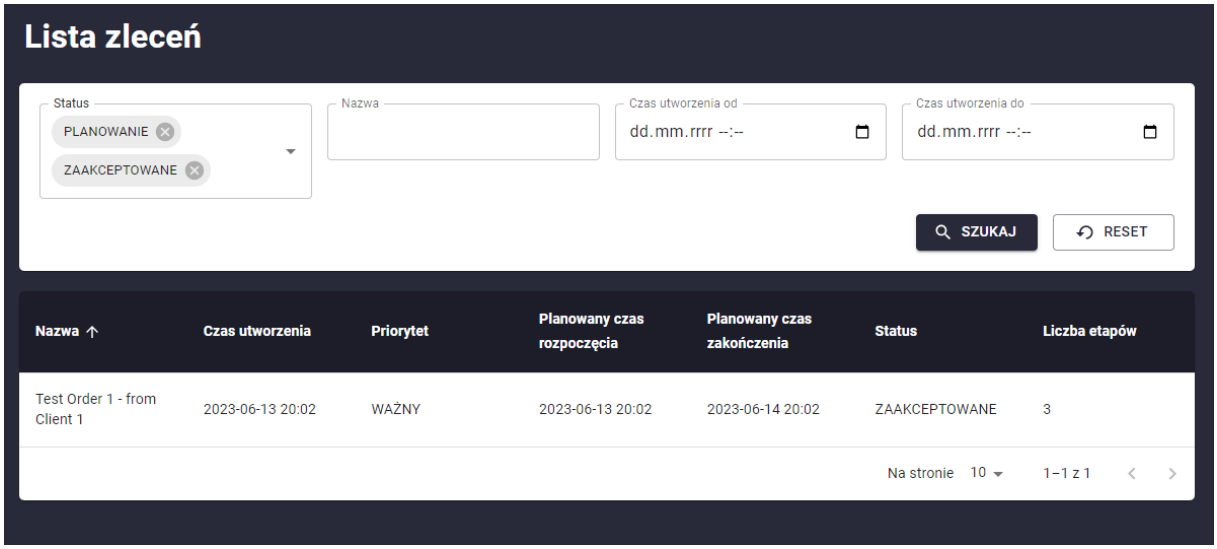
Po poprawnym podaniu danych zwrócony zostanie token na podstawie, którego użytkownik autoryzowany jest w aplikacji. Token zostaje zapisany w localstorage, a użytkownik przekierowany zostanie na odpowiedni dla niego homepage.

6.1.2 Widok Tabeli

W celu przedstawienia danych w czytelnej formie, wykorzystany został komponent FatTable. Funkcjonalności, które są dostępne w tym komponencie to:

- Wyświetlanie ikonki ładowania podczas pobierania danych.
- Tworzenie komponentu filtrowania, którego implementacja znajduje się po stronie backendu.
- Tworzenie komponentu tabeli, który wyświetla dane w czytelnej formie.
- Paginacja danych, której implementacja odbywa się po stronie frontendu.
- Sortowanie kolumn w tabeli, które również jest realizowane po stronie frontendu.
- Możliwość nawigacji do szczegółowych informacji po kliknięciu na wybrany wiersz.
- Wyświetlanie nazwy tabeli.

Rysunek 59 przedstawia przykładowy widok tabeli.



Rysunek 59 E-montażysta - Przykładowy widok tabeli. Źródło: Opracowanie własne.

Listing 42 przedstawia Komponent FatTable.

Listing 42 E-montażysta - Komponent FatTable. Źródło: Opracowanie własne.

```
type FatTableParams<T> = {

      query: UseQueryResult<T[], AxiosError>

      filterProps?: Filter

      headCells: Array<HeadCell<T>>

      initOrderBy: keyof T

      initOrderByDesc?: boolean

      onClickRow: (event: React.MouseEvent, row: T) => void
```

```
      pageHeader?: string

      idPropName: keyof T

}
      query: UseQueryResult<T[], AxiosError>
      filterProps?: Filter
      headCells: Array<HeadCell<T>>
      initOrderBy: keyof T
      initOrderByDesc?: boolean
      onClickRow: (event: React.MouseEvent, row: T) => void
      pageHeader?: string
      idPropName: keyof T
}
```

UseQueryResult<T[], AxiosError> to wynik metody useQuery z biblioteki react-query. Dostarcza informacje o statusie zapytania HTTP oraz zwraca tablicę obiektów danego typu lub błąd typu AxiosError.

Filter to parametr komponentu TableFilter, który zawiera dane niezbędne do generowania i funkcjonowania filtru w tabeli. Wykorzystuje generyczny komponent FormStructure.

Array<HeadCell<T>> to lista kolumn wyświetlanych w tabeli. Zawiera informacje niezbędne do generowania kolumn, funkcjonowania sortowania oraz formatowania komórek w tabeli.

6.1.3 Widok Formularza

W celu prezentacji formularzy, wykorzystano komponent FormStructure. Funkcjonalności, które są dostępne w tym komponencie to:

- Tworzenie struktury obiektu.
- Zarządzanie stanem obiektu, wyświetlanie oraz edycja danych.
- Zarządzanie dostępem do pól danych w zależności od roli użytkownika oraz trybu pracy strony (wyświetlanie, edycja lub tworzenie nowego obiektu).
- Walidacja danych.

Rysunek 60 przykładowy widok formularza.

Rysunek 60 E-montażysta - Przykładowy widok formularza. Źródło: Opracowanie własne

Listing 43 przedstawia przykład widoku formularza.

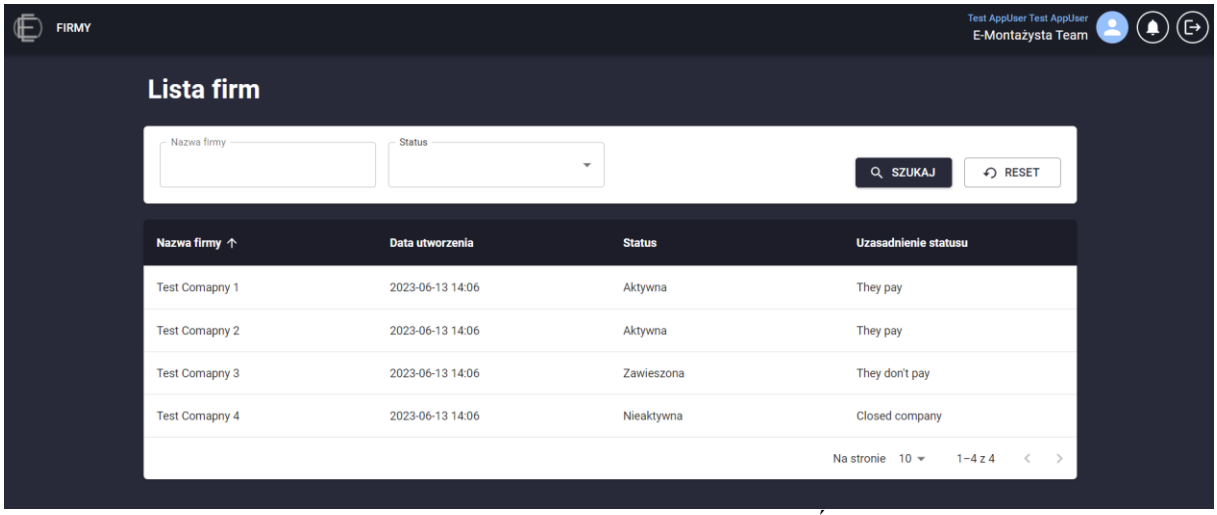
Listing 43 E-montażysta - Komponent FormStructure. Źródło: Opracowanie własne.

```
export type FormStructureParams = {

      formStructure: Array<FormInputProps>

      formik: any

      pageMode?: PageMode

      thin?: boolean

}
      formStructure: Array<FormInputProps>
      formik: any
      pageMode?: PageMode
      thin?: boolean
}
```

6.1.4 Interfejs Administratora chmury

W systemie E-montażysta, kluczową rolę pełni interfejs Administratora chmury. To w tym miejscu dodani zostaną potencjalni klienci systemu. Bez tego, niemożliwe jest otwarcie jakiekolwiek innej części systemu. Wszystkie pozostałe widoki i funkcjonalności budowane są wokół danych firm.

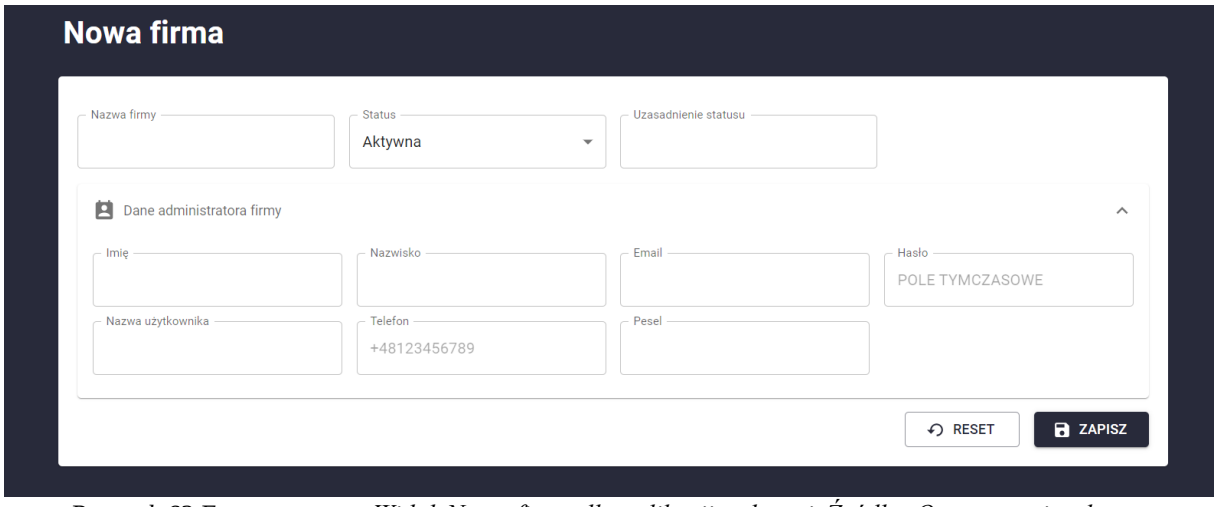
W interfejsie Administratora chmury, istnieje funkcjonalność umożliwiająca wyświetlenie listy dodanych firm. Na rysunku 61 przedstawiono przykład tego widoku.



Rysunek 61 E-montażysta – Widok listy firm dla aplikacji webowej. Źródło: Opracowanie własne.

W celu ułatwienia wyszukiwania, interfejs udostępnia filtry umożliwiające precyzyjnie określić kryteria i wyświetlić tylko te firmy, które je spełniają. Każda pozycja na liście zawiera podstawowe informacje o firmie, takie jak nazwa, data utworzenia status i uzasadnienie statusu. Klikając na jedną z pozycji z listy, Administrator ma możliwość edycji istniejących informacji. Może również usunąć firmę.

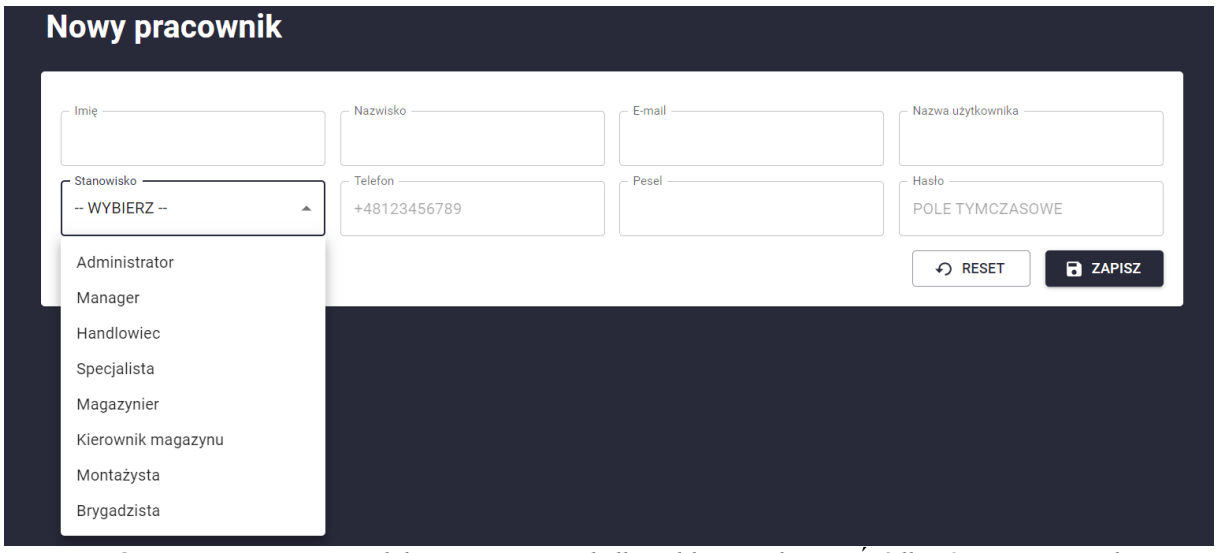
Rysunek 62 przedstawia widok Nowa firma. Umożliwia on dodanie nowej firmy do systemu. Aby to zrobić, konieczne jest podanie danych Administratora firmy, który będzie odpowiedzialny za dodanie pozostałych użytkowników.



Rysunek 62 E-montażysta – Widok Nowa firma dla aplikacji webowej. Źródło: Opracowanie własne.

6.1.5 Interfejs Administratora firmy

W ramach roli Administratora firmy, użytkownik ma możliwość zarządzania pracownikami w firmie poprzez dedykowany interfejs. Odpowiedzialny jest między innymi za dodawanie nowych użytkowników oraz przypisywanie im odpowiednich ról w systemie. Rysunek 63 przedstawia Widok Nowy pracownik.

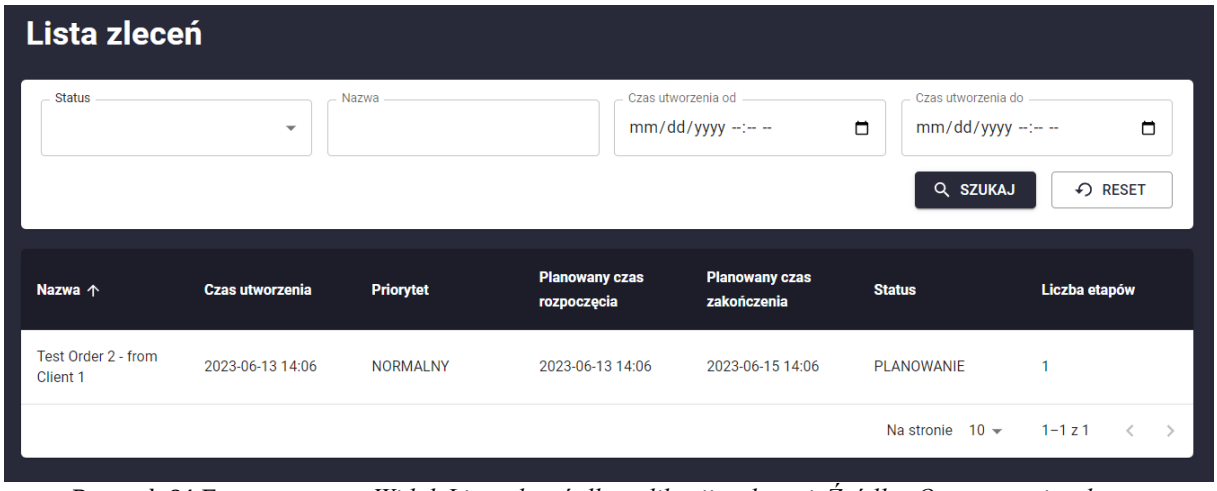


Rysunek 63 E-montażysta – Widok Nowy pracownik dla aplikacji webowej. Źródło: Opracowanie własne.

Widok Nowy pracownik firmy umożliwia dodawanie nowych użytkowników poprzez wprowadzanie ich danych, takich jak imię, nazwisko, adres e-mail oraz preferowane hasło. Administrator ma również możliwość przypisania odpowiednich ról dla każdego użytkownika w firmie.

6.1.6 Widok Zlecenia

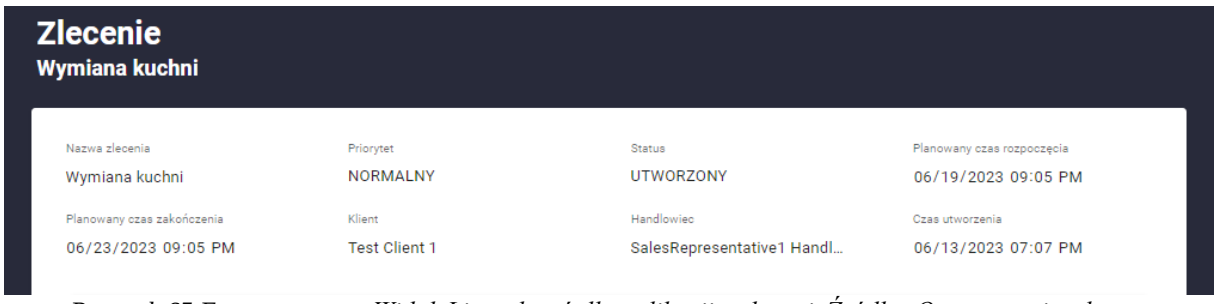
W systemie, w zależności od roli, użytkownik może mieć możliwość wyświetlenia widoku listy zleceń jakie ma do wykonania jego firma. Zawiera on istotne informacje, takie jak: nazwa zlecenia, czas utworzenia, priorytet, planowany czas rozpoczęcia, planowany czas zakończenia, status oraz liczbę etapów. Do dyspozycji jest również panel do filtrowania listy, w którym użytkownik ma możliwość wprowadzenia kryteriów takich jak: status, nazwa, czas utworzenia od i czas utworzenia do. Widok pozwala również na wybranie, ile zleceń wyświetlać na danej stronie. Po kliknięciu w dane zlecenie zostajemy przeniesieni do widoku szczegółów zlecenia. Rysunek 64 przedstawia widok Listy zleceń.



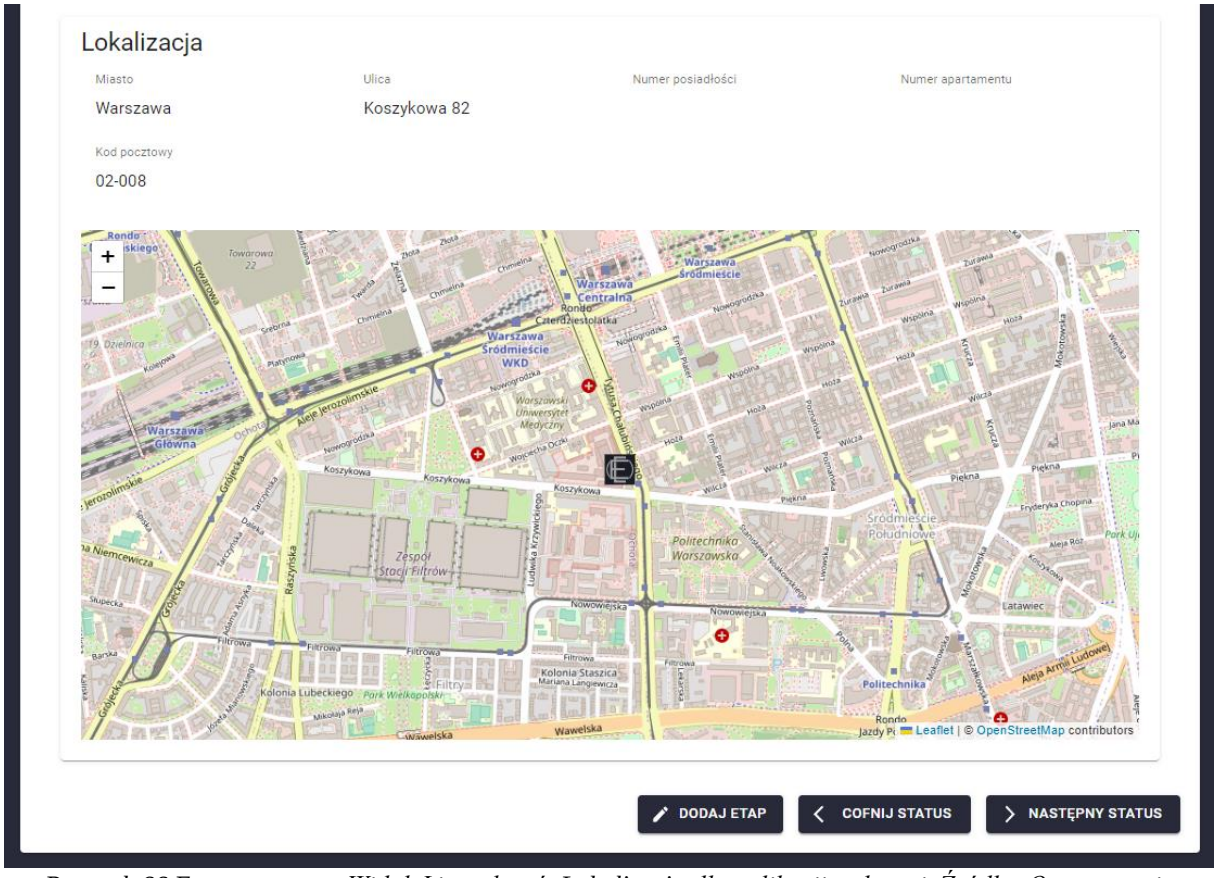
Rysunek 64 E-montażysta – Widok Lista zleceń dla aplikacji webowej. Źródło: Opracowanie własne.

1. Szczegóły zlecenia

Widok przedstawia szczegóły związane z konkretnym zleceniem. Rysunek 65 prezentuje wszystkie informacje, które zostały wymienione w liście zleceń, a dodatkowo zawiera dane dotyczące Klienta, dla którego wykonane będzie zlecenie, oraz Handlowca, który zarejestrował to zlecenie w systemie. Dostarczane są również informacje o lokalizacji, która szczegółowo precyzuje miejsce, w którym zlecenie będzie realizowane co przedstawia rysunek 66. Ponadto każde zlecenie zawiera listę etapów, które opisują jakie prace, kiedy i przy użyciu jakich narzędzi i elementów będą wykonywane, co przedstawia rysunek 67.



Rysunek 65 E-montażysta – Widok Lista zleceń dla aplikacji webowej. Źródło: Opracowanie własne.

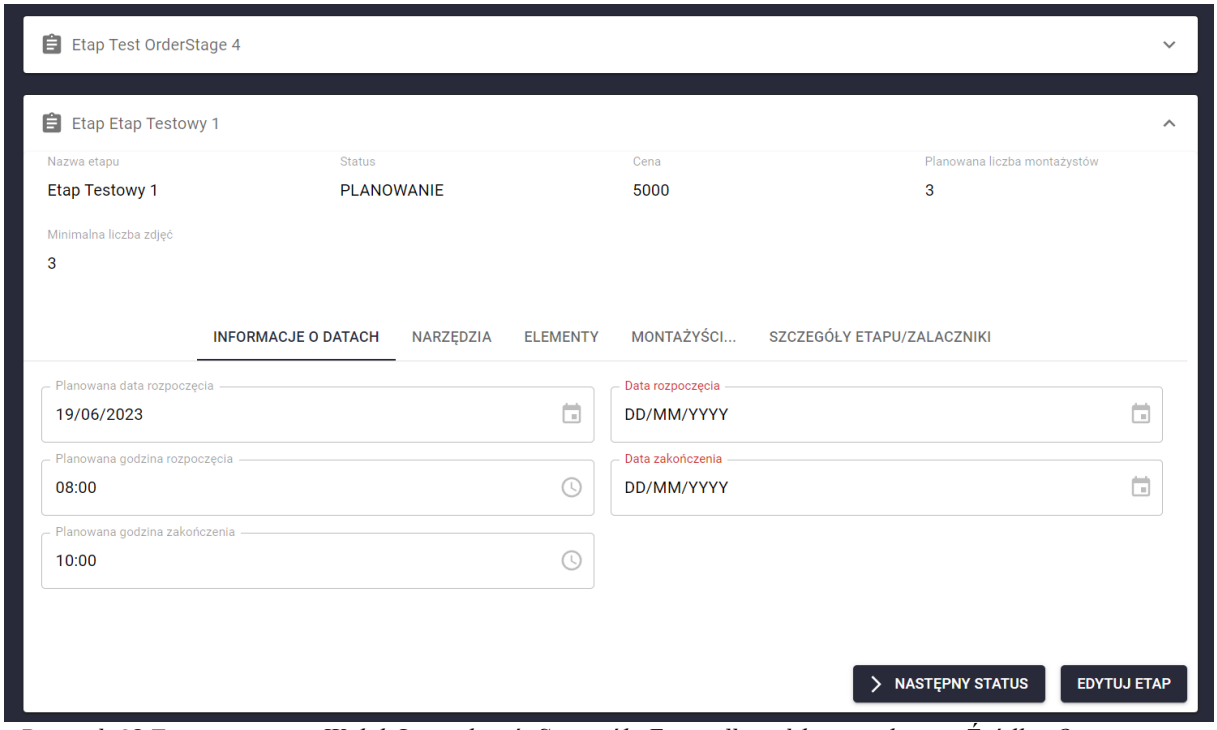


Rysunek 66 E-montażysta – Widok Lista zleceń, Lokalizacja dla aplikacji webowej. Źródło: Opracowanie własne.

Rysunek 67 E-montażysta – Widok Lista zleceń, Etap dla aplikacji webowej. Źródło: Opracowanie własne.

2. Szczegóły etapu

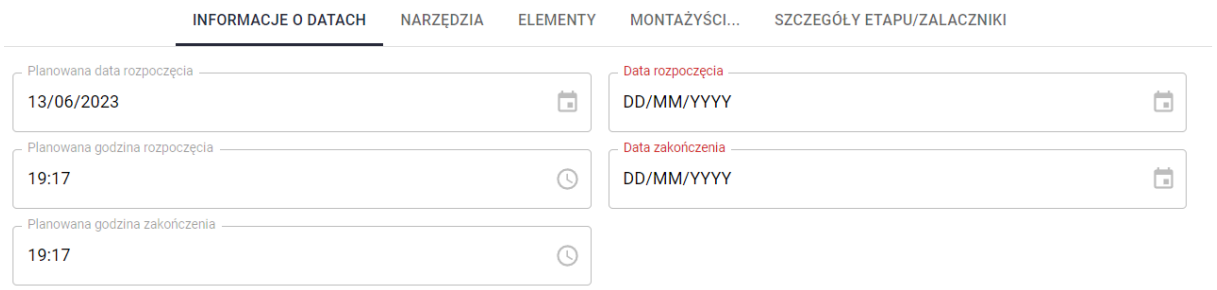
Każdy etap zbudowany jest z rozwijanego kafelka i zawiera szczegółowe informacje dotyczące jego wykonania – Rysunek 68.



Rysunek 68 E-montażysta – Widok Lista zleceń, Szczegóły Etapu dla aplikacji webowej. Źródło: Opracowanie własne.

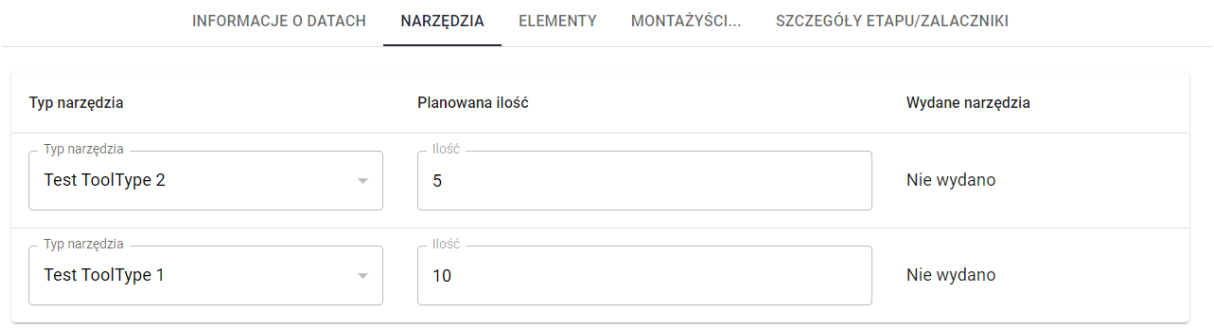
W polu górnym przedstawione są generalne dane dotyczące etapu: jego nazwa, status, cena, planowana liczbą montażystów oraz minimalna liczba zdjęć jakie są wymagane do zrobienia po skończonych pracach. Poniżej znajduje się zestaw pięciu zakładek, z których każda prezentuje unikalne treści, tworząc różnorodne obszary informacyjne. Do wyboru są:

a. Informacje o datach, przedstawione na rysunku 69 – W tej zakładce możemy dowiedzieć się o zaplanowanej dacie i godzinie rozpoczęcia oraz zakończenia etapu oraz o faktycznym momencie rozpoczęcia i zakończenia prac.



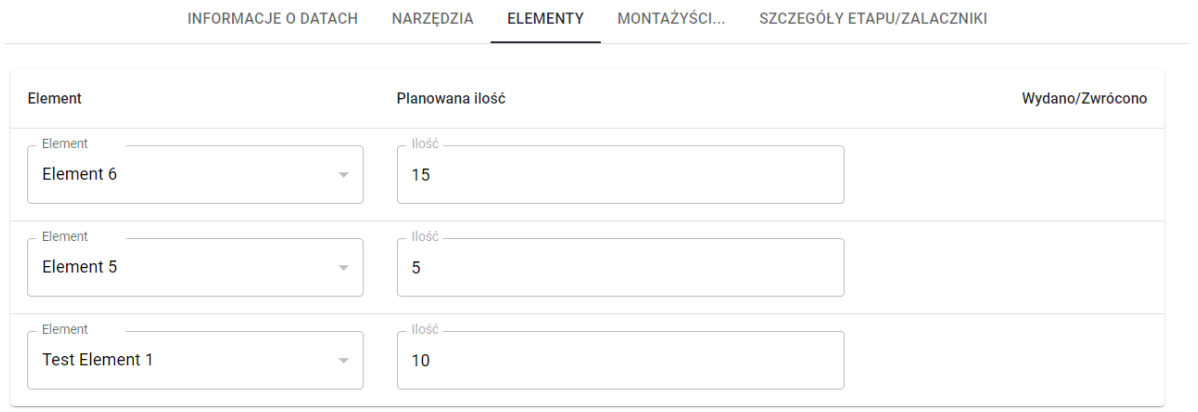
Rysunek 69 E-montażysta – Widok Lista zleceń, Informacje o datach dla aplikacji webowej. Źródło: Opracowanie własne.

b. Narzędzia przedstawione na rysunku 70 – Zakładka ta przedstawia jakie Typy narzędzi i w jakiej ilości zostały zaplanowane do wykorzystania przy wykonywaniu etapu. W kolumnie Wydane narzędzia widzimy dodatkowo kody konkretnych narzędzi wydanych do prac nad etapem. Każde z nich pozwala na przejście do widoku szczegółów narzędzia.



Rysunek 70 E-montażysta – Widok Lista zleceń, Narzędzia dla aplikacji webowej. Źródło: Opracowanie własne.

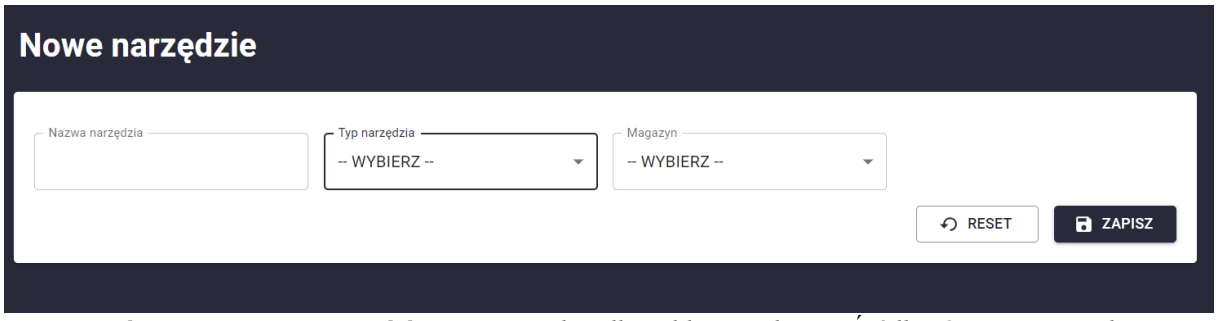
c. Elementy przedstawione na rysunku 71 - Podobnie jak w przypadku widoku narzędzi, w zakładce Elementy widzimy zaplanowane elementy oraz ich ilość potrzebną do wykonania etapu. Dodatkowo otrzymujemy informacje na temat tego, ile elementów zostało wydanych oraz zwróconych do magazynu



Rysunek 71 E-montażysta – Widok Lista zleceń, Elementy dla aplikacji webowej. Źródło: Opracowanie własne.

6.1.7 Widok Narzędzia, Typy Narzędzi i Magazyny

Rysunki 72-74 przedstawiają trzy powiązane widoki: Narzędzia, Typy Narzędzi oraz Magazyny.



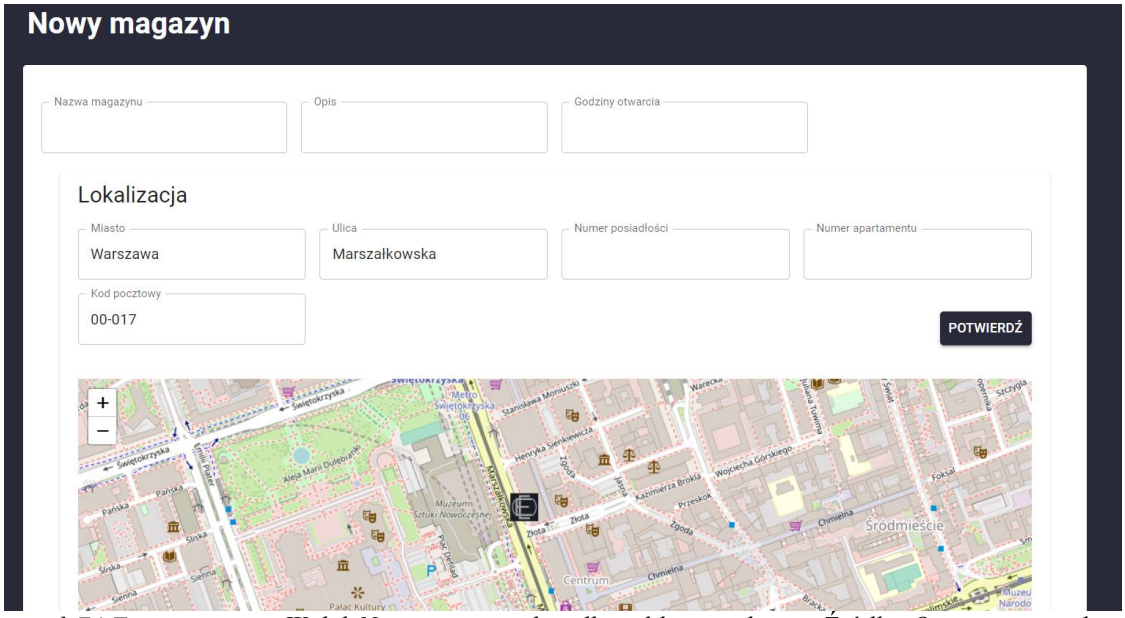
Rysunek 72 E-montażysta – Widok Nowe narzędzie dla aplikacji webowej. Źródło: Opracowanie własne.

Widok Nowe Narzędzie umożliwia dodawanie nowych narzędzi do systemu. Tylko Magazynier oraz Kierownik Magazynu mają takie uprawnienia. Aby to zrobić, użytkownik musi wybrać odpowiedni

typ narzędzia z listy. Jeśli jest pusta, Kierownik Magazynu musi dodać nowy typ narzędzia z poziomu widoku Nowy Typ Narzędzia.

Rysunek 73 E-montażysta – Widok Nowy typ narzędzia dla aplikacji webowej. Źródło: Opracowanie własne.

Po wyborze typu narzędzia, użytkownik musi wybrać magazyn, do którego narzędzie zostanie przypisane. Tylko Administrator firmy ma możliwość dodawania nowych magazynów.

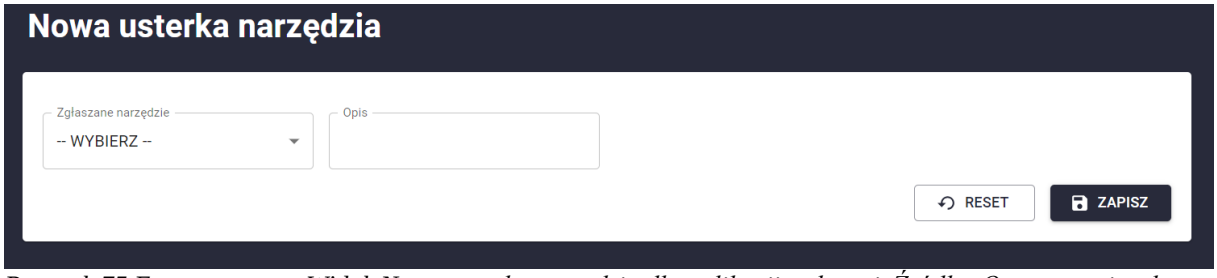


Rysunek 74 E-montażysta – Widok Nowy typ narzędzia dla aplikacji webowej. Źródło: Opracowanie własne.

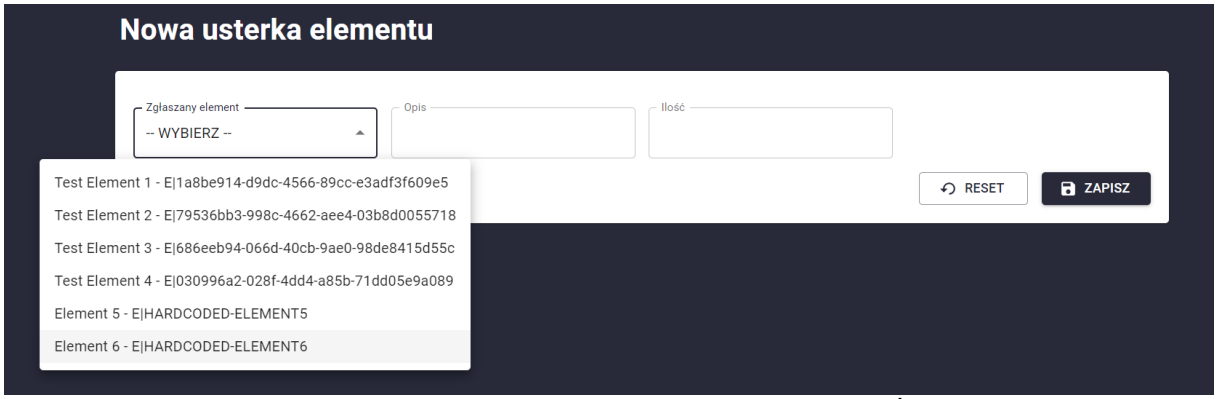
Aby wybrać lokalizację magazynu, użytkownik może skorzystać z funkcji wskazywania znacznikiem na mapie. Po wybraniu odpowiedniego miejsca, pola adresowe w widoku automatycznie wypełnią się odpowiednimi wartościami. Użytkownik może również ręcznie wprowadzić te dane w odpowiednich polach tekstowych.

6.1.8 Widok Usterki

Widok Usterka dostępny dla Magazyniera, Managera oraz Montażysty umożliwia zgłaszanie usterek narzędzi lub elementów w systemie. Należy wybrać odpowiednią pozycję z listy narzędzi lub elementów, które są obecne w systemie. Następnie użytkownik ma możliwość wpisania opisu usterki, w który sprecyzuje powstały problem. W przypadku zgłaszania usterki dotyczącej elementu, użytkownik ma również możliwość wprowadzenia ilości produktu, na której ona występuje. Rysunek 75 i 76 przedstawiają opisane widoki Nowa usterka narzędzia i Nowa usterka elementu.



Rysunek 75 E-montażysta – Widok Nowa usterka narzędzia dla aplikacji webowej. Źródło: Opracowanie własne.



Rysunek 76 E-montażysta – Widok Nowa usterka elementu dla aplikacji webowej. Źródło: Opracowanie własne.

6.2. Aplikacja mobilna

W poniższym podrozdziale przedstawiono wygląd i ekrany aplikacji na android.

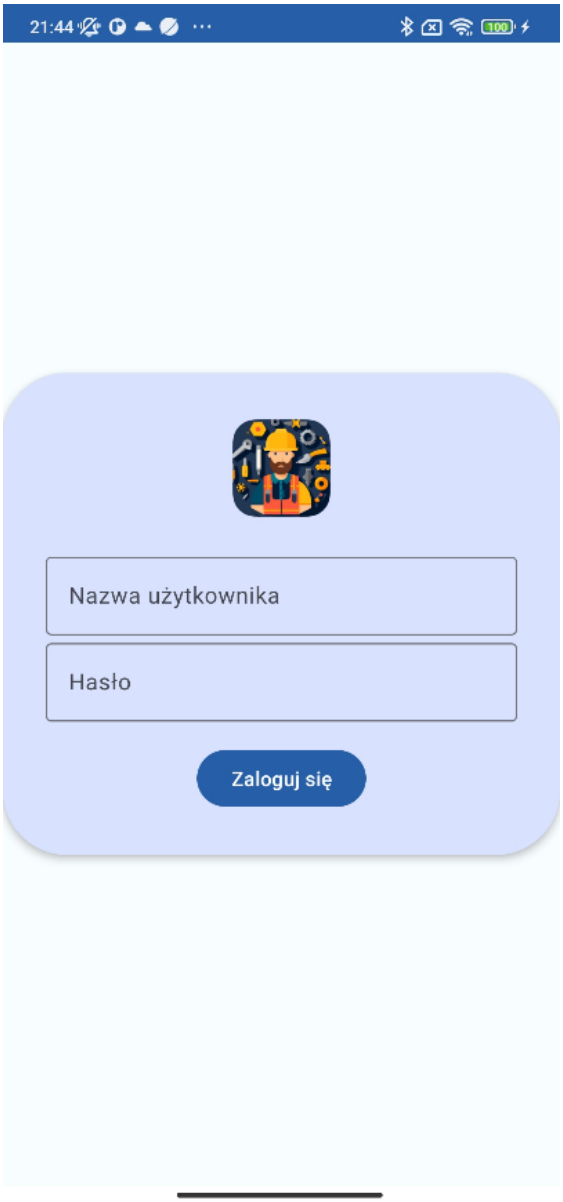
Interfejs aplikacji jest ważną częścią projektu to właśnie z jej pomocą użytkownik prowadzi interakcję z systemem. Przy stworzeniu aplikacji kierowano się 3 podstawowymi zasadami.

- Prostota i przejrzystość;
- Responsywność;
- Atrakcyjny wygląd;

Aplikacja powstała z myślą, aby pracownik w terenie mógł podejrzeć w sposób prosty i szybki szczegóły dotyczące zleceń i do wydawania elementów do zleceń poprzez skanowanie kodów QR. Na rysunkach 77-81 przedstawiono wybrane ekrany.

6.2.1 Ekran logowania

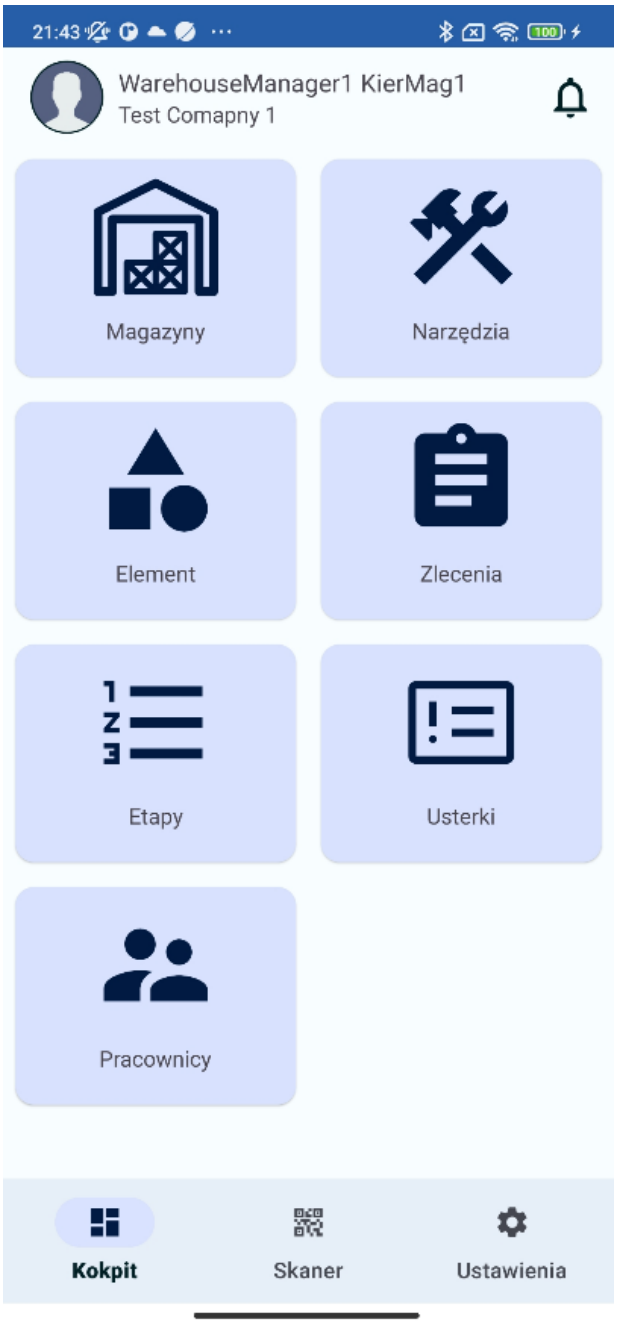
Na Rysunku 77 przedstawiony jest pierwszy widok, który użytkownik zobaczy przy logowaniu do aplikacji.



Rysunek 77 E-montażysta - Ekran logowania. Źródło Opracowanie własne.

6.2.2 Dashboard

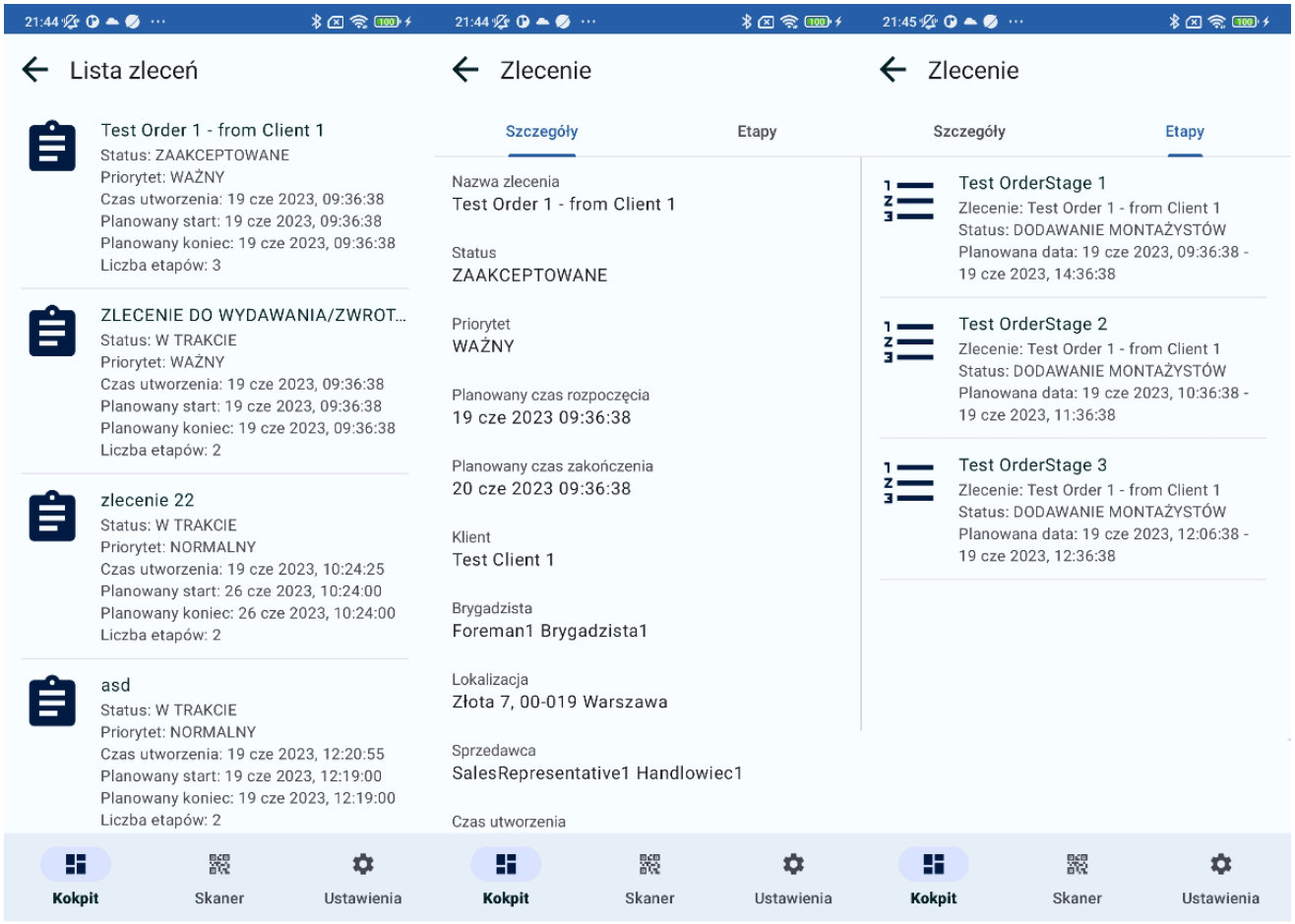
Na Rysunku 78 przedstawiony jest główny panel aplikacji, który daje użytkownikowi możliwość podejmowania interakcji z aplikacją. Z tego panelu użytkownik ma kontrolę nad funkcjonalnościami i opcjami dostępnymi w aplikacji, umożliwiając mu wykonywanie zadań.



Rysunek 78 E-montażysta – Dashboard. Źródło: Opracowanie własne.

6.2.3 Ekrany zlecenia

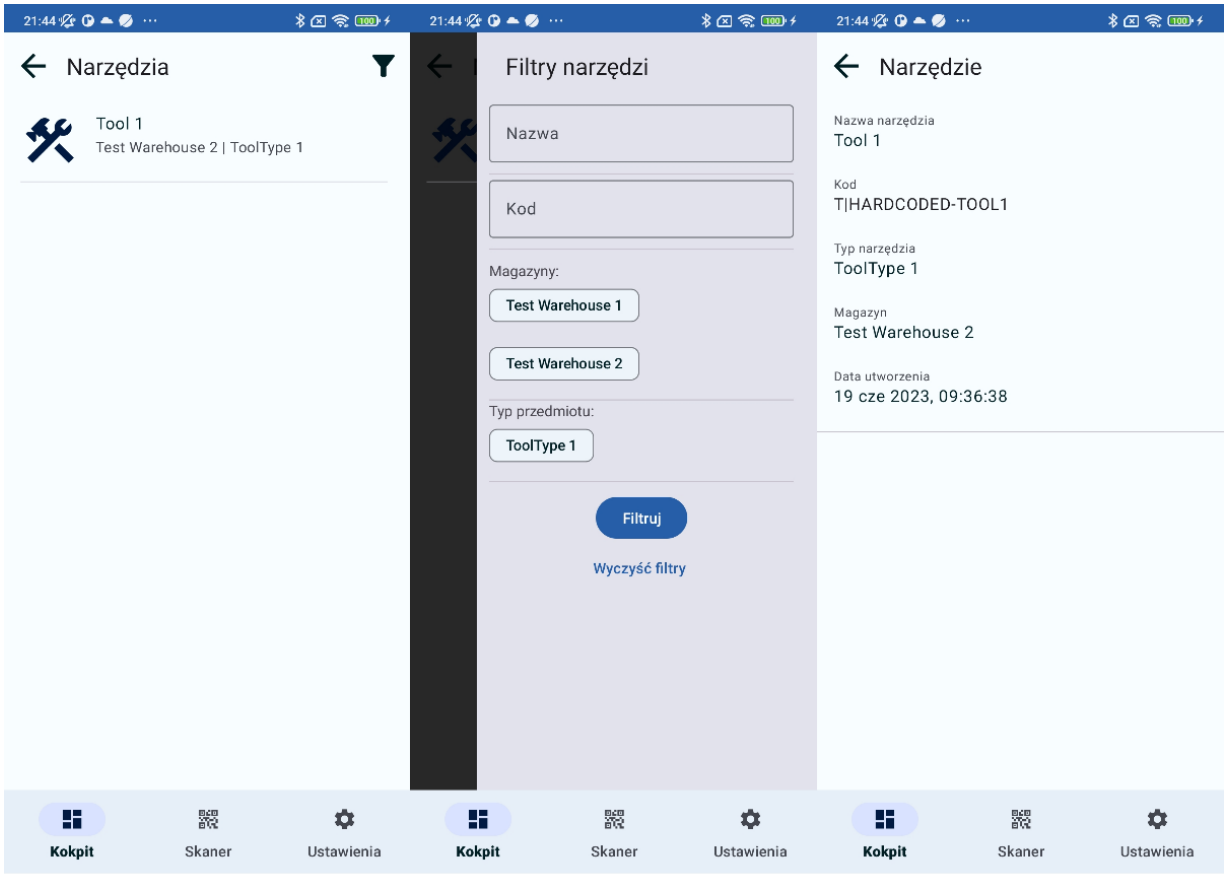
Na Rysunku 79 przedstawione są trzy ekrany związane z zleceniami. Na lewym ekranie znajduje się lista zleceń. Wyświetlane są podstawowe informacje, takie jak status zlecenia, klient, czas utworzenia itp. Na środkowym ekranie przedstawiono widok, na którym użytkownik ma możliwość wybrania konkretnego zlecenia z listy i wyświetlenia jego szczegółów. Zawiera bardziej szczegółowe informacje, takie jak przypisany brygadzista czy sprzedawca, lokalizacja, priorytet i inne informacje istotne dla danego zlecenia. Na prawym ekranie przedstawiona jest lista etapów zlecenia. Użytkownik może zobaczyć wszystkie etapy, które są częścią wybranego zlecenia. Każdy etap może zawierać nazwę, planowaną datę rozpoczęcia, status oraz inne istotne informacje.



Rysunek 79 E-montażysta – Widoki zleceń. Źródło: Opracowanie własne.

6.2.4 Szczegóły narzędzi

Rysunek 80 przedstawia trzy ekrany związane z narzędziami. Na lewym ekranie znajduje się widok listy narzędzi. Wyświetlane są podstawowe informacje, takie jak nazwa, kategoria, magazyn itp. Widok szuflady filtrów znajduje się w środku. Jest to panel, który pozwala użytkownikowi na filtrowanie narzędzi według różnych kryteriów. Może to obejmować wybór określonego typu przedmiotu, zastosowanie konkretnego magazynu lub jakiekolwiek inne dostępne filtry, które ułatwiają wyszukiwanie odpowiednich narzędzi.

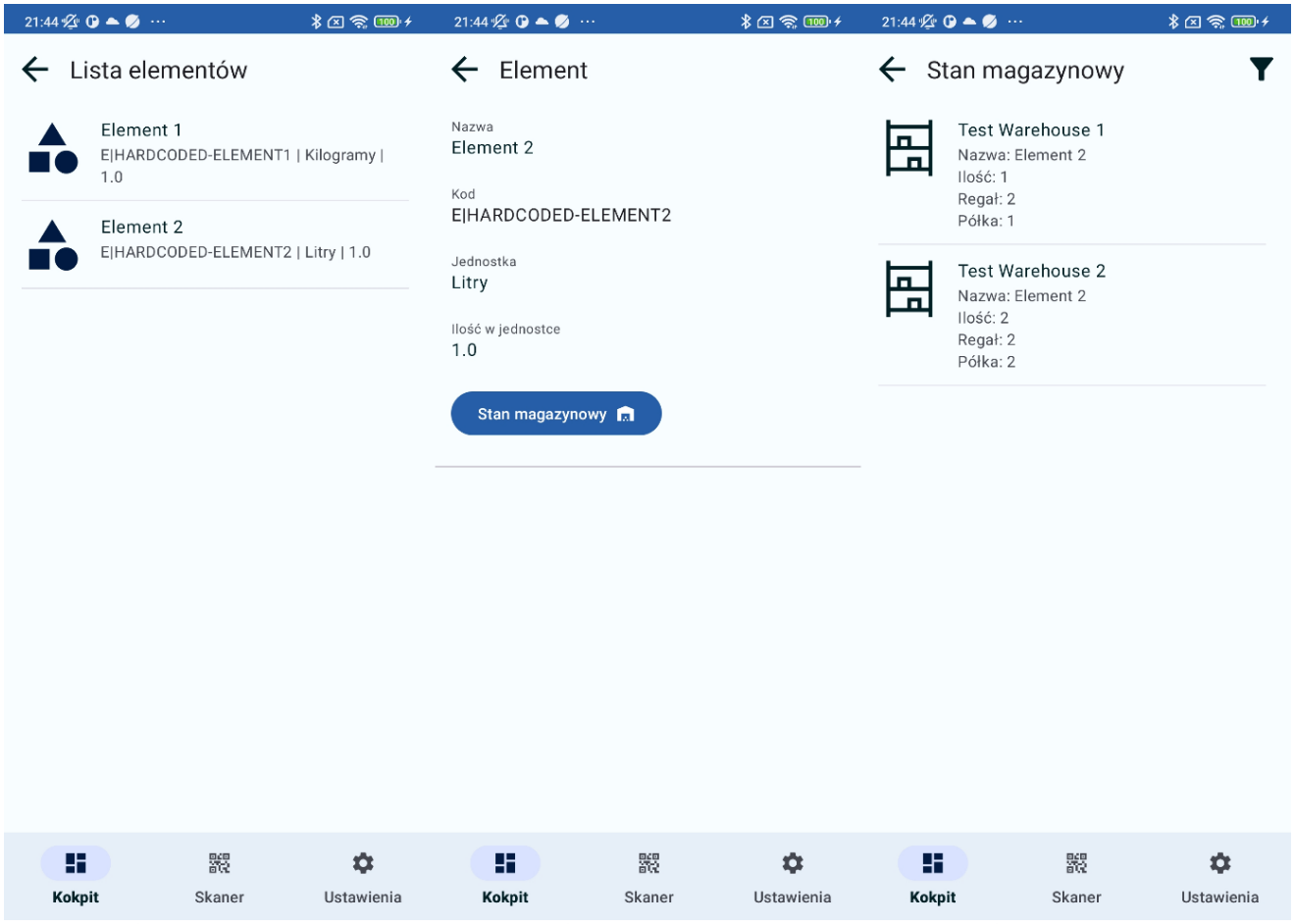


Rysunek 80 E-montażysta – Szczegóły narzędzi. Źródło: Opracowanie własne.

6.2.5 Widok elementów

Rysunek 81 przedstawia trzy widoki związane z elementami. Na lewym ekranie znajduje się widok listy, który prezentuje użytkownikowi spis elementów. Wyświetlane są podstawowe informacje, takie jak nazwa, kod, jednostka miary itp. Widok szczegółów elementu znajduje się w środku. Po

wybraniu konkretnego elementu z listy, użytkownik może zobaczyć jego bardziej szczegółowe informacje. Użytkownik może zapoznać się z tymi szczegółami i uzyskać kompletny obraz danego elementu. Na prawym ekranie przedstawiony jest widok stanu magazynowego. Pokazuje on informacje dotyczące dostępności i ilości danego elementu w magazynie. Użytkownik może sprawdzić aktualne stany magazynowe, np. ilość dostępnych sztuk, regał i półkę na których jest zlokalizowany, itp. Ten widok pomaga użytkownikowi w monitorowaniu stanu zapasów.



Rysunek 81 E-montażysta – Widok elementów. Źródło: Opracowanie własne.

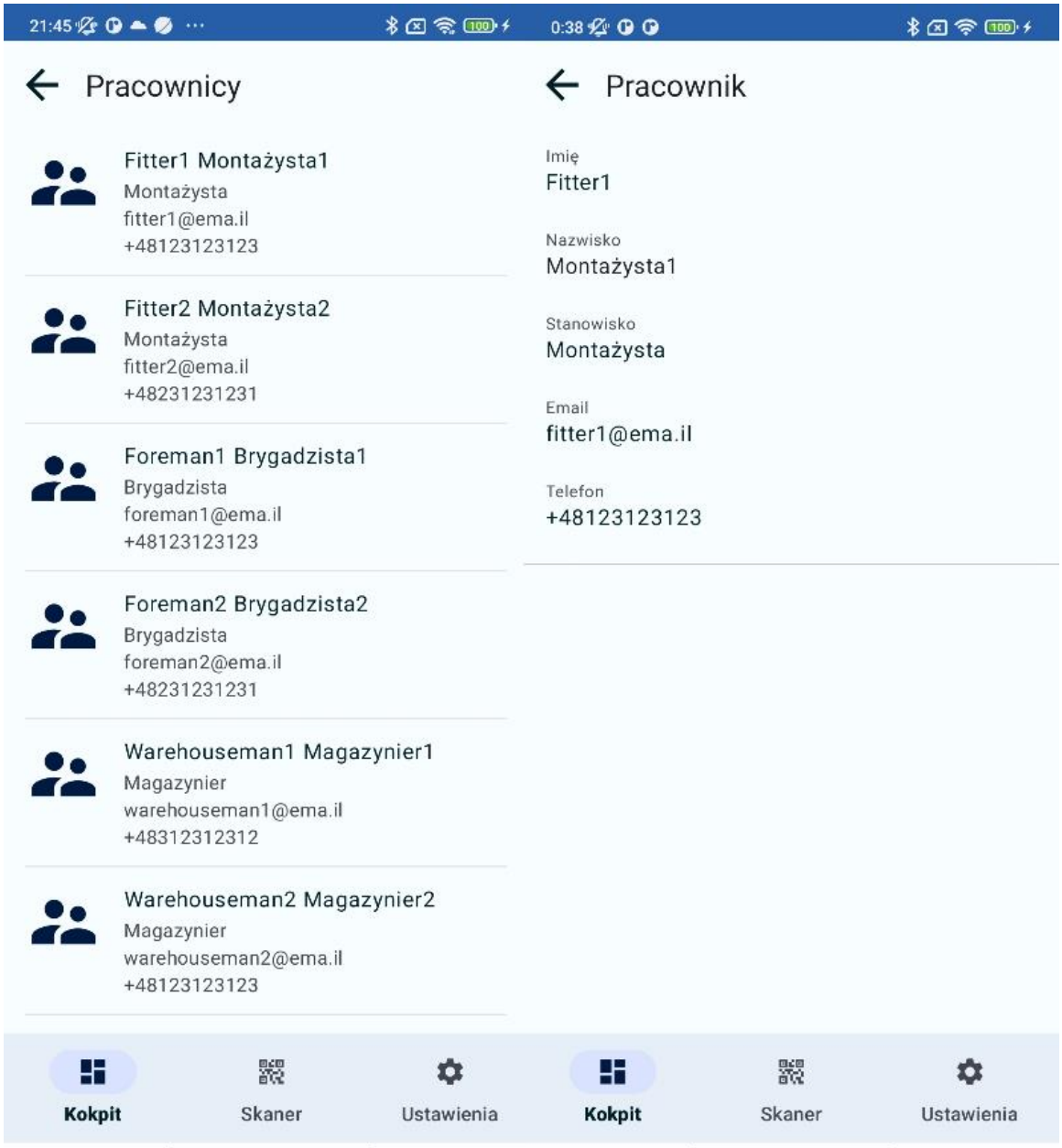
6.2.6 Widok powiadomień

Na Rysunku 82 przedstawiony jest widok powiadomień. Na tym ekranie użytkownik może zobaczyć zgłoszone powiadomienia, takie jak informacje o usterek narzędzi, ostrzeżenia lub inne istotne komunikaty systemowe.

Rysunek 82 E-montażysta – Widok powiadomień. Źródło: Opracowanie własne.

6.2.7 Widok pracowników

Rysunek 83 przedstawia dwa widoki związane z pracownikami. Na lewym ekranie znajduje się widok listy pracowników. Wyświetlane są podstawowe informacje, takie jak imię, nazwisko, stanowisko, kontakt itp. Widok szczegółów pracownika znajduje się po prawej stronie. Po wybraniu konkretnego pracownika z listy, użytkownik może zobaczyć bardziej szczegółowe informacje na jego temat.



Rysunek 83 E-montażysta – Widok pracowników. Źródło: Opracowanie własne.

7. Testy funkcjonalne

Ważnym elementem przy tworzeniu oprogramowania jest przeprowadzenie testów, które mają na celu weryfikację czy system działa poprawnie, zgodnie z założeniami klienta oraz spełnia jego wymagania. Wiele systemów, ze względu na zaniedbanie tej części procesu wytwarzania, ostatecznie pozostaje nieukończonych i niezaakceptowanych przez klienta z powodu dużego poziomu awaryjności oraz niezgodności z założeniami.

W przypadku naszego projektu, podjęliśmy decyzję o rozpoczęciu testowania natychmiast po utworzeniu pierwszych elementów i funkcjonalności aplikacji przez nasz zespół deweloperski, tak, aby jak najszybciej wykryć błędy i wyeliminować już na wczesnym etapie tworzenia naszego oprogramowania. Jest to kluczowe z względu, że w późniejszych etapach projektu znalezienie przyczyn błędu może być utrudnione i naprawa bardzo kosztowna, a nawet równoważna z napisaniem funkcjonalności programu od nowa.

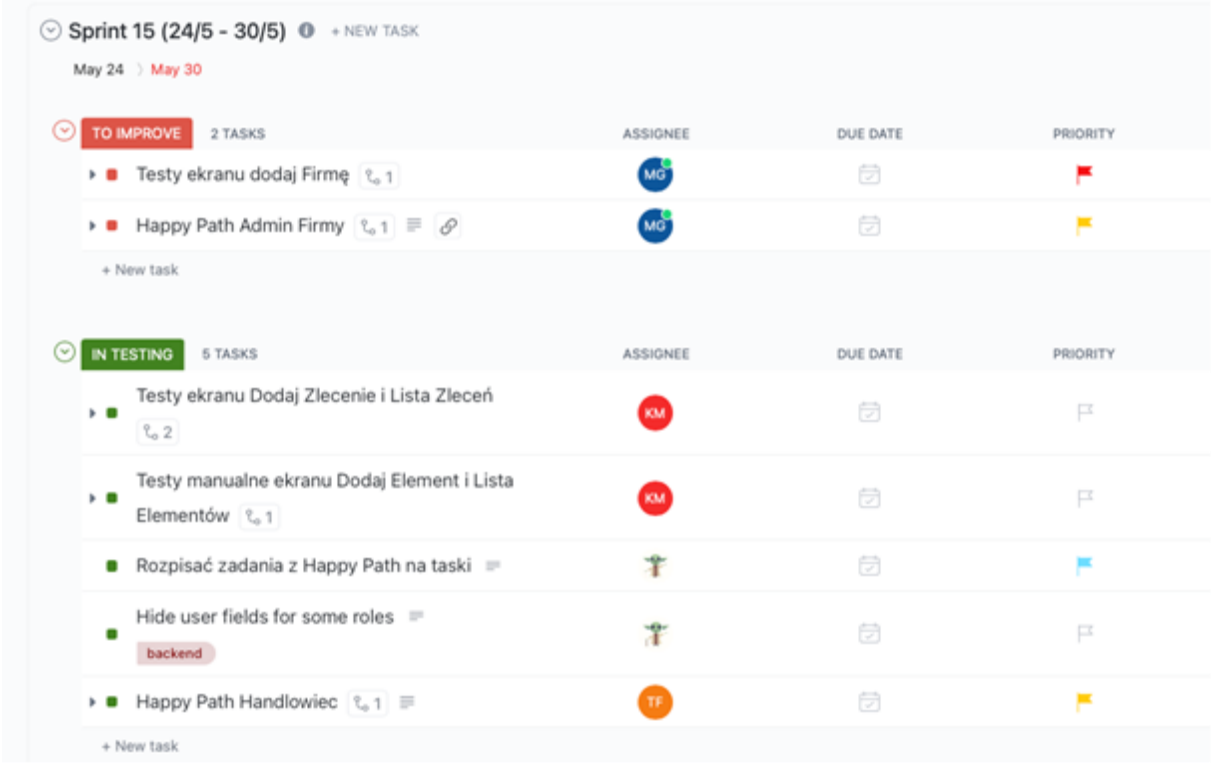
Jako zespół testów wykonaliśmy testy integracyjne polegające na sprawdzeniu czy nasze komponenty systemu współpracują ze sobą poprawnie i zgodnie z oczekiwaniem. Przeprowadziliśmy również testy systemowe. Jest to rodzaj testów oprogramowania, które weryfikują, czy system spełnia podstawione mu wymagania, działa zgodnie z życzeniem klienta, jest stabilny oraz wydajny. Testy przeprowadzone zostały na podstawie scenariuszy przypadków użycia stworzonych podczas analizy biznesowej.

Spotkania grupy testowej odbywały się w trybie tygodniowych sprintów odnotowanych w narzędziu ClickUp **Błąd! Nie można odnaleźć źródła odwołania.**. Podczas nich omawialiśmy aktualne s tatusy zadań, napotkane problemy i planowaliśmy następne etapy prac oraz ustalaliśmy priorytety zadań. Priorytety zadań i błędów zostały ustalone zgodnie z wartościami przedstawionymi w tabeli 12.

Tabela 11 Priorytety zadań i błędów. Źródło: Opracowanie własne.

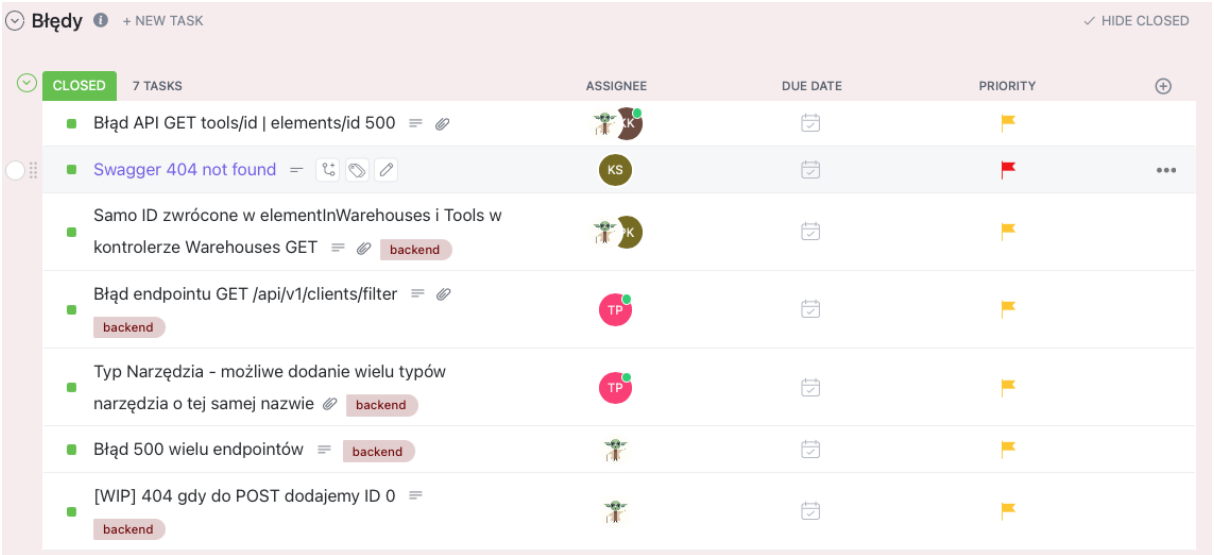
Piorytet	Opis
Urgent	Priorytet najwyższy – błąd krytyczny uniemożliwiający realizację scenariusza testowego i mający wpływ na pozostałe moduły lub scenariusze przypadków użycia.
High	Priorytet wysoki – błąd krytyczny uniemożliwiający realizację scenariusza testowego
Normal	Priorytet standardowy – błąd nieblokujący realizację scenariusza testowego.
Low	Priorytet niski – zmiany kosmetyczne, nie wpływające na działanie procesu

Przykładowy sprint grupy testów został przedstawiony na rysunku 84, który jest zrzutem ekranu z narzędzia ClickUp. Widać na nim widok sprintu z zaznaczonymi zadaniami przypisanymi do grupy testów.



Rysunek 84 ClickUp - Zrzut ekranu prezentujący przykładowy sprint grupy testów. Źródło: Opracowanie własne.

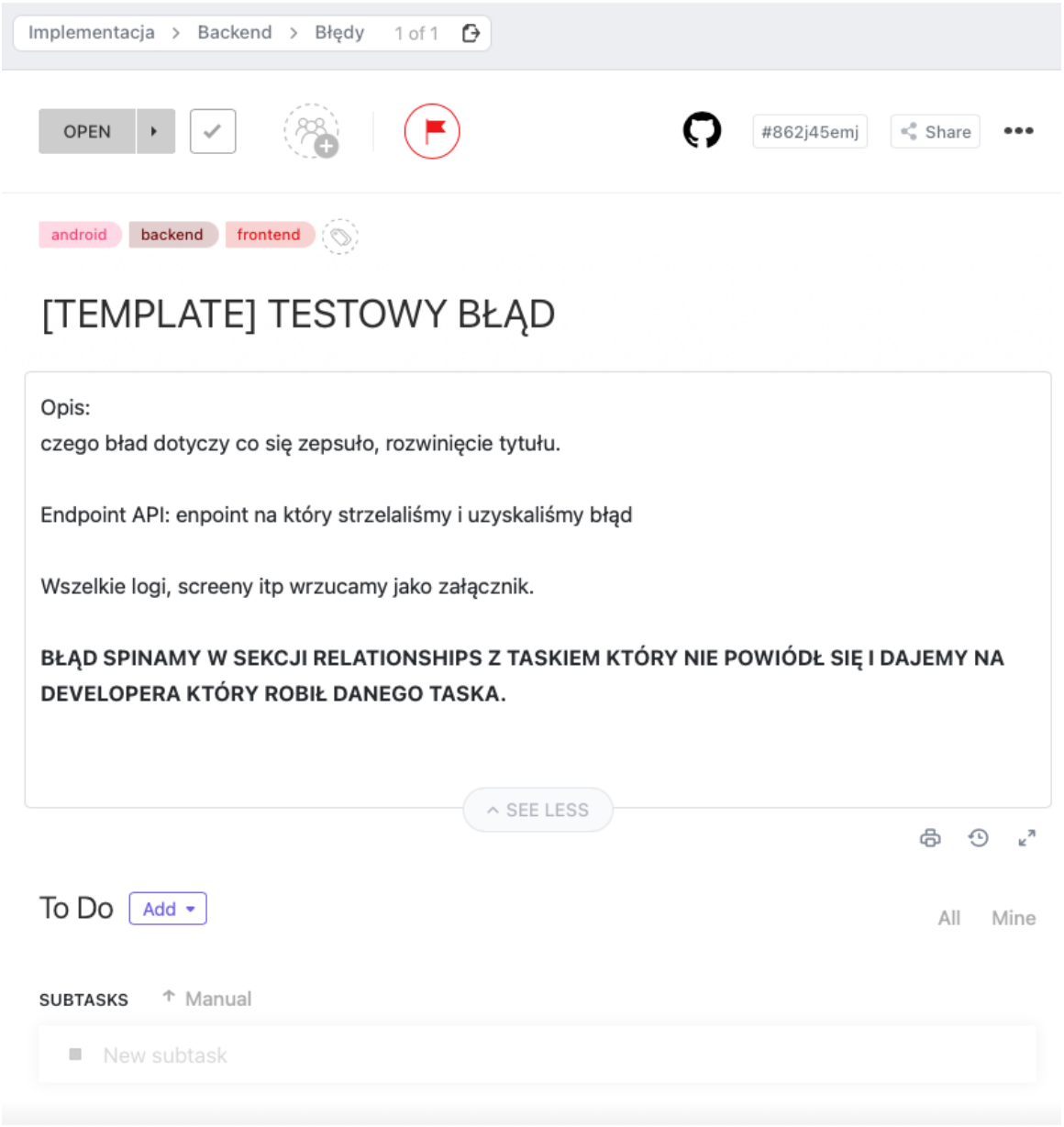
Podczas napotkania błędów, tester rejestruje wystąpienie błędu w systemie ClickUp, zgodnie z odpowiednią przestrzenią (Backend, Frontend, Android), w której został zidentyfikowany. Na przykład, jeśli błąd występuje na stronie aplikacji, zostaje on przypisany do listy Frontend, wraz ze wszystkimi niezbędnymi szczegółami umożliwiającymi programiście zreprodukowanie błędu u siebie. Rysunek 85 przedstawia zrzut ekranu z narzędzie ClickUp z listą zgłaszanych błędów.



Rysunek 85 ClickUp - Zrzut ekranu z listą zgłaszanych błędów. Źródło: Opracowanie własne.

Tester do zgłoszenia błędu wykorzystuje dedykowany szablon, który ułatwia zgłaszanie i dokumentowanie błędów w systemie ClickUp. Po zidentyfikowaniu błędu, tester wypełnia szablon, zawierający niezbędne informacje dotyczące błędu, takie jak opis problemu, końcówkę oraz inne istotne szczegóły. Dodatkowo, tester ustala priorytet zadania zgodnie z tabelą, która określa hierarchię

priorytetów dla zgłaszanych błędów. Priorytety te uwzględniają m.in. wpływ błędu na funkcjonalność systemu, stopień krytyczności oraz potencjalne konsekwencje dla użytkowników. Dzięki temu ustaleniu priorytetu, zespół odpowiedzialny za naprawę błędu może skoncentrować się na rozwiązywaniu najważniejszych problemów w pierwszej kolejności. Rysunek 86 przedstawia zrzut ekranu z narzędzia ClickUp z szablonem zgłaszania błędów.



Rysunek 86 ClickUp - Zrzut ekranu z szablonem błędu. Źródło: Opracowanie własne.

Podczas przeprowadzania testów korzystaliśmy z dostępnej publicznie platformy https://dev.emontazysta.pl, na której była udostępniona najnowsza wersja developerska aplikacji. Dodatkowo, testowaliśmy aplikację również lokalnie, pobierając kod z repozytorium na GitHub. Dzięki pracy na różnych gałęziach mieliśmy możliwość testowania funkcji przed ich wprowadzeniem do głównej gałęzi. W pierwszej fazie testów skoncentrowaliśmy się na przetestowaniu API za pomocą metody CRUD (Create, Read, Update, Delete), co oznacza, że sprawdzaliśmy funkcjonalność tworzenia, odczytu, aktualizacji i usuwania danych. Na każdym etapie testów przeprowadzaliśmy zarówno testy manualne, jak i automatyczne.

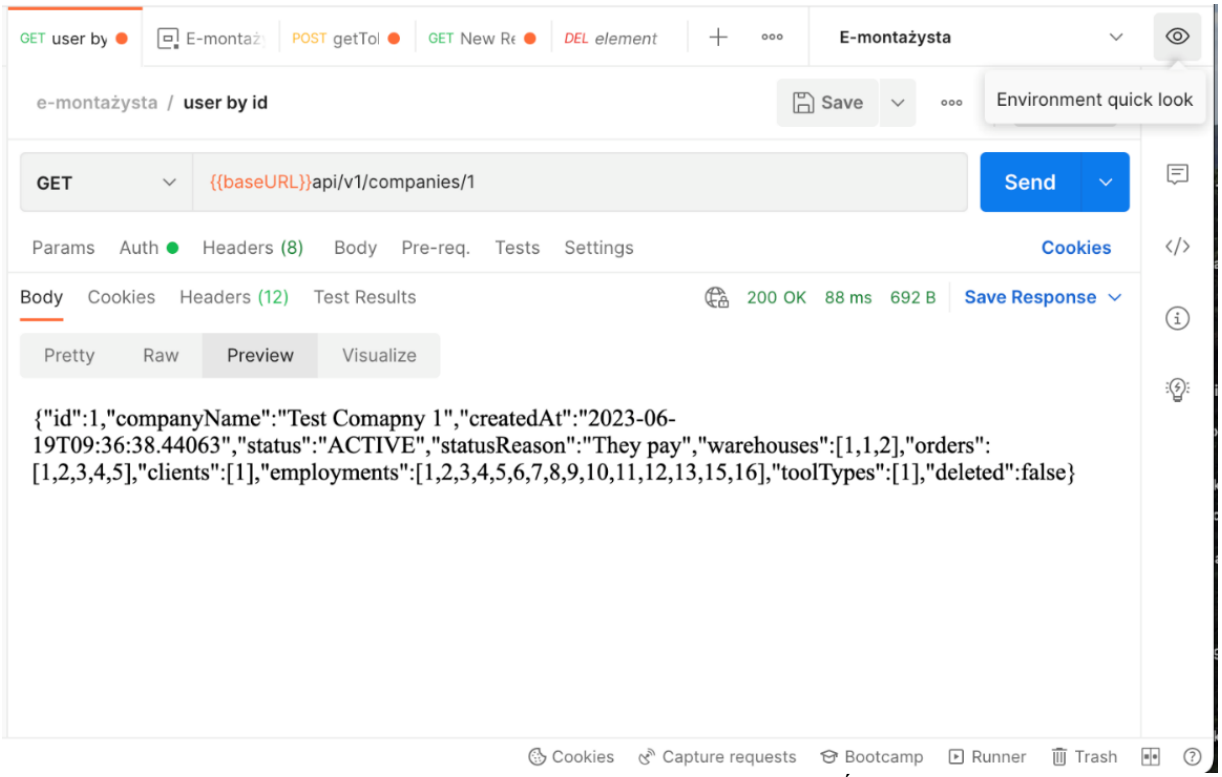
7.1. Testy manualne

Testy manualne API zostały przeprowadzone przy użyciu narzędzi takich jak Postman oraz Swagger. Odbywały się zgodnie z metodą czarnej skrzynki, co oznaczało, że tester nie był zaznajomiony z kodem odpowiedzialnym za testowaną funkcjonalność. Zamiast tego, skupiał się na testowaniu funkcji według wcześniej przygotowanego scenariusza testowego. W tabelach od 13 do 18, przedstawione zostaną wybrane scenariusze testowe dla kontrolerów.

Tabela 12 Test końcówki GET /company/{id}. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu GET /company/{id}	
Opis	Sprawdzenie działania endpointu do pobierania informacji o firmie	
Warunek początkowy	minimum 1 firma w bazie danych	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Za id podstaw id firmy do pobrania i wyślij request	Informacje o firmie przekazane do response endpointu
2	Potwierdź, że w informacje w bazie danych są takie same jak pobrane z endpointu.	Informacje zgadzają się z danymi z endpointu.
Wynik testu	Pozytywny	

Do przetestowania opisanego scenariusza wykorzystano narzędzie Postman oraz zapytanie przedstawione na rysunku 87.



Rysunek 87 Postman – zapytanie pod test końcówki GET /company/{id}. Źródło: Opracowanie własne.

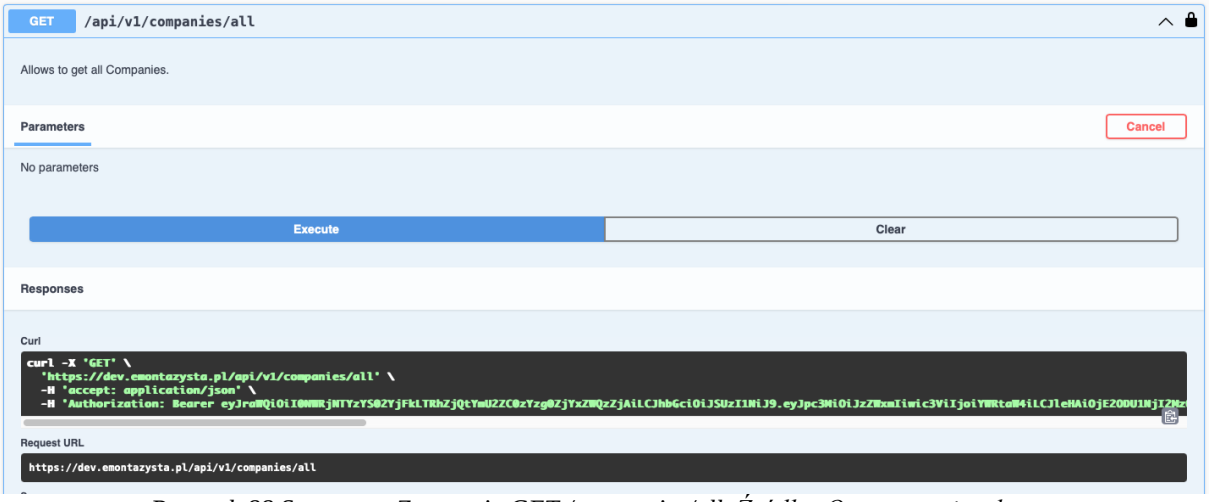
Tabela 13 Test końcówki DELETE /company/{id}. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu DELETE /company/{id}	
Opis	Sprawdzenie działania endpointu do usuwania firmy	
Warunek początkowy	minimum 1 firma w bazie danych	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Za id podstaw id firmy do usunięcia i wyślij request	Response potwierdza usunięcie firmy o podanym Id
2	Potwierdź, że w bazie danych firma została usunięta.	Firma usunięta z bazy danych
Wynik testu	Pozytywny	

Tabela 14 Test końcówki GET /companies/all. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu GET /companies/all	
Opis	Sprawdzenie działania endpointu do pobierania informacji o wszystkich dostępnych firmach	
Warunek początkowy	minimum 2 firmy w bazie danych	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Wyślij request do pobrania wszystkich firm.	Informacje o dostępnych firmach przekazane do response endpointu
2	Potwierdź, że w informacje w bazie danych są takie same jak pobrane z endpointu.	Informacje zgadzają się z danymi z endpointu.
Wynik testu	Pozytywny	

Dla przeprowadzenia prostych testów końcówek zastosowano również narzędzie Swagger. Przykładowo, dla scenariusza przedstawionego w tabeli 15, z wykorzystaniem tego narzędzia wyglądałoby to tak, jak przedstawiono na rysunku 88. Dodatkowo, odpowiedź serwera na to zapytanie została przedstawiona na rysunku 89.



Rysunek 88 Swagger - Zapytanie GET /companies/all. Źródło: Opracowanie własne.

Rysunek 89 Swagger - Zrzut ekranu na odpowiedź na zapytanie GET. Źródło: Opracowanie własne.

Tabela 15 Test końcówki POST /company. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu POST /company	
Opis	Sprawdzenie działania endpointu do utworzenia nowej firmy	
Warunek początkowy	brak	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Wyślij request utworzenia nowej firmy.	Informacje o firmie przekazane w formie json do systemu i dodane do bazy.
2	Potwierdź, że w informacje w bazie danych są takie same jak te przesłane w requestcie	Informacje zgadzają się z danymi wysłanymi w requestcie.

Wynik testu	Pozytywny	

Tabela 16 Test końcówki POST /company. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu PUT /company/{id}	
Opis	Sprawdzenie działania endpointu do zaktualizowania już istniejącej firmy	
Warunek początkowy	firma o podanym id istnieje w bazie danych.	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Wyślij request utworzenia nowej firmy.	Informacje o firmie przekazane w formie json do systemu i dodane do bazy.
2	Potwierdź, że w informacje w bazie danych są takie same jak te przesłane w requestcie	Informacje zgadzają się z danymi wysłanymi w requestcie.
Wynik testu	Pozytywny	

Tabela 17 Test końcówki GET/elements/{id}. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test endpointu GET/elements/{id}	
Opis	Sprawdzenie działania endpointu GET/elements/{id} do pobierania elementu z bazy danych	
Warunek początkowy	Minimum jeden element w bazie danych	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Podaj numer id elementu do pobrania i wyślij zapytanie.	Zapytanie zwróciło response z informacją o elemencie
2	Sprawdź w bazie danych i potwierdź że otrzymany element jest tym samym co element w bazie danych	W bazie istnieje ten sam element który został zwrócony
Wynik testu	Negatywny – wymagana naprawa	

7.2. Testy automatyczne

Zespół testów poza testami manualnymi napisał i przeprowadził testy automatyczne przy użyciu biblioteki Rest-assured i narzędzia Selenium. Pierwszy sposób został wykorzystany głównie do testów API, a drugi do testów widoków oraz przykładowych scenariuszy zachowania użytkownika. Narzędzia zostały szerzej opisane w rozdziale 4.

W tabeli 19 przedstawiono scenariusz testowy dla sprawdzenia poprawności działania API dla obiektu firma, który został przetestowany przy użyciu biblioteki Rest assured. Fragment został przedstawiony na listingu 44 oraz na rysunku 90 wynik przeprowadzenia testu.

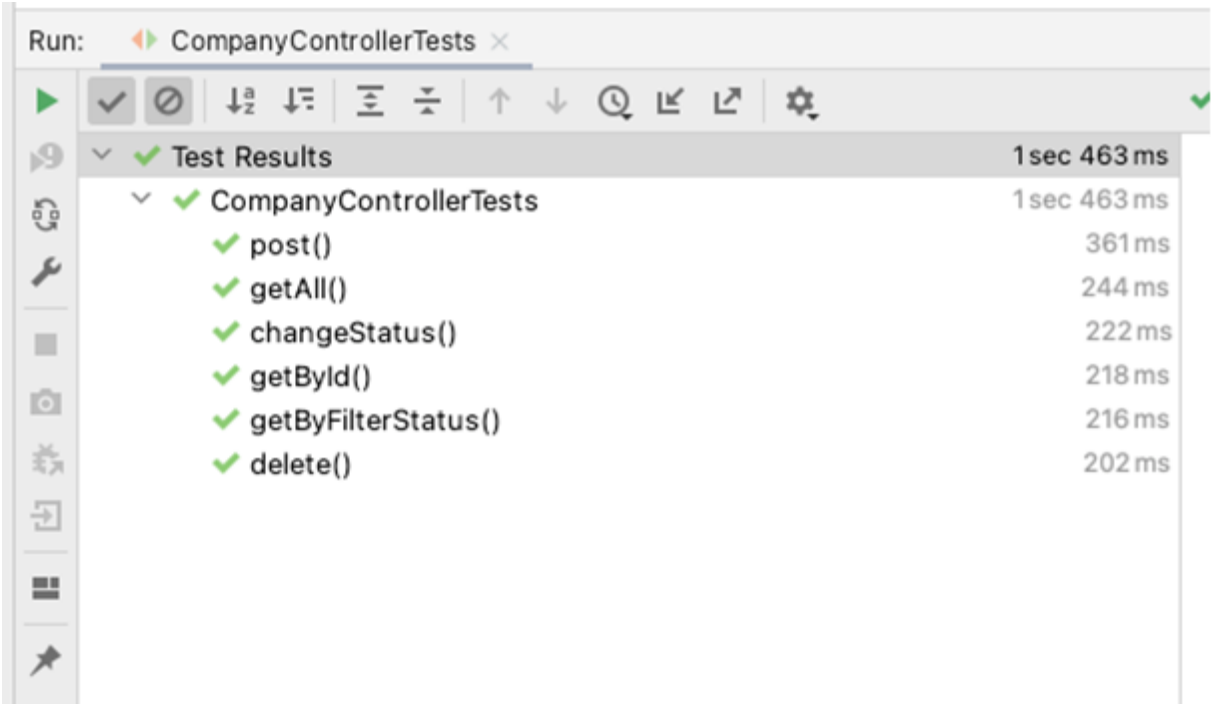
Tabela 18 Test kontrolera firmy. Źródło: Opracowanie własne.

Nazwa scenariusza testowego	Test API CompanyController	
Opis	Sprawdzenie działania API dla obiektu Firma	
Warunek początkowy	brak	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Dodanie nowej firmy.	Dodana firma
2	Pobranie wszystkich firn i sprawdzenie czy dodana firma z kroku 1 znajduje się na liście	Firma jest na liście
3	Zmiana statusu firmy na nieaktywna.	Status zmieniony
4	Pobranie firmy przy użyciu ID	Dane zgadzają się pobranej firmie
5	Sprawdzenie poprawności działania filtrowania po statusie	Lista firm z statusem
6	Usunięcie firmy	Usunięcie firmy
7	Usunięcie nieistniejącej firmy	Informacja o nieudanym usunięciu
Wynik testu	Pozytywny	

Listing 44 E-montażysta - Test kontrolera firmy dla oprogramowania Selenium. Źródło: Opracowanie własne.

```
@Test
@Order(1)
public void post() {
    Response response = given()
            .header("Content-type", "application/json")
            .header("Authorization", "Bearer " + AbstractTest.TOKEN)
            .and()
            .body(postBody)
            .when()
            .post(AbstractTest.BASE_PATH + "/companies")
            .then()
            .extract().response();

    response.getBody().prettyPeek();
    companyId = response.getBody().jsonPath().getInt("id");

    Assertions.assertEquals(201, response.statusCode());
}
@Test
@Order(2)
void getAll() {
    Response response = given()
            .header("Authorization", "Bearer " + AbstractTest.TOKEN)
            .contentType(ContentType.JSON)
            .when()
            .get(AbstractTest.BASE_PATH + "/companies/all")
```

```
            .then()
            .extract().response();

    response.getBody().prettyPeek();

    Assertions.assertEquals(200, response.statusCode());
    Assertions.assertTrue(response.getBody().jsonPath().getList("$").st
ream()
            .anyMatch(company -> company.toString().contains("id=" +
companyId)));
}

@Test
@Order(3)
void changeStatus() {
    Response response = given()
            .header("Content-type", "application/json")
            .header("Authorization", "Bearer " + AbstractTest.TOKEN)
            .and()
            .body(putBody)
            .when()
            .put(AbstractTest.BASE_PATH + "/companies/" + companyId)
            .then()
            .extract().response();

    response.getBody().prettyPeek();
    System.out.println(putBody);
    Assertions.assertEquals(200, response.statusCode());
    Assertions.assertEquals(status,
response.getBody().jsonPath().getString("status")); }
```



Rysunek 90 E-montażysta - Test kontrolera firmy. Źródło: Opracowanie własne.

W tabeli 20 przedstawiono prawidłowo działający system z perspektywy menadżera firmy, który zawiera takie ścieżki jak:

- Edycja zlecenia (priorytet i przypisanie brygadzisty)
- Edycja usterki (zmień datę zakończenia/datę wysłania do serwisu/opis/status/dodaj załącznik)
- Zarządzanie nieobecnościami (dodawanie, usuwanie i edycja)
- Dodawanie komentarza do zapotrzebowania ad hoc.

A kryteria akceptacji zostały ustalone na:

- Wszelkie walidacje na formularzu powinny spełniać wymogi.
- Po utworzeniu danego obiektu sprawdzić czy (jeśli występują) wszystkie powiązania z innymi obiektami są poprawne i wyświetlają się.
- Zmiana dowolnego parametru obiektu powinna zostać odzwierciedlona po naciśnięciu przycisku zapisz. Wtedy formularz włącza tryb tylko do odczytu a nowe dane są uwzględnione w widoku szczegółów obiektu.
- Obiekty wyświetlane w listach musza przynależeć do FIRMY, do której przynależy zalogowany przez nas użytkownik tj. logujemy się osobą która jest zatrudniona w firmie X to wszystkie zlecenia, usterki i inne obiekty mają być przypisane do firmy X, wszelkie odstępstwa od tej reguły powinny być zgłaszane.

Do tego scenariusza test został napisany w Selenium zaprezentowany na listingu 45 oraz na rysunku 91.

Tabela 19 Scenariusz Happy Path Manager Źródło: Opracowanie własne.

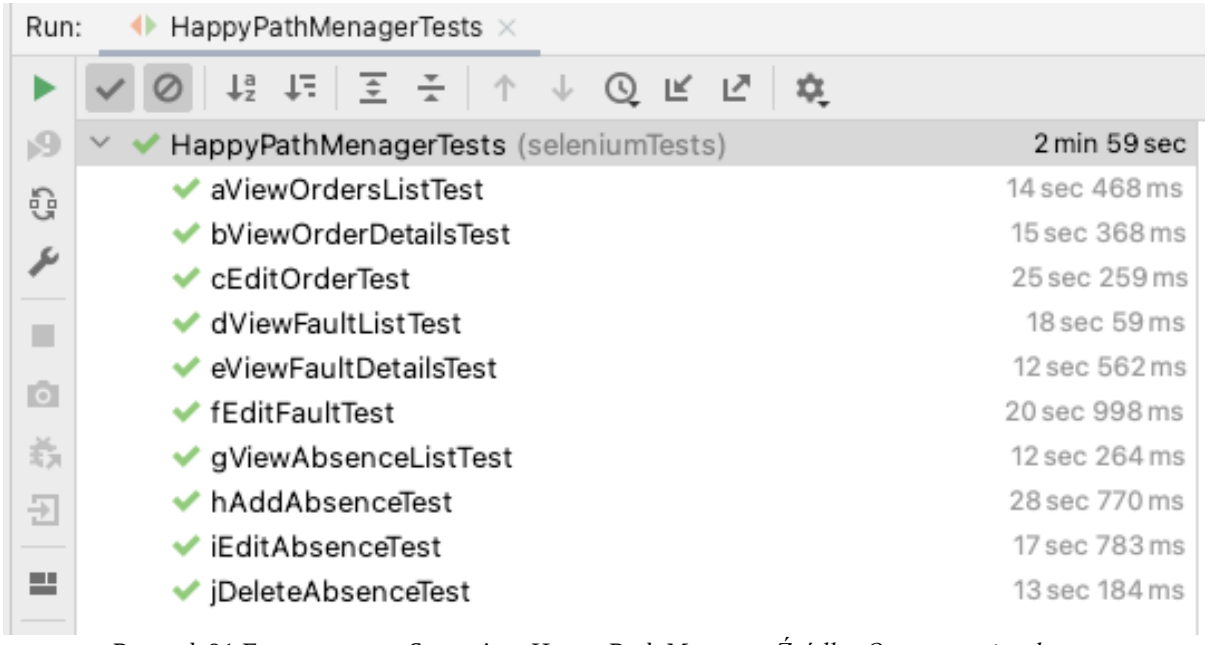
Nazwa scenariusza testowego	Happy Path Manager	
Opis	Scenariusz testowy dla pracy Managera	
Warunek początkowy	Utworzone konto Managera w firmie	
Przepływ zdarzeń		
Krok	**Test**	**Rezultat**
1	Zalogowanie do platformy	
2	Zlecenia • Wyświetlenie listy zleceń • Wyświetlanie szczegółów zlecenia i etapów • Edycja zlecenia poprzez zmiany priorytetu • Przypisanie brygadzisty do zlecenia	
3	Usterki • Wyświetlenie listy usterek • Wyświetlenie szczegółów usterki • Edycja usterki poprzez zmianę daty zakończenia, opisu,statusu Dodanie załącznika	
4	Nieobecności • Wyświetlenie listy nieobecności • Dodanie nieobecności • Usunięcie nieobecności • Edycja nieobecności	
5	Zapotrzebowanie Adhoc	

	• Wyświetlenie listy zleceń • Wyświetlenie listy zapotrzebowań adhoc • Dodanie komentarza do zapotrzebowania adhoc
Wynik scenariusza	Pozytywny

Listing 45 E-montażysta - Scenariusz Happy Path Manager. Źródło: Opracowanie własne.

```
public class HappyPathMenagerTests {

    private WebDriver driver;
    private WebDriverWait wait;

    @Before
    public void setUp() {
        WebDriverManager.firefoxdriver().setup();
        driver = new FirefoxDriver();
        wait = new WebDriverWait(driver, Duration.ofMillis(1000));

        driver.get("https://dev.emontazysta.pl/login");
        driver.manage().window().maximize();

        wait.until(ExpectedConditions.elementToBeClickable(By.id("usern
ame"))).click();

        driver.findElement(By.id("username")).sendKeys("manager1");


        driver.findElement(By.id("login-logIn")).click();

    }
    @After
    public void tearDown(){
        driver.quit();
    }

    @Test
    public void aViewOrdersListTest() throws InterruptedException {


        wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(
"navBtn-/orders")));
        driver.findElement(By.id("navBtn-/orders")).click();

        driver.findElement(By.id("navMenu-/orders")).click();

        Thread.sleep(5000);

        driver.findElement(By.id("navBtn-logout")).click();
        Thread.sleep(1000);
    }

    @Test
    public void bViewOrderDetailsTest() throws InterruptedException {
```

```
        wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(
"navBtn-/orders")));
        driver.findElement(By.id("navBtn-/orders")).click();

        driver.findElement(By.id("navMenu-/orders")).click();

        wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpa
th("/html/body/div/div/div[2]/div/div/table/tbody/tr[1]/td[1]")));
        driver.findElement(By.xpath("/html/body/div/div/div[2]/div/div/
table/tbody/tr[1]/td[1]")).click();

        Thread.sleep(5000);

        driver.findElement(By.id("navBtn-logout")).click();
        Thread.sleep(1000);
    }
```



Rysunek 91 E-montażysta - Scenariusz Happy Path Manager. Źródło: Opracowanie własne.

8. Podsumowanie i wnioski

W tym rozdziale omówiono trudności napotkane podczas tworzenia systemu E-montażysta oraz przedstawiono propozycje dotyczące dalszego rozwoju.

8.1. Wprowadzenie

Tworzenie aplikacji od podstaw, obejmujące cały proces powstawania oprogramowania, było wymagającym i długotrwałym zadaniem, trwającym przez dwa semestry nauki, uwzględniając w tym przerwy świąteczne. Realizacja projektu E-montażysta wiązała się z wyzwaniem organizacyjnym. W skład zespołu projektowego weszło jedenastu studentów informatyki ze specjalizacji Inżynierii Oprogramowania i Bazach Danych, o różnym poziomie wiedzy.

W trakcie realizacji projektu pojawiły się trudności związane z komunikacją wewnątrz zespołu, zróżnicowanymi umiejętnościami jego członków, nierzetelnym wykonywaniem zadań oraz brakiem motywacji niektórych uczestników. Dodatkowo, doszło do zmian w składzie zespołu oraz w obszarze kierownictwa projektu. Wszelkie napotkane problemy, były regularnie analizowane oraz podejmowane były działania mające na celu ich rozwiązanie, aby zakończenie prac nastąpiło zgodnie z ustalonym harmonogramem. Odpowiedzialność za to spoczywała na kierowniku projektu oraz kierownikach poszczególnych zespołów.

8.2. Zrealizowane cele

E-montażysta osiągnął sukces podczas prac programistycznych, realizując szereg założeń z początku projektu. Poniżej przedstawiono niektóre z osiągniętych celów:

1. System umożliwia rejestracje oraz logowanie się użytkowników o różnych rolach.
2. Stworzone zostały wszystkie zaplanowane funkcjonalności obsługi zleceń i magazynów.
3. Pracownicy mają możliwość komunikacji poprzez system powiadomień.
4. System umożliwia zarządzanie przechowywanymi w magazynach zasobami.
5. Aplikacja powstała w wersji webowej oraz mobilnej.

8.3. Struktura organizacyjna

W projekcie utworzono różne zespoły, takie jak zespół analizy, projektowania, implementacji, DevOps, testów oraz dokumentacji. Podczas całego przedsięwzięcia grupa działała jak przedsiębiorstwo, specjalizujące się w inżynierii oprogramowania.

8.4. Wyzwania i trudności

Trudności napotykane podczas rozwoju projektu wynikały z różnych przyczyn, w tym problemów projektowych, organizacyjnych i technicznych. W związku z tym, zespół musiał poświęcić wiele czasu na doskonalenie umiejętności i współpracę w rozwiązywaniu zadań. Oto lista problemów, które wystąpiły:

1. Zmienność składu zespołu - W trakcie trwania projektu, skład zespołu ulegał zmianom z różnych powodów, takich jak zmiana specjalizacji, trudności w realizacji przydzielonych zadań, a także sprawy prywatne. W rezultacie, należało radzić sobie ze zmiennością zespołu i adaptować się do nowych warunków pracy.

2. Brak doświadczenia w fazie analizy - niedoszacowanie pewnych aspektów projektu spowodowały trudności w późniejszych etapach implementacji. Ze względu na niewielkie doświadczenie w analizie projektowej konieczne było wprowadzanie wielu poprawek w trakcie trwania projektu, co opóźniało i utrudniało wdrożenie systemu.

3. Opóźnienie w podjęciu radykalnych kroków w przypadku aplikacji Android - Na początku projektu zespół nie miał doświadczenia w tworzeniu aplikacji mobilnych ani w języku Kotlin. To spowodowało opóźnienie we wdrożeniu aplikacji mobilnej, ponieważ należało poświęcić czas na naukę języka i programowania mobilnego od podstaw. Dodatkowo pierwotny skład zespołu do implementacji aplikacji mobilnej został początkowo zastąpiony tylko jednym studentem, w kolejnym semestrze doszła jeszcze jedna osoba. Zbyt długo zwlekano z podjęciem niezbędnych działań w celu naprawy problemów związanych z aplikacją na platformie Android. To spowodowało opóźnienia w rozwoju i wprowadzało trudności w dostarczaniu działającej wersji dla użytkowników. Jednak dzięki dużej pracy włożonej przez zespół w dodatkowe szkolenia, udało się nadrobić braki w wiedzy i umiejętności w krótkim czasie.

4. Trudności z implementacją kalendarza: Okazało się, że implementacja funkcjonalności kalendarza była bardziej skomplikowana, niż początkowo zakładano. To wymagało dodatkowego czasu i wysiłku ze strony zespołu implementacji.

5. Opóźnienie w rozpoczęciu procesu implementacji: Zbyt długo oczekiwano z rozpoczęciem procesu implementacji niektórych funkcjonalności, które mogłyby być zrealizowane już na etapie planowania projektu. To prowadziło do opóźnień w dostarczaniu funkcji użytkownikom.

6. Brak regularnych spotkań: Brak regularnych spotkań w początkowych fazach projektu wpływał negatywnie na organizację pracy. Dopiero późniejsze wprowadzenie regularnych zebrań pomogło w poprawie organizacji pracy.

7. Niedoszacowanie czasu trwania zadań: Szacowanie czasu trwania większości zadań okazało się błędne, co prowadziło do opóźnień.

8.5. Planowane ulepszenia systemu

Możliwe ulepszenia systemu:

1. Przejście na architekturę single tenant: Przebudowa systemu w taki sposób by umożliwić korzystanie z niego przez pojedyncze podmioty, co zapewniłoby większą niezależność i bezpieczeństwo danych.

2. Migracja do chmury: Przeniesienie infrastruktury systemu do chmury obliczeniowej, co pozwoliłoby na elastyczne skalowanie zasobów, lepszą dostępność oraz obniżenie kosztów utrzymania.

3. Dynamiczne powoływanie środowisk: Wprowadzenie mechanizmu umożliwiającego tworzenia i usuwania środowisk w sposób automatyczny.

4. Przepisanie części serwerowej na mikroserwisy: Podzielenie aplikacji na mniejsze serwisy ułatwiłoby skalowanie horyzontalne, zapewniłoby większą niezawodność oraz umożliwiłoby szybszy rozwój poszczególnych funkcjonalności.

5. Rozwinięcie systemu o supply chain management (SCM): Dodanie modułu SCM, który umożliwiłby optymalizację procesów logistycznych.

6. Modularne podejście i rozwinięcie w kierunku systemu ERP: Rozważenie bardziej modularnego podejścia do systemu, umożliwiającego dodawanie kolejnych modułów, takich jak CRM, zarządzanie projektami, księgowość, itp., pozwoliłoby na kompleksowe zarządzanie firmą.

7. Tłumaczenie interfejsu na język angielski: Umieszczenie opcji wielojęzyczności w systemie zwiększyłoby liczbę potencjalnych klientów zagranicznych.

8. Wprowadzenie systemu monitoringu i przechowywania logów: Dodanie narzędzi do monitorowania działania systemu oraz przechowywania logów, takich jak Elasticsearch, Logstash czy Kibana (ELK), umożliwiłoby lepszą analizę ewentualnych problemów.

9. Udoskonalenie interfejsu użytkownika: Przeprojektowanie interfejsu użytkownika w celu poprawy użyteczności, przyczyniłoby się do łatwiejszej obsługi systemu.

10. Rozważenie integracji z innymi systemami: Analiza możliwości integracji systemu z innymi istniejącymi narzędziami, które mogłyby przynieść dodatkowe korzyści dla firm monterskich.

9. Bibliografia

[1] Aplikacja MonterGO by MonterGO.com. https://www.montergo.com/wpcontent/uploads/2022/02/image002.png. Dostęp: 2023-04-25
[2] Aplikacja MonterGO by MonterGO.com. https://www.montergo.com/. Dostęp: 2023-04-25
[3] Aplikacja Field Service Dispatch by Verizon Connect. https://play.google.com/store/apps/details?id=com.verizonconnect.fsdapp. Dostęp: 2023-04-28
[4] Aplikacja Zoho by Zoho Corporation. https://www.zoho.com/inventory/. Dostęp: 2023-04-26
[5] velog io: RESTful API. https://velog.io/@0andwild/RESTful-API%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80. Dostęp: 2023-26-04
[6] Aplikacja Figma by Figma Inc. http://www.figma.com/. Dostęp: 2023-23-05
[7] W3.CSS: React Class Components. https://www.w3schools.com/react/react_class.asp. Dostęp: 2023-02-04
[8] Meta Platforms, Inc. https://legacy.reactjs.org/docs/components-and-props.h . Dostęp: 2023-02-04
[9] Monesi Daniele: TypeScript vs. JavaScript: Your Go-to Guide. https://www.toptal.com/typescript/typescript-vs-javascript-guide. Dostęp: 2023-05-05.
[10] Aplikacja Service Titan by SerivceTitan. https://www.servicetitan.com/features/dispatch-software . Dostęp: 2023-01-04.
[11] Aplikacja Service Titan by SerivceTitan. https://www.servicetitan.com/features/field-service-app. Dostęp: 2023-01-04.
[12] Sullivan Megan: ServiceTitan Forms Unified SEO and Content Strategy to Nurture Leads. https://www.newbreedrevenue.com/blog/inbound-growth-story-servicetitan-024. Dostęp: 2023-01-04.
[13] DiscoverCloud: HouseCall Pro by HouseCall Pro. https://www.discovercloud.com/products/housecall-pro#/alternatives. Dostęp: 2023-06-23
[14] Aplikacja HouseCall Pro by Codefied Inc. https://www.youtube.com/watch?v=LgotBzpC1hs. Dostęp: 2023-06-23.
[15] Endeavor Business Media: Housecall Pro Launches New XL Plan https://www.contractormag.com/technology/software/article/20884211/housecall-pro-launches-new-xl-plan. Dostęp: 2023-06-23.
[16] Aplikacja HouseCall Pro by Codefied Inc. https://www.housecallpro.com/. Dostęp: 2023-06-23
[17] Oracle: Oracle Certified System Configurations - Java SE 6. https://www.oracle.com/java/technologies/system-configurations.html. Dostęp: 2023-03-13
[18] Tomasz Kozon, Boring Owl: Spring. https://boringowl.io/tag/spring. Dostęp: 2023-04-14.
[19] Bykowski.pl: Swagger UI – przejrzysta wizualizacja zasobów API. https://bykowski.pl/swagger-ui-przejrzysta-wizualizacja-zasobow-api/. Dostęp: 2023-03-04.
[20] Tomasz Kozon, Boring Owl: Swagger UI – przejrzysta wizualizacja zasobów API. https://boringowl.io/blog/hibernate. Dostęp: 2023-26-04.
[21] Kamil Klimek: Lombok – Czyli Generowanie Kodu W Javie. https://1024kb.pl/nauka-java/lombok-czyli-generowanie-kodu-java/ . Dostęp: 2023-26-04.
[22] Dominik Szczepaniak: Wprowadzenie do REST API. https://devszczepaniak.pl/wprowadzenie-do-rest-api/. Dostęp: 2023-03-15.
[23] Narzędzie Junit by The JUnit Team. https://junit.org/. Dostęp: 2023-04-04.
[24] O'Reilly/Helion, David Griffiths: Kotlin. Rusz głową! Dostęp: 2023-03-24.
[25] Baeldung: A Guide to REST-assured. https://www.baeldung.com/rest-assured-tutorial. Dostęp: 03-04-2023

[26] Jakub Świątkowski: Co to jest Postman, do czego służy i jakie są jego funkcjonalności? https://www.droptica.pl/blog/co-jest-postman-do-czego-sluzy-i-jakie-sa-jego-funkcjonalnosci/. Dostęp: 2023-05-04.
[27] Tomasz Kozon, Boring Owl: Selenium. https://boringowl.io/tag/selenium. Dostęp: 2023-02-03.
[28] Tomasz Kozon, Boring Owl: Dlaczego twoja aplikacja powinna być napisana w React.js? https://boringowl.io/blog/dlaczego-twoja-aplikacja-powinna-byc-napisana-w-reactjs. Dostęp: 2023-01-06.
[29] Technologia TypeScript by Microsoft. https://www.typescriptlang.org/docs. Dostęp: 2023-01-06.
[30] Tomasz Kozon, Boring Owl: Figma. https://boringowl.io/tag/figma. Dostęp: 2023-02-05
[31] Fly On The Cloud sp. z o.o.: Draw.io – diagram online krok po kroku https://fotc.com/pl/blog/drawio-diagram-online-instrukcja/. Dostęp: 2023-02-04.
[32] Martin Fowler: GUI Architecture. https://martinfowler.com/eaaDev/uiArchs.html. Dostęp: 2023-04-06.
[33] DigitalOcean, LLC: SOLID: The First 5 Principles of Object Oriented Design. https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. Dostęp: 2023-04-06.
[34] Aplikacja ClickUp by ClickUp . https://clickup.com/?noRedirect=true. Dostęp: 2023-01-06.
[35] Baeldung: The DTO Pattern (Data Transfer Object). https://www.baeldung.com/java-dto-pattern. Dostęp: 2023-04-06.
[36] Java Development Jounral: Rysunek Android MVVM. https://journaldev.nyc3.digitaloceanspaces.com/2018/04/android-mvvm-pattern.png. Dostęp: 2023-04-06.
[37] Microsoft: Model-View-ViewModel (MVVM). https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. Dostęp: 2023-06-06.
[38] Koin & Kotzilla: Injecting in Android. https://insert-koin.io/docs/reference/koin-android/get-instances. Dostęp: 2023-06-06.
[39] Technologia Retrofit by Square. https://square.github.io/retrofit/. Dostęp: 2023-04-06.
[40] Technologia Spring Security by VMware, Inc. https://spring.io/projects/spring-security. Dostęp: 2023-04-06.
[41] Baeldung: JPA Criteria Queries. https://www.baeldung.com/hibernate-criteria-queries. Dostęp: 2023-01-06.
[42] Baeldung: Introduction to Spring Data JPA. https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa. Dostęp: 2023-06-06.
[43] Oracle: Co to jest Docker?. https://www.oracle.com/pl/cloud/cloud-native/container-registry/what-is-docker/. Dostęp: 2023-06-06.
[44] C# Corner: Docker Architecture - Environment - Advantages. https://www.c-sharpcorner.com/article/docker-architecture-environtment-advantages/. Dostęp: 2023-06-06.
[45] Technologia Kubernetes by The Linux Foundation. https://kubernetes.io/pl/. Dostęp: 2023-06-06.
[46] Technologia Traefik by Traefik Labs. https://doc.traefik.io/traefik/. Dostęp: 2023-06-06.
[47] Technologia Github Actions by GitHub, Inc. https://github.com/features/actions. Dostęp: 2023-06-06.
[48] Biblioteka mui by Material UI SAS. https://mui.com/. Dostęp: 2023-03-06.
[49] Biblioteka emotion by Emotion. https://emotion.sh/docs/introduction. Dostęp: 2023-03-06.
[50] Biblioteka acios by John Jakob "Jake" Sarjeant. https://axios-http.com/. Dostęp: 2023-05-06.
[51] Bibliotek formic by Formium, Inc. https://formik.org/. Dostęp: 2023-05-06.
[52] Biblioteka leafletjs by Volodymyr Agafonkin. https://leafletjs.com/. Dostęp: 2023-05-06.
[53] Biblioteka React-qr-code by Npm. https://www.npmjs.com/package/react-qr-code. Dostęp: 2023-05-06.

[54] Biblioteka jwt by Auth0. https://jwt.io/. Dostęp: 2023-05-06.
[55] Krzysztof Baranowski: Wzorzec MVVM w 5 minut. https://programistabyc.pl/mvvm-1-koncepcja/. Dostęp: 2023-05-06.
[56] Biblioteka koin by Koin & Kotzilla. https://insert-koin.io/. Dostęp: 2023-06-06.
[57] Biblioteka react router by Remix Software, Inc. https://reactrouter.com/en/main. Dostęp: 2023-06-06.

10. Spis rysunków

11. Spis tabel

12. Spis listingów