
POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

**Wydział Informatyki**

**Katedra Inżynierii Oprogramowania**

Inżynieria Oprogramowania i Baz Danych

**Robert Radziszewski**
Nr albumu S23993

# Uniwersalny system wspomagający pracę architektów

Praca magisterska pod kierunkiem:
Dr Mariusz Trzaska

PJATK Warszawa, Luty, 2023

## Streszczenie

Problem przedstawiony w pracy dotyczy aplikacji wspomagających pracę architektów wnętrz. Obecnie dostępne rozwiązania pozwalają na tworzenie prostych list zakupowych produktów, które są wymagane do wdrożenia w życie pomysłów zawartych w projekcie pomieszczeń/budynków. W przypadku jednej z aplikacji, twórcy zaoferowali również wtyczkę do przeglądarek internetowych, która pozwala pobrać dane o produkcie z jego strony, używając prostych znaczników HTML. Programy te przyczyniły się do zwiększenia komfortu pracy, jednak stwierdzono, iż zawierają kilka istotnych wad które należy zredukować, by zmaksymalizować stopień wykorzystania ich potencjału. Jednym z pomysłów jest zwiększenie użyteczności poprzez zastosowanie zestawów atrybutów produktów które mogą być łączone z projektami, a także posiadać mechanizm autouzupełniania wartości, z wykorzystaniem znaczników standardu *Schema.org* oraz zdefiniowanych przez użytkowników. Kolejnym usprawnieniem, zaoferowanym w propozycji takiej aplikacji, jest możliwość uzyskania zarobku przez projektantów, poprzez uczestnictwo w zintegrowanej sieci afiliacyjnej i uzyskiwanie prowizji od produktów zamieszczanych, a także polecanych w projektach. Wychodząc naprzeciw oczekiwaniom potencjalnych klientów, zastosowano także usprawnienia interfejsu użytkownika oraz sposobu jego obsługi, co pozwala zaoszczędzić także czas.

Prototyp powstał w oparciu o aktualne standardy i techniki programowania. W skład zaproponowanego rozwiązania wchodzą: serwerowa aplikacja internetowa, wtyczka na przeglądarkę Chrome, oraz dedykowana na platformę Android. Część *backendowa* została zbudowana z wykorzystaniem dwóch kontenerów Docker, czyli bazy danych PostgreSQL i aplikacji Spring w języku Kotlin. Warstwę interfejs użytkownika zaprojektowano w oparciu o *framework* Vue. Aplikacja mobilna powstała przy użyciu Android Studio, również w języku Kotlin. Wtyczka przeglądarki Chrome jest kompatybilna ze standardem Manifest. Solucja, będąca efektem tej pracy, powstała jako narzędzie modułowe, co pozwala zwiększyć zakres funkcjonalności w przyszłej perspektywie.



## Podziękowania

Autor pracy wyraża podziękowanie promotorowi pracy dr Mariuszowi Trzaska za cenne wsparcie podczas tworzenia koncepcji i realizacji prototypu, a zwłaszcza za udzielanie celnych uwag na temat funkcjonalności prototypu. Podziękowania zostają złożone również Wydziałowi Informatyki za przekazaną wiedzę w czasie toku studiów.

# Spis treści


**1. WSTĘP ... 5**

1.1. Cel pracy ... 5
1.2. Organizacja pracy ... 5
1.3. Rozwiązanie przyjęte w pracy ... 5
1.4. Rezultaty pracy ... 6
1.5. Motywacja tematu pracy ... 6

**2. STAN SZTUKI I ANALIZA ISTNIEJĄCYCH ROZWIĄZAŃ ... 8**

2.1. Istniejące rozwiązania ... 8
*2.1.1 KIS List ... 8*
*2.1.2 Listonic ... 10*
*2.1.3 Formly ... 11*
*2.1.4 Webepartners ... 14*
2.2. Wymagania wobec prototypu ... 16
*2.2.1 Wnioski z analizy istniejących rozwiązań ... 16*
*2.2.2 Wymagania wobec propozycji rozwiązania ... 17*
*2.2.3 Propozycja rozwiązania ... 17*

**3. WSPÓŁPRACA AFILIACYJNA ... 20**

3.1. Sieci afiliacyjne ... 20
3.2. Proces afiliacyjny ... 21
3.3. Miary afiliacyjne ... 22

**4. WYKORZYSTANE NARZĘDZIA I TECHNOLOGIE ... 23**

4.1. Schema.org ... 23
4.2. Architektura MVC ... 25
4.3. REST ... 28
4.4. Kotlin ... 30
4.5. Spring MVC ... 31
4.6. HTML ... 33
4.7. Javascript ... 35
4.8. VUE ... 36
4.9. PostgreSQL ... 38
4.10. Intelij IDEA ... 39
4.11. Android Studio ... 40

**5. SPECYFIKACJA WYMAGAŃ ... 42**

5.1. Zarys ... 42
5.2. Aktorzy biznesowi ... 43
5.3. Wymagania projektowe – funkcjonalne ... 43
5.4. Wymagania projektowe – niefunkcjonalne ... 45
5.5. Przypadki użycia – diagram ... 46
*5.5.1 Sprzedawca ... 46*
*5.5.2 Projektant ... 47*
*5.5.3 Klient ... 48*
*5.5.4 Administrator techniczny ... 49*
5.6. Specyfikacja niektórych przypadków użycia ... 50
*5.6.1 Moduł logowania i autoryzacji ... 50*
*5.6.2 Moduł kreatora list produktów ... 51*
*5.6.3 Moduł afiliacji i integracji ... 53*
*5.6.4 Moduł atrybutów ... 54*
5.7. Specyfikacja wybranych procesów biznesowych ... 56
*5.7.1 Proces dodawania produktu z wykorzystaniem wtyczki ... 56*
*5.7.2 Proces zakupu produktu z wykorzystaniem linku afiliacyjnego ... 57*

5.8. Architektura systemu ... 58
*5.8.1 Diagram komponentów ... 58*
*5.8.2 Diagram pakietów ... 59*
*5.8.3 Diagram ERD ... 63*

**6. IMPLEMENTACJA PROTOTYPU ... 64**

6.1. Architektura aplikacji – implementacja ... 64
6.2. Konfiguracja bazy danych ... 64
6.3. Struktura bazy danych ... 65
*6.3.1 Dane o użytkownikach ... 65*
*6.3.2 Projekty ... 68*
*6.3.3 Programy partnerskie i afiliacje ... 72*
6.4. API ... 73
6.5. Wybrane aspekty implementacji aplikacji internetowej ... 80
6.6. Wybrane aspekty implementacji wtyczki do przeglądarki Chrome ... 90
6.7. Wybrane aspekty implementacji aplikacji na system Android ... 94

**7. PREZENTACJA APLIKACJI ... 97**

7.1. Moduł logowania i autoryzacji ... 97
7.2. Moduł atrybutów ... 99
7.3. Moduł projektów ... 101
7.4. Dodawanie nowych produktów do projektu ... 104
7.5. Moduł afiliacji ... 110
7.6. Aplikacja mobilna ... 114

**8. ANALIZA REZULTATÓW ... 117**

8.1. Wady i zalety ... 117
8.2. Kierunki rozwoju aplikacji ... 118
8.3. Podsumowanie ... 119

**BIBLIOGRAFIA ... 120**

**SPIS RYSUNKÓW ... 123**

**SPIS TABEL ... 125**

**SPIS LISTINGÓW ... 126**

# 1. Wstęp

Postępujący rozwój komputeryzacji spowodował powstanie różnego rodzaju rozwiązań i aplikacji, które wspomagają pracę w różnych dziedzinach życia. Podobny trend został zaobserwowany w branży projektantów wnętrz i *homestagingu*, który zaowocował pojawieniem się oprogramowania ułatwiającego tworzenie projektów rozumianych jako wizualizacje i listy zakupowe. Obecne rozwiązania są relatywnie popularne, ale cechują się szeregiem niedoskonałości, jak brak integracji pomiędzy różnymi funkcjami i aspektami tej dziedziny, co szerzej zostało przedstawione w rozdziale 2. Celem poniższej pracy jest opracowanie koncepcji, a także pośrednio uzyskanie prototypu który ma pomóc rozwiązać problem przedstawiony w następnych rozdziałach, a także ułatwić współpracę z tego typu oprogramowaniem oraz wprowadzić możliwość uzyskania zarobku za promowanie produktów – jako forma sieci afiliacyjnej.

## 1.1. Cel pracy

Celem postawionym przed tą pracą jest analiza tematyki problemu, obecnie wykorzystywanych rozwiązań, a także przedstawienie propozycji wprowadzenia usprawnień wraz z implementacją prototypu. Ważnym elementem jest analiza uzyskanych rezultatów pod kątem spełnienia postawionych wymagań.

## 1.2. Organizacja pracy

Praca została podzielona na rozdziały, które przedstawiają cały proces przygotowania i wdrażania koncepcji prototypu z podziałem na rozdziały, podrozdziały odwzorowujące kolejne etapy tego procesu. W pierwszym rozdziale zamieszczono krótki opis problemu oraz przedstawiono oczekiwane rezultaty. W drugim – analizę istniejących rozwiązań, pomysłów co można usprawnić, a także motyw podjętej pracy. W trzecim i czwartym – opisano elementy wiedzy, które są wymagane, aby uzyskać zamierzone cele. Pierwszy z nich przedstawia opis zagadnień merytorycznych, zaś drugi – informacje na temat użytych technologii, standardów oraz wzorców architekturalnych. W piątym rozdziale umieszczono projekt aplikacji. W kolejnym – objaśniono proces implementacji rozwiązania. Ostatni rozdział przedstawia uzyskane rezultaty, wraz z opisem poszczególnych ekranów interfejsu użytkownika. Całość zakończono podsumowaniem, które dotyka wniosków z tego w jakim stopniu udało się osiągnąć cele, a także możliwych kierunków rozwoju.

## 1.3. Rozwiązanie przyjęte w pracy

W celu implementacji rozwiązania przyjęto aktualne standardy i dobre praktyki dotyczące poniższych technologii:

- Tworzenie aplikacji w architekturze MVC;
- Programowanie obiektowe, a także funkcyjne;
- Komunikacja za pomocą protokołu HTTP z wykorzystaniem REST;
- Administrowanie bazami danych (DML, DQL, DDL);
- Rozwijanie aplikacji mobilnych.

Praca wykorzystuje elementy teoretyczne następujących dziedzin:

- Sieci afiliacyjne;
- Metody przetwarzania danych zawartych na stronach internetowych za pomocą webscrapingu;
- Modelowanie struktury systemu za pomocą notacji UML;
- Wzorce architekturalne MVC, MVVM oraz Event Driven;
- Projekt bazy danych w minimum 3 postaci normalnej.

W celu opracowania prototypu przyjęto poniższy zbiór aplikacji:

**A. Aplikacja internetowa**

Aplikacja internetowa odpowiada za przechowywanie danych i realizacji większości zamierzonych funkcjonalności w sposób bezpieczny. Nie wymaga konieczności instalacji na komputerze użytkownika. Z aplikacją internetową pracują obie wersje wtyczki, jest ona sercem systemu.

**B. Wtyczka na przeglądarkę Chrome**

Pełni ona rolę kolektora produktów, wykorzystując techniki ekstrakcji danych internetowych.

**C. Aplikacja mobilna na system Android**

Co do zasady, działa analogicznie do wtyczki powyżej, jednak dostarcza rozwiązanie na systemy mobilne Android, gdzie nie ma możliwości instalacji rozszerzeń wprost do przeglądarki Chrome Mobile.

## 1.4. Rezultaty pracy

Lista pożądanych efektów pracy została określona w formie poniższych punktów:

- Przedstawienie problemu i analizy istniejących rozwiązań, przeprowadzanie specyfikacji wymagań postawionych wobec prototypu;
- Opracowanie koncepcji i projektu rozwiązania, które spełnią wymagania, jako efekt bezpośredni;
- Efektem pośrednim jest wykonanie prototypu z wykorzystaniem aktualnych standardów i prawidłowych praktyk programistycznych.

## 1.5. Motywacja tematu pracy

Temat ten wynika nie tylko z zainteresowań ogólną konstrukcją systemów informatycznych, ale także analizy potencjału zarobkowego takiego rozwiązania. Temat jest syntezą oczekiwań projektantów wnętrz (bezpośrednia znajomość osób które się tym zajmują), architektów oraz ich klientów.

Prototyp dzięki zastosowaniu odpowiedniego poziomu generyczności, może nie tylko usprawniać pracę osób zajmujących się wnętrzami, ale także potencjalnie wspomóc pracowników i ich klientów , biorąc pod uwagę także inne dziedziny życia. Temat jest nieznacznie zbliżony do tematyki przerabianej

w czasie pracy zawodowej autora, w której podejmuje się on tworzenia, a także rozwijania systemów wspierających pracę publicznej instytucji prowadzących działalność edukacyjną.

# 2. Stan sztuki i analiza istniejących rozwiązań

W tym rozdziale przedstawione zostały działające na rynku rozwiązania, które są ważne pod kątem wymogów postawionych przed opracowywaną aplikacją, w ramach pracy magisterskiej. Opisane zostały ich funkcjonalności oraz wady lub istotne niedoskonałości dotyczące ich użyteczności.

Poniżej przedstawione rozwiązania, odnoszące się zarówno do sposobu tworzenia listy, produktów, kosztorysów, jak również zarządzania procesem afiliacyjnym, stanowią punkt wyjściowy do opracowania rozwiązania przedstawionego w późniejszym etapie.

## 2.1. Istniejące rozwiązania

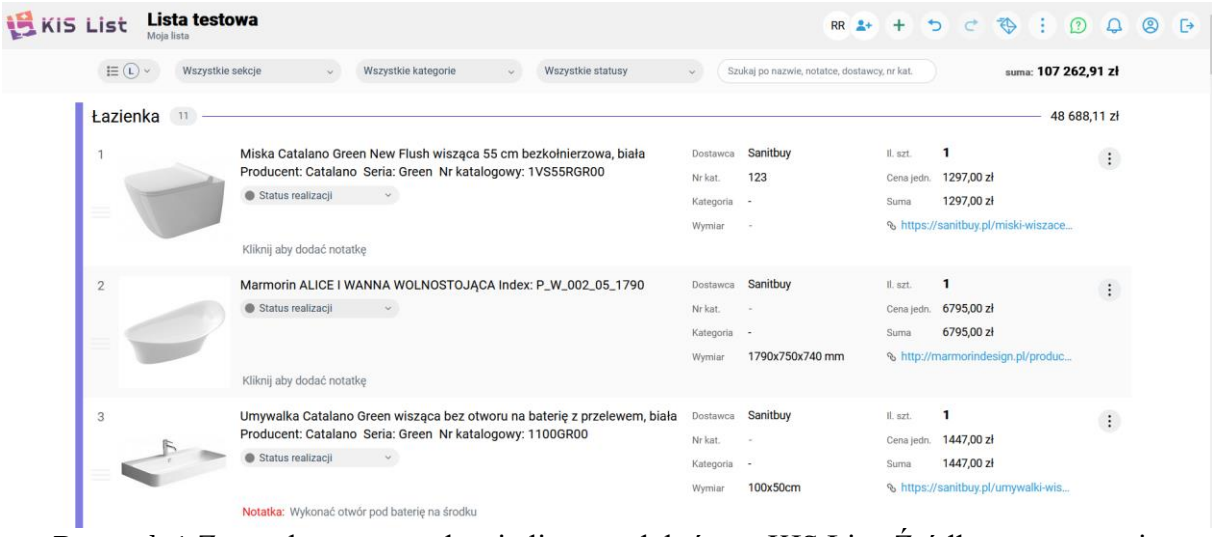
W poniższych podrozdziałach dokonano analizy rozwiązań i programów komputerowych obecnie funkcjonujących na rynku. Poszczególne punkty zawierają krótkie przedstawienie funkcjonalności, zrzuty ekranu prezentującymi graficzny interfejs użytkownika, a także spis wad oraz zalet względem konkurencyjnych aplikacji.

### 2.1.1 KIS List

KIS List [2] jest prostym narzędziem, wykorzystywanym do zapisywania informacji o przeglądanych produktach za pomocą udostępnionej wtyczki do przeglądarek Chrome, Firefox i Safari. Funkcjonalność omawianej aplikacji składa się z następujących punktów:

- Tworzenie i zarządzanie listami produktów z podziałem na sekcje. Przykład listy przedstawiono na rysunku nr 1;
- Dodawanie i edycja produktów z wykorzystaniem wtyczki do przeglądarek WEB. Pozwala ona przechwycić ze strony internetowej wskazane informacje o produkcie:
  - Grafikę produktu;
  - Cenę;
  - Nazwę;
  - Link do produktu.

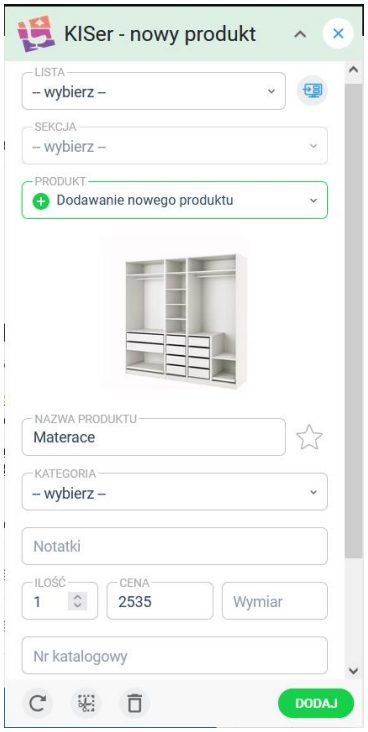
  Udostępnia również dodatkowe pola jak notatki do ręcznego uzupełnienia przez użytkownika. Informacje o produktach pobierane są ze strony internetowej na podstawie uprzednio przygotowanych danych, w formie znaczników <meta> lub struktury json-ld w standardzie schema.org [20]. Standard schema.org jest obecnie spełniany przez większość nowoczesnych stron internetowych w języku polskim, a także angielskim w stopniu całkowitym lub częściowym, co oznacza, że może dojść do sytuacji braku wartości niektórych pól. Interfejs użytkownika tej wtyczki pokazano na rysunku nr 2.




Rysunek 1 Zrzut ekranu zarządzania listą produktów w KIS List. Źródło: opracowanie własne.

Do zalet możemy zaliczyć intuicyjność oraz prostotę jej obsługi, a także szybkość przetwarzania danych, jednak w kontekście pracy można wyróżnić kilka wad:

- Brak wsparcia przeglądarek na systemy mobilne;
- Brak integracji z sieciami afiliacyjnymi;
- Meta tagi produktów są parsowane w ograniczonym zakresie – nie możliwy jest automatyczny odczyt innych cech produktów niż wymienione powyżej;
- Lista nie udostępnia możliwości zdefiniowania dodatkowych pól określonych przez użytkownika aplikacji;

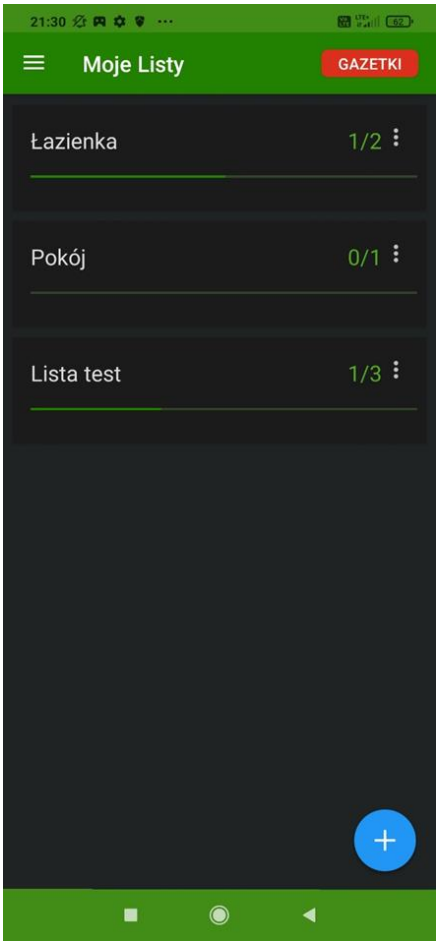



Rysunek 2 Interfejs użytkownika wtyczki do dodawania/edycji produktów. Źródło: opracowanie własne

### 2.1.2 Listonic

Jednym z rozwiązań do sporządzania listy produktów jest Listonic [3]. Jest to aplikacja przeznaczona dla użytkowników systemu Android. Zastępuje ona konwencjonalną listę zakupową – pozwala utworzyć elektroniczną *checklistę* produktów do zakupu, co ukazano na rysunku nr 3.

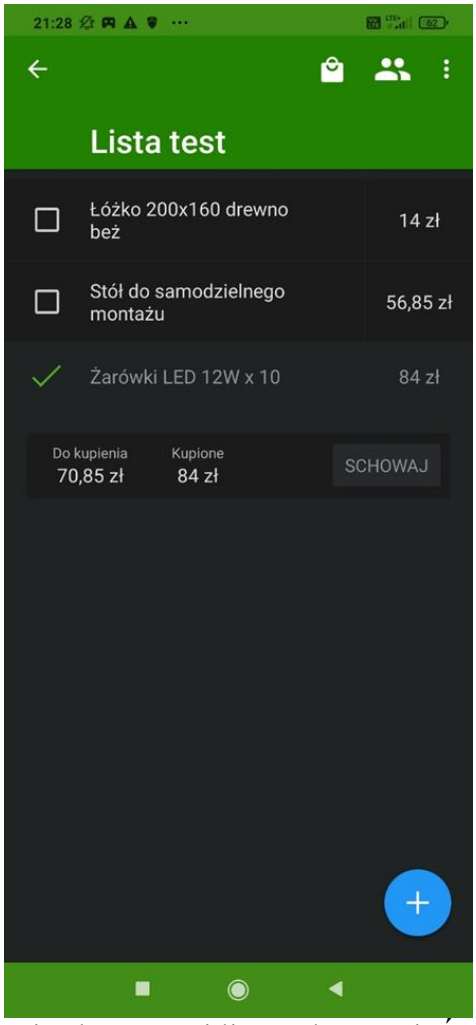
Wspiera ona synchronizację pomiędzy wieloma urządzeniami, której przebiegiem zarządza serwis WWW. Pozwala ona na współdzielenie list zakupowych pomiędzy użytkownikami, którymi mogą być np. członkowie rodziny. W kontekście wparcia pracy projektantów, jej ważną cechą jest możliwość podzielenia list na kategorie, a także możliwość sprawdzenia statystyk odnośnie wybieranych produktów, czy też podpowiedzi na podstawie wprowadzonych już produktów (rysunek nr 4). W skład funkcjonalności wchodzi możliwość oznaczenia elementów listy jako zrealizowane. [3]




Rysunek 3 Ekran wyboru oraz definiowania listy zakupowej. Źródło: opracowanie własne

Jako aplikacja na systemy mobilne, Listonic stał się wzorem solucji na platformę Android. Aplikacja opracowana na potrzeby implementacji koncepcji, jest mobilnym odpowiednikiem wtyczki do przeglądarek Chrome. Wersja *Mobile* nie wspiera mechanizmu wtyczek i rozszerzeń. Podobnie jak w przypadku rozwiązania z punktu 2.1.1, występuje tu kilka istotnych obszarów, które należy przeprojektować, by poprawić efektywność pracy projektantów i wpłynąć na użyteczność aplikacji. Są to m. in.:

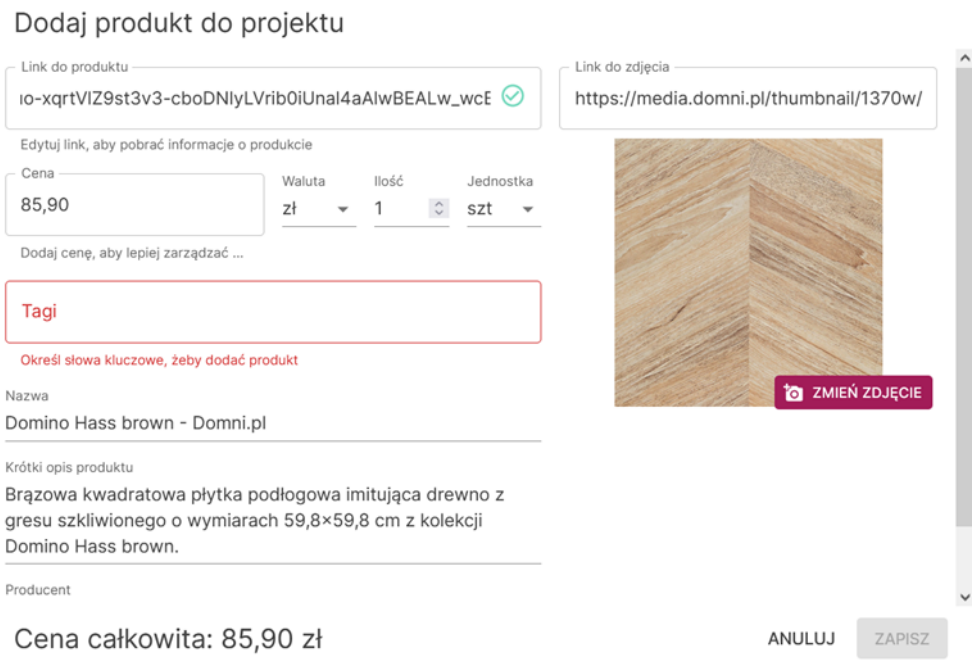
- Brak atrybutów produktu. Dany element jest zdefiniowany jako wpis z notatką oraz ceną, co może być problematyczne w utworzeniu oferty/wyceny przez projektanta;
- Aplikacja nie udostępnia mechanizmu automatycznego odczytywania cech produktu.




Rysunek 4 Ekran zarządzania elementami listy zakupowej. Źródło: opracowanie własne

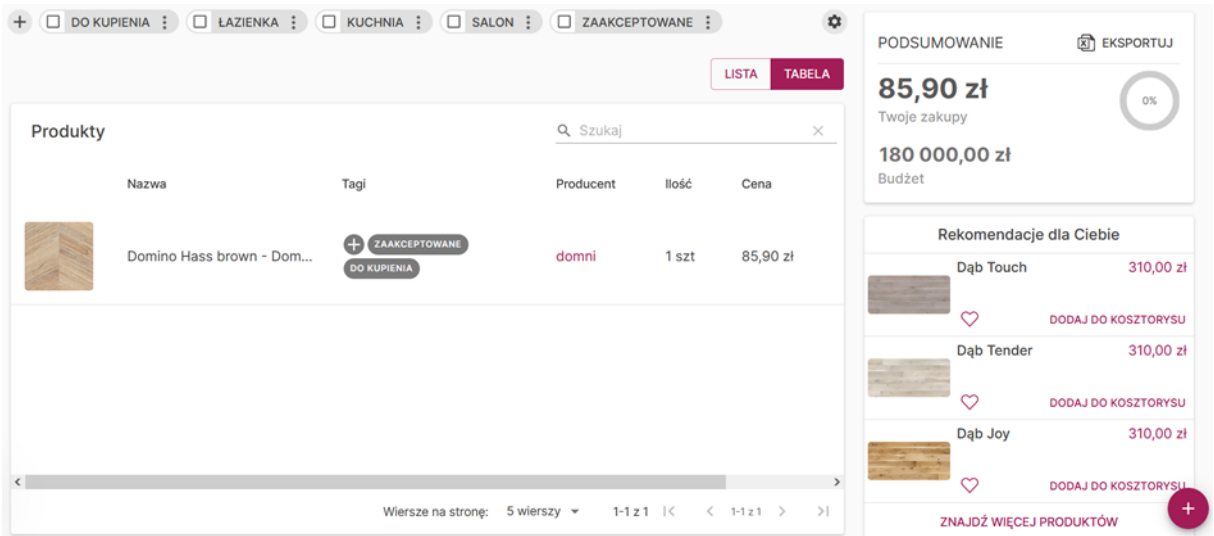
### 2.1.3 Formly

Aplikacja Formly [4] została opracowana w celu tworzenia i zarządzania projektami na potrzeby architektów wnętrz oraz *homestage-rów*. Jej głównymi obszarami działania są tworzenie list produktów, a także wizualizacji będących zbiorem plików zdjęciowych. Podobnie jak w przypadku KIS List, każdy element kolekcji produktów (przedstawiony na rysunku nr 5) zawiera grafikę, cenę, odnośnik do jego internetowej strony sklepowej, nazwę, ilość z możliwością przypinania do niego tagów.



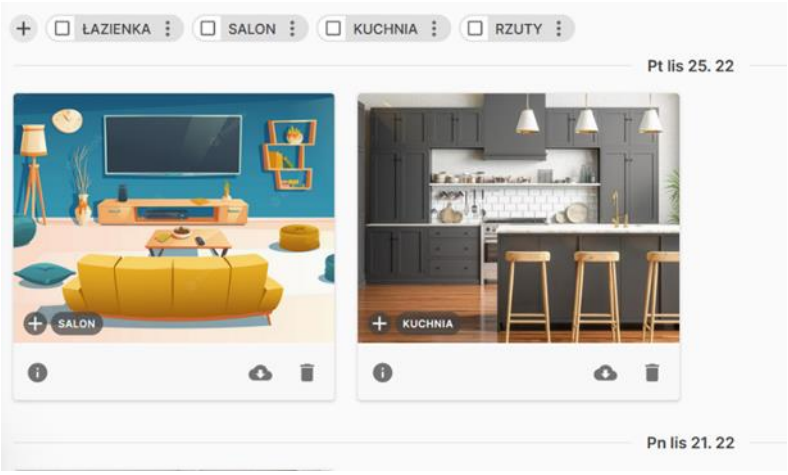

Rysunek 5 Ekran edycji danych produktu w aplikacji Formly. Źródło: opracowanie własne

Twórcy programu umożliwiają przypinania tagów w celu ustrukturyzowania produktów, jednak w odróżnieniu od rozwiązania w KIS List, brakuje tu możliwości definiowania sekcji jako jednostki ich podziału funkcjonalnego. Aplikacja wyznacza, a następnie przedstawia sumę ceny produktów w projekcie, a także umożliwia eksport listy produktów do pliku tekstowego rozdzielonego średnikiem (*csv*), co zaprezentowano na rysunku nr 6. [4]

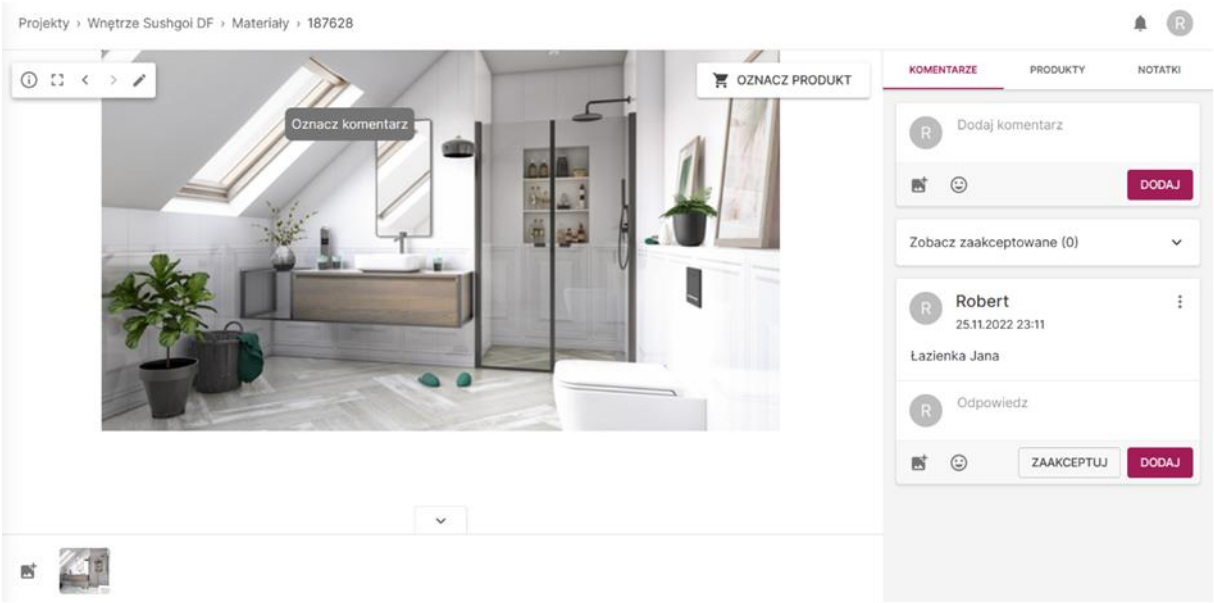



Rysunek 6 Fragment aplikacji Formly podsumowujący projekt. Źródło: opracowanie własne

Jedną z funkcji serwisu jest także możliwość dodawania wizualizacji zwanych materiałami (Rysunek nr 7). Dostarczane są one w formie plików graficznych standardowych formatów *.jpg, .png, .tiff, .svg*. Analogicznie do spisu produktów, wizualizacje mogą być oznaczane mechanizmem tagów. Opcjonalnie istnieje możliwość dodania komentarzy do materiałów (Rysunek nr 8). [4]




Rysunek 7 Zrzut okna przedstawiający listę ostatnio dodanych materiałów. Źródło: opracowanie własne




Rysunek 8 Zrzut ekranu przedstawiający zakładkę "materiały". Źródło: opracowanie własne

Zaletą, względem poprzednich rozwiązań, jest także możliwość umieszczenia komentarzy przez innych użytkowników, którzy posiadają wymagane uprawnienia do projektu, możliwość pozostawienia notatki czy podpięcia produktów z projektu do wizualizacji.

Biorąc pod uwagę oczekiwania użytkowników, aplikacja nie jest wolna od wad, które przedstawiają się następująco:

- Brak wsparcia przeglądarek na systemy mobilne, nieintuicyjny mechanizm pobierania danych ze strony, który umożliwiałby wybranie innej treści niż domyślnie wskazana przez schema.org oraz microdata;
- Brak integracji z sieciami afiliacyjnymi;
- Meta tagi produktów są parsowane w ograniczonym zakresie – automatyczny odczyt innych cech produktów, niż wymienione powyżej, nie jest możliwy;

- Strona nie oferuje możliwości zdefiniowania pól dodatkowych, określanych przez użytkownika aplikacji.

### 2.1.4 Webepartners

Rozwiązanie przygotowane przez twórców tej aplikacji internetowej jest jedną z tzw. sieci afiliacyjnych. Uczestniczy w niej 3 głównych aktorów: reklamodawca, wydawca, a także administrator techniczny systemu. [5] [6]

**Reklamodawca**

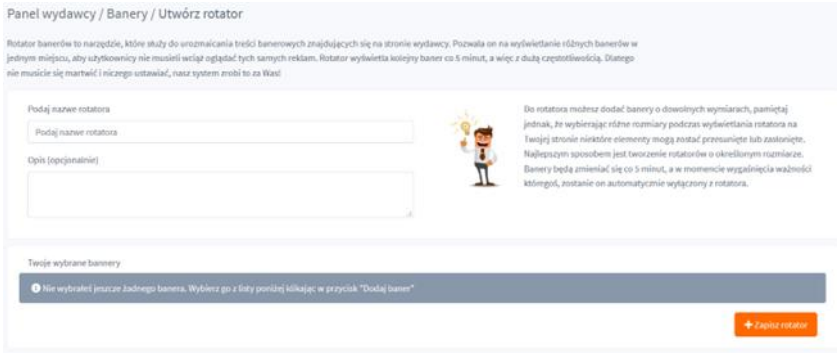
Użytkownik ten [5] uruchamiając program partnerski, zobowiązuje się wypłacać określoną przez siebie prowizję od sprzedaży (tylko i wyłącznie powiedzionej sukcesem) produktów poleconych przez wydawców (*Cost Per Sale*). Może on także ustalać stawkę za poprawnie zrealizowaną akcję (*Cost Per Lead*). Określa on profil firmy, oferowane produkty poprzez narzędzia udostępnione przez twórców:

- *Deep Link* – Pozwala utworzyć on odnośnik do produktu podpięty do dowolnej grafiki, tekstu z możliwością umieszczenia na stronie internetowej, udostępnienia poprzez pocztę email, a także dowolny komunikator internetowy;
- *Widgety Produktowe* – Stanowią one różnego rodzaju banery. W ich skład wchodzą pola takie jak nazwa, cena oraz ukryty przed użytkownikiem końcowym link afiliacyjny do produktu na stronie reklamodawcy.
  Przykład widgetu zaprezentowano na rysunku nr 10;
- *Data Feed* – Możliwość tworzenia ofert produktów w postaci plików *xml,* zawierających strukturę sklepu w postaci elementów posiadających atrybuty takie jak zdjęcie, opisy, cena, a także przyporządkowane kategorie produktów. W rezultacie może to zostać wykorzystane do tworzenia wyszukiwarki produktów czy też systemu ich rekomendacji;
- *Banery i Rotatory* – Stanowią komponenty podobne do *widgetów*, z tą różnicą, że pozwalają reklamować więcej niż jeden produkt, a także usługi jak wyprzedaż czy specjalne promocje. Przykład rotatora zaprezentowano na rysunku nr 9.

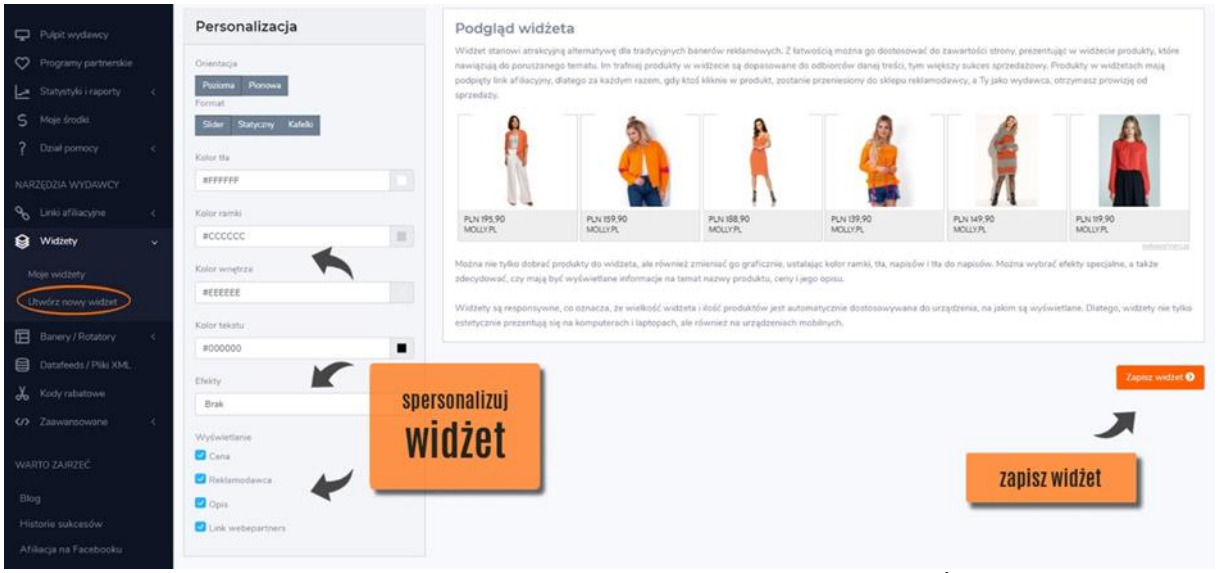
Zobligowany jest on do umieszczenia specjalnego skryptu JS (oparty o *third party cookies, frontend-to-backend*), bądź biblioteki zainkludowanej po stronie serwera działającego w technice *backend-to-backend,* w aplikacji swojego sklepu. Rolą mikroprogramu jest wykrycie i odnotowanie 2 istotnych z punktu widzenia procesu afiliacji akcji:

- Wejściu na stronę reklamodawcy przy wykorzystaniu linku;
- Poprawne wykonanie transakcji zakończone sukcesem, wraz z datą realizacji transakcji (m. in. sprzedaż).

Rozwiązanie to, oparte o pliki *thirdy-party cookies*, jest aktualnie wycofywane z użytku na terenie Unii Europejskiej w związku zastrzeżeniami co do zachowania bezpieczeństwa danych i możliwości ich nieuprawnionego wykorzystania, w związku z czym *backend-to-backend* jest obecnie preferowaną techniką *trackingu* [40].




Rysunek 9 Obraz przedstawia fragment ekranu kreatora rotatorów. Źródło: opracowanie własne




Rysunek 10 Zdjęcie przedstawia przykład kreatora *widgetów*. Źródło: [6]

**Wydawca**

Użytkownik, biorąc udział w tej roli [6], jest uprawniony do udostępniania treści reklamowych na różnego rodzaju nośnikach informacyjnych, których jest właścicielem. Są to m. in.: blogi, portale i serwisy tematyczne, social media, bazy mailingowe, wyszukiwarki i porównywarki itp. itd. Wydawca otrzymuje statystyki na temat uzyskanych prowizji, zrealizowanych poleconych przez siebie transakcji (Rysunek nr 11). W celu prawidłowego umieszczenia ofert produktów w przestrzeni internetowej, wykorzystuje on komponenty i linki uprzednio przygotowane przez reklamodawców wraz z kodem afiliacyjnym, który stanowi unikalny identyfikator tego aktora systemowego.

| Lp | Program | Data kliknięcia | Data zakupu | Wartość zamówienia | Prowizja Wydawcy |
|---|---|---|---|---|---|
| 1 | Włoskie rośliny_cytrusy24 | 28.12.2018 11:32:01 | 29.12.2018 21:41:01 | 60,00 | 9,00 |
| 2 | Włoskie rośliny_cytrusy24 | 27.12.2018 13:26:14 | 27.12.2018 13:59:22 | 150,00 | 22,50 |
| 3 | Włoskie rośliny_cytrusy24 | 26.12.2018 12:52:02 | 26.12.2018 13:06:14 | 330,01 | 49,50 |
| 4 | Włoskie rośliny_cytrusy24 | 25.12.2018 22:53:13 | 26.12.2018 09:46:51 | 249,98 | 37,50 |
| 5 | Włoskie rośliny_cytrusy24 | 19.12.2018 21:38:06 | 19.12.2018 21:48:42 | 394,00 | 59,10 |
| 6 | Włoskie rośliny_cytrusy24 | 17.12.2018 17:11:52 | 17.12.2018 18:03:18 | 150,00 | 22,50 |
| 7 | Włoskie rośliny_cytrusy24 | 17.12.2018 13:39:18 | 17.12.2018 14:07:06 | 150,00 | 22,50 |
| 8 | Włoskie rośliny_cytrusy24 | 17.12.2018 12:48:10 | 17.12.2018 13:25:53 | 96,00 | 14,40 |
| 9 | Włoskie rośliny_cytrusy24 | 12.12.2018 10:48:35 | 12.12.2018 14:22:34 | 72,00 | 10,80 |
| 10 | Włoskie rośliny_cytrusy24 | 06.12.2018 16:44:45 | 11.12.2018 23:06:42 | 60,00 | 9,00 |
| 11 | Włoskie rośliny_cytrusy24 | 30.11.2018 12:51:02 | 10.12.2018 14:27:57 | 60,00 | 9,00 |
| 12 | Włoskie rośliny_cytrusy24 | 07.12.2018 12:52:20 | 07.12.2018 13:00:31 | 90,00 | 13,50 |
| 13 | Włoskie rośliny_cytrusy24 | 06.12.2018 17:13:03 | 06.12.2018 18:10:50 | 156,00 | 23,40 |
| 14 | Włoskie rośliny_cytrusy24 | 03.12.2018 10:23:14 | 03.12.2018 10:43:44 | 266,00 | 39,90 |
| Podsumowanie | | | | 2283,99 | 342,60 |

Rysunek 11 Zrzut ekranu przedstawiający podsumowanie transakcji. Źródło: [7]

Bardzo dużą zaletą tego zestawu narzędzi jest uniwersalność. Integrują one różne platformy na których powstały serwisy sklepów (np. WooCommerce, Magento Shop). Jednak jako rozwiązanie generyczne, a niezintegrowane z aplikacjami służącymi do tworzenia ofert i kosztorysów, posiada kilka istotnych wad:

- Brak wsparcia w procesie polecania produktu poprzez jego wybór na liście w projektach;
- Brak podziału na specyficzną role projektanta pełniącego rolę polecającego, a także klienta dokonującego zakupu z czym wiąże się brak kontroli nad podziałem prowizji z udanych transakcji pomiędzy nimi.

## 2.2. Wymagania wobec prototypu

W tym rozdziale zaprezentowane zostały walory konkurencyjnych rozwiązań, na których powinien bazować prototyp, a także istotne wady i braki w funkcjonalnościach, które powinny zostać rozwiązane. Sporządzony został zarys architektury, który pozwolił spełnić wymagania co do aplikacji. W tej części także przedstawiono motywację autora do podjęcia wybranego tematu pracy.

### 2.2.1 Wnioski z analizy istniejących rozwiązań

Analiza istniejących rozwiązań składnia do poniższych wniosków:

- W dobie popularności urządzeń mobilnych ważne jest wsparcie użytkowników poprzez odpowiednią aplikacją przenośną;
- Wymagana jest integracja z siecią afiliacyjną poprzez opracowanie modułu pośredniczącego między sklepem, projektantem oraz jego klientami;

- W związku z potrzebą obsługi innych cech produktów, a także potencjalną możliwością rozszerzenia funkcjonalności po za projekty sporządzane przez projektantów, przewiduje się dodanie zestawów atrybutów dla cech produktów które nie są ujęte w podstawowych znacznikach HTML według standardu schema.org;
- Wprowadzenia wymagają usprawnienia w odczycie informacji o produktach ze strony sklepu;
- Rozwiązanie powinno być intuicyjne - elementy GUI oraz UX powinny nieść jasne znaczenie i nie wpływać negatywnie na komfort klienta lub utrudniać mu pracę;
- Dostępność rozwiązania – Prototyp zachowa funkcjonalność zarówno w klasycznej przeglądarce internetowej jak i we wtyczce;
- Perspektywy rozwoju – dobór technologii powinien pozwolić na płynną zmianę założeń, a także być otwarty na duże modyfikacje;
- Wymagana możliwość dodania wizualizacji wraz z komentarzem w ramach projektu.

### 2.2.2 Wymagania wobec propozycji rozwiązania

Aplikacja ma na celu dostarczenie rozwiązania, które oferuje przyjazne tworzenie list zakupowych tworzonych przez projektantów (między innymi architektów) oraz udostępnianie usług sieci afiliacyjnej. Obecne rozwiązania oferują proste listy zakupowe lub interaktywne listy w obrębie danego, dedykowanego portalu lub sieci afiliacyjne bez możliwości integracji tych rozwiązań ze sobą. Zaproponowane rozwiązanie pozwoliło na utworzenie serwisu składającego z zestawu niżej wymienionych podsystemów:

- Aplikacja webowa;
- Wtyczka do przeglądarki Chrome;
- Aplikacja na system Android w roli wtyczki mobilnej.

Przewidziano, iż cechą tego rozwiązania, według koncepcji, powinna być intuicyjność UX, ułatwienie pracy projektantów, a także odciążenie klientów od dodatkowych czynności. W związku z różnoraką charakteryzacją produktów, aplikacja pozwala na tworzenie zestawu atrybutów oraz domyślnych tagów html/atrybutów tagów (pola jeśli są dostępne wypełniałyby się automatycznie) typów tekstowych, zdjęć (ładowanie zdjęć z *src* lub *background-image* w wyjątkowych przypadkach) oraz komponentu GUI (Komponent *WebView* w aplikacji android z referencją na adres URL przekazany przez przeglądarkę internetową lub *toolbar* w przypadku klasycznych wtyczek), który pozwala na wybranie zasobów tekstowych/multimedialnych w celu uzupełnienia informacji o produktach, według określonych przez kategorię atrybutów (przed analizą użytkownik miałby możliwość wyboru listy oraz sekcji).

Rozwiązanie pozwala pracować z dowolnymi portalami sklepów, a w szczególnych przypadkach wypełnić formularz ręcznie – gdy produkty nie są dostępne poprzez sklepy internetowe. W tej koncepcji możemy wyróżnić 3 aktorów biznesowych którzy biorą udział w tym procesie – sprzedawca (właściciel portalu internetowego), pośrednik (np. projektant wnętrz, który kompletuje wyposażenie łazienki) oraz klient – osoba która dokonuje zakupu produktów. W celach komercyjnych aplikacja umożliwia dodatkowo utworzenie sieci afiliacyjnej, w której właściciele strony internetowej będą mogą tworzyć programy partnerskie (i wprowadzić ewentualne zmiany do swojego systemu w celu obsługi zdarzenia zakupu po stronie ich serwisów), a projektant – zarabiać na prowizji od wskazanych przez siebie produktów (które zostały zakupione w wykorzystaniem wygenerowanego linku).

### 2.2.3 Propozycja rozwiązania

Wymagania przedstawione powyżej narzucają konieczność podziału prototypu na 3 podsystemy:

- Aplikację webową dostępną przez przeglądarki internetowe;
- Wtyczkę na przeglądarkę Chrome (która po niedużych modyfikacjach może zostać dostosowana do innych przeglądarek);
- Aplikację mobilną;

Interfejs wszystkich tych części musi charakteryzować się prostotą, intuicyjnością, by potencjalny użytkownik końcowy (projektant) nie miał problemu z obsługą wszystkich przewidzianych funkcjonalności. Rozwiązanie jest oparte o architekturę klient-serwer, a także architekturę MVC reprezentowaną przez *framework* Spring.

Serwerowa część prototypu będzie aplikacją monolityczną, jednak podzieloną na logiczne fragmenty zwane modułami. W celu spełnienia wymagań, aplikacja zostanie pofragmentowana w następujący sposób:

**A. Moduł logowania i autoryzacji**

Główną rolą tego modułu jest przeprowadzenie procesu i rejestracji użytkowników, umożliwienie zarządzania danymi o kontach użytkowników przez administratorów. Ważną funkcją części autoryzacyjnej jest zabezpieczenie poszczególnych akcji, wywołań oraz dostępu do poszczególnych ekranów tylko dla osób do tego uprawnionych. Przykładem zastrzeżonej akcji jest administracja grupami użytkowników. Tego typu moduł jest bazową częścią większości aplikacji internetowych.

**B. Moduł zarządzania atrybutami**

Podstawową rolą tego modułu jest możliwość zarządzania zestawami atrybutów przez użytkowników. Zestawy atrybutów pozwalają zdefiniować dowolne cechy produktu oraz przypisać im znaczniki html, które zawierają stosowną informację w strukturze HTML na stronie sklepu. Atrybuty te mogą być tworzone na podstawie różnych typów danych, w ramach powyższych wymagań przewidziano m.in. podane typy: tekstowy, liczbowy, obrazkowy, wyboru (*select*) z kilkoma wariantami. Każdy zestaw atrybutów może zostać przypisany przez użytkownika do danej listy produktów. Wymaganą częścią modułu jest umożliwienie dodawanie kolejnych typów atrybutów na wzór pluginów co pozwala definiować potencjalnie nowe typy pól.

**C. Moduł kreatora list produktów**

Funkcją tego modułu jest możliwość definiowania projektów, sekcji w projekcie, edycji produktu, dodawanie komentarzy i wizualizacji. Pozwala on na wgląd do globalnego spisu produktów, zawierający statystyki wykorzystania danego towaru. Omawiana część aplikacja ściśle współpracuje z modułem afiliacji, a także wtyczką i aplikacją mobilną.

Moduł ten udostępnia usługę eksport projektu do zestawienia .csv.

**D. Moduł afiliacji**

Ta część aplikacji dostarcza rozwiązania do obsługi procesów afiliacyjnych w systemie, zaczynając od generowania linku, rejestrowania transakcji, obsługi *landing page* oraz akcji wywoływanej na *purchase page*. W ramach integracji ze sklepami, moduł ten dostarcza skrypt .js do dołączenia w kodzie strony stron trzecich, który spełnia rolę *trackera*. Moduł ten oferuje zestaw punktów końcowych API obsługujących akcje afiliacyjne po stronie *backend*.

**E. Wtyczka na przeglądarkę Chrome**

Rozszerzenie do przeglądarki internetowej pozwala na uruchomienie akcji dodawania produktu do projektu. Użytkownik, po wybraniu ikony wtyczki na stronie produktu, otrzymuje formularz danych o produkcie, wraz z uzupełnionymi informacjami, jeśli strona internetowa spełnia standardy schema.org i zawiera dane w odpowiednich tagach. Może wybrać tam projekt, który go interesuje, przejrzeć

wizualizacje, ocenić czy dany produkt spełnia wymagania postawione przez projektanta, skorygować dane bądź pobrać dane z ostatniego wybranego produktu używając kojarzenia po linku.

### F. Aplikacja na system Android

Aplikacja na systemy mobilne charakteryzuje się podobną funkcjonalnością, co wtyczka na *desktopową* przeglądarkę internetową, jednak różni się aspektami implementacyjnymi. Link produktu, do parsowania przez aplikację, przesłany zostanie za pomocą akcji w systemie Android o nazwie „*Share*”.

# 3. Współpraca afiliacyjna

W tym rozdziale przedstawiony zostały opis zagadnień merytorycznych, a także omówiono proces polecania produktów poprzez usługi internetowe. Ta część kładzie nacisk na teorię o sieciach afiliacyjnych, a także jakim stopniu wykorzystano ten zakres wiedzy w celu implementacji prototypu. Zagadnienia przedstawione w tym rozdziale, a także proces, zachodzący podczas rejestrowania transakcji zakupu produktu, są niezbędne, by Moduł Afiliacji dział w sposób kompletny i poprawny. Na jej podstawie zrealizowano skrypt śledzący akcje użytkownika, a następnie wysyłający dane pod wskazane punkty końcowe API, zawarte w części serwerowej.

## 3.1. Sieci afiliacyjne

Pojęciem Afiliacja [12] został nazwany *zbiór technik oraz podejmowanych akcji służących do osiągania dochodu pasywnego, który generowany jest przez niewielką prowizję od kupna i polecenia produktów, czy też powiększanie puli klientów firm które są w tzw. programie partnerskich z współpracującymi z nimi afiliatorami*. Jest to model biznesowy, który w historii ludzkości pojawia się od czasów antycznych jako tzw. marketing szeptany, jednak pierwsze użycie tej metody nastąpiło w 1989 r. przez inwestora Wiliama J. Tobina, który założył firmę PC Flowers inc. będącą jednym z pierwszych sklepów internetowych oferującą sprzedaż kwiatów i prezentów okolicznościowych. Firma Prodigy Network dzięki promowaniu produktów omawianego inwestora uzyskiwała prowizję od sprzedaży. Firma w ten sposób uzyskała 2700 partnerów afiliacyjnych, co przesądziło o sukcesie tego modelu biznesowego. [11]

Po roku 1996 nastąpił istotny rozwój działalności w tej dziedzinie w wyniku której przyjęto 4 modele wynagrodzenia podmiotów polecających produkty:

**CPC (*cost per click*)**

Model ten odnosi się gratyfikacji właściciela strony przez reklamodawcę. W tym modelu następuje proces przejścia na jego serwis, po przekierowaniu będącym wynikiem kliknięcia w elementy multimedialne takie jak *widget*, baner reklamowy oraz link. Każde unikalne kliknięcie (przez innych użytkowników w ramach innej sesji) wynagradzane jest niewielką prowizją. [13]

**CPS (*cost per sale*)**

Model ten jednym z najczęściej wykorzystywanych w działalności sklepów internetowych. W procesie zakupu brane pod uwagę jest tylko zarejestrowane wykonanie transakcji, która w zależności od przyjętych kryteriów może być ograniczona tylko do tych które się powiodły sukcesem i nie nastąpił zwrot. [14]

**CPL (*cost per lead*)**

Jest jednym z najczęściej wykorzystywanych modeli przez banki. Polega on na gratyfikacji finansowej akcji wypełnienia formularza, pobrania wskazanej aplikacji czy też rejestracji w serwisie internetowym. [15]

**CPM (*cost per thousand*)**

CPM to wycofywany i rzadko używany model afiliacji. W założeniu reklamodawca przyjmuje daną liczbę wyświetleń reklam za które przyznaje wypłatę. Powszechnie wykorzystywaną wersją była płatność za każde 1000 wyświetleń reklam. [16]

Podział ten może być też rozwinięty co do warunków współpracy.

**Model mieszany**

W ramach współpracy wydawca otrzymuje prowizję uzyskaną z różnych modeli ze zbioru CPC, CPS, CPL, CPM [17]

**Model wielopoziomowy**

Najczęstszym rodzajem tego modelu jest dwupoziomowy wymiar współpracy. Pierwszą część stanowi wynagrodzenie przysługujące z transakcji, a następną – za pozyskanie klientów oraz potencjalnie nowych afiliantów. [17]

## 3.2. Proces afiliacyjny

W tym procesie bierze udział 3 aktorów biznesowych dla których rola każdego podmiotu opisana jest następująco:

1. **Reklamodawca**

Oferuje on produkty z pewnych dziedzin, a także wykonawstwo usług. Reklamodawca zgłasza chęć uczestnictwa w programie afiliacyjnym, co oznacza, że będzie on reklamował swoją działalność drogą sieci afiliacyjnej. Decyduje on o wyborze przyjętej metody lub metod w tym procesie, co najlepiej odpowiada charakterystyce i profilu aktywności sprzedażowej. [18]

2. **Wydawca**

Za pomocą środków wirtualnych (tzw. powierzchnia online) poleca wskazane produkty, bądź usługi na rzecz przychodu reklamodawcy, z którym współpracę zaczął w momencie dołączenie do programu partnerskiego, a następnie otrzymuje od niego gratyfikację za wskazane efekty marketingowe. [18]

3. **Użytkownik**

Podmiot ten to osoba dokonująca transakcji lub także akcji w serwisie internetowym reklamodawcy jako rezultat polecenia przez afilianta. [18]

Afiliacja zaczyna się wraz z momentem zawarcia umowy pomiędzy wydawcą, a reklamodawcą. Fakt ten stanowi także akceptacja regulaminu oraz rejestracja w serwisie oferującym usługi tego rodzaju. Następnie określane są szczegółowe warunki wynagrodzenia. [18]

Reprezentanci pierwszego podmiotu otrzymują dostęp do panelu administratorskiego, w którym możliwy jest przegląd firm polecających. Z jego wykorzystaniem są w stanie akceptować zgłoszenia nowych wydawców, zarządzać uruchomionymi programami, wyświetlać statystyki transakcji oraz aktywności użytkowników skojarzonych z danym wydawcą. Po ustaleniu warunków afiliant może wygenerować linki, kody rabatowe, banery oraz *widgety* które będą później wykorzystywane. W określonych przypadkach dostarcza on własne narzędzia do informowania o produktach bądź wykorzystuje oprogramowanie zaimplementowane przez drugi podmiot procesu. [18]

W przypadku zamiaru reklamowania produktów, istniejących w bazie danych sklepów, wydawca może dostarczyć *Data Feed*, który stanowi plik XML wraz ze strukturą opisującą kolekcję produktów i ich atrybuty co może być pomocne w szybkim wybieraniu tych które będą uczestniczyć w tej usłudze. [18]

Istnieje wiele metod rejestracji udanych transakcji, jednak z punktu widzenia tworzonego oprogramowania, wybrany został następujący sposób:

**Krok 1**. Afiliant tworzy spis produktów, które poleca klientowi w celu ich zakupu. W tym celu za pomocą systemowego generatora linków tworzy odnośniki do ich stron macierzystych, wraz z dodatkowymi parametrami (m.in. unikalny identyfikator polecającego).

**Krok 2**. Aplikacja internetowa sklepu, przy pomocy skryptu dostarczonego przez system, rejestruje fakt przekierowania do serwisu i gromadzi następujące dane: datę przejścia, identyfikator polecającego, informacje o produkcie np. w formacie json. Strony które korzystają z udostępnionego kodu lub skryptu nazywamy *Landing Page*. Szczegółowe informacje o produkcie pobierane są z danych zamieszczonych w źródle strony lub *Product Feed*.

**Krok 3.** Przekierowanie zarejestrowane przez *Landing Page* zostaje wysyłane odpowiedzią zwrotną do serwera aplikacji (technika *Backend-To-Backed*), lub zarejestrowane w *Thirdy-Party Cookie*. W tym momencie zostaje nadany identyfikator *trackingowy* ciasteczka.

**Krok 4.** Użytkownik wybiera kolejne produkty. Informacje o nich są składowane w pliku cookie. Wszystkie produkty wybrane w trakcie pojedynczej sesji są zapisywane jako lista pod wygenerowanym wcześniej identyfikatorem.

**Krok 5.** Użytkownik dokonuje zakupu. Sklep waliduje wybrany koszyk oraz płatność. Gdy zakup kończy się sukcesem, klient zostaje przekierowany na *Purchase Page*. W tym momencie jest wykonywana zostaje inna część udostępnionego kodu która odpowiada za rejestrację zdarzenia dokonania zakupu na dwa sposobu: odczytujące dane z Thirdy-Party cookie i wysyłając na serwer za pomocą synchronicznego wywołania lub bezpośrednio przesyłając treść ze strukturą odpowiedzi w formacie JSON, przygotowany przez aplikację sklepu (*Backend-To-Backend*).

**Krok 6.** System waliduje poprawność transakcji, kojarzy z właściwym polecającym i wylicza prowizję według ustalonych zasad. Zarówno reklamodawca jak i wydawca otrzymuje wgląd do statystyk transakcyjnych.

## 3.3. Miary afiliacyjne

W celu oceny jakości modeli afiliacyjnych, stosuje się następujące wskaźniki. Ich rolą jest określenie stosunku poniesionego kosztu do uzyskanego efektu.

#### A. Koszt kliknięcia (*Cost Per Click*, CPC)

Stanowi on miarę jakościową znaczenia kampanii reklamowej. Wyznaczany jest on następująco (1) [19]:

$$CPC = \frac{Cost_k}{N_k} \tag{1}$$

Gdzie $Cost_k$ stanowi łączny koszt kliknięć, a $N_k$ liczbę kliknięć. Im wyższa wartość, tym kampania ma wyższą wartość marketingową.

#### B. Koszt sprzedaży (*Cost Per Sale*, CPS)

Stanowi on miarę jakościową opłacalności poświęconego czasu względem uzyskanych przychodów. Wyznaczany jest on następująco (2) [19]:

$$CPC = \frac{Cost_k}{Income_k} \tag{2}$$

Gdzie $Cost_k$ stanowi łączny koszt środków przeznaczonych w celach marketingowych, a $Income_k$ sumę dochodów uzyskanych ze sprzedaży w ramach kampanii. Inaczej niż w przypadku wskaźnika CPC, najbardziej pożądanym efektem jest uzyskanie wartości dążącej do 0.

# 4. Wykorzystane narzędzia i technologie

Rozdział ten przedstawia zbiór powszechnie wykorzystywanych praktyk programistycznych, jak również technologie i narzędzia, które wspomogły powstanie aplikacji i umożliwiły implementację założonych funkcjonalności według wskazanych w powyższym rozdziale zasad.

## 4.1. Schema.org

Schema.org [20] jest otwartym standardem wymiany danych, którego misją jest tworzenie, utrzymywanie i promowanie schematów danych strukturalnych w Internecie, na stronach internetowych oraz jego innych środkach multimedialnych.

Standard Schema.org może być używany z wieloma metodami kodowania, głównie RDF, Microdata, a także JSON-LD. W jego zbiór wchodzą encje, relacje pomiędzy nim, działania. Są przystosowane do łatwego rozszerzania zbioru tagów oraz słów kluczowych, pod warunkiem właściwego udokumentowania modelu rozszerzeń. W przestrzeni internetowej istnieje ponad 10 milionów witryn przystosowanych do tego standardu w formie pełnej lub częściowej (wykorzystując tylko część słów kluczowych). Projekt został uruchomiony w 2011 roku przez wyszukiwarki internetowe Bing, Yahoo oraz Google. [20]

W skład Schema.org wchodzą elementy, które wywodzą się z encji Thing, oznaczane są jako [21]:

- Action – Stanowi odwzorowanie czynności wykonywanej przez agentów bezpośrednich i uczestników pośrednich na obiekcie;
- BioChemEntity – Określa zawartość strony na której występuje obiekt biologiczny, chemiczny, a także biochemiczny;
- CreativeWork – W jej skład wchodzą elementy zawierające pracę twórczą jak książki, filmy, fotografie czy też oprogramowanie komputerowe. W jej skłąd wchodzą liczne podencje jak np. Article, Comment, Blog czy też Movie;
- Event – Stanowi opis pewnego zdarzenia, które odbywa się w danym miejscu i czasie, np. koncert, festiwal;
- Intangible – Opisuje różnego rodzaju wartości kwantowe, lub strukturowe;
- MedicalEntity – Generyczny typ opisujący stany zdrowotne czy też praktyki medyczne;
- Organization – W jej skład wchodzą różnego rodzaju instytucje i placówki. Przykładem mogą być podencje Corporation, Project, EducationalOrganization, a także GovernmentOrganization;
- Person – Określa osoby w różnych stanach, także martwe oraz fikcyjne;
- Place – Odnosi się do przestrzeni, jej części, a także środków lokomocji;
- Product – Definiuje występowanie usług bądź produktów, które oferuje strona stosująca tą encję. Jest to najbardziej istotny element standardu Schema.org z punktu widzenia tematyki pracy;
- Taxon – Reprezentuje naturalne jednostki biologiczne.

Schema.org wspiera obsługę kolekcji lub grup produktów, które posiadają pewne wspólne cechy. W tym celu powstały bardziej specyficzne encje jak [21]:

- ProductGroup – Reprezentuje on grupę produktów, które różnią się tylko wybranymi atrybutami jak rozmiar czy kolor;

- ProductCollection – Zawiera on spis produktów, które występują w ramach oferty;
- ProductModel – Stanowi pewny abstrakcyjny opis grupy towarów lub usług zawierające wspólne cechy;
- IndividualProduct – Stanowi pojedynczy identyfikowalny byt encji Product.

Istnieją rozwiązania walidujące poprawność implementacji tego standardu w aplikacjach internetowych. Jednym z nim jest Google Rich Result Test Tool [22] weryfikujący poprawność zastosowania znaczników ze słownika Schema.org, oraz przedstawia sparsowane dane, w celu weryfikacji merytorycznej danych.

Praktyczne zastosowanie tego standardu opiera się na 2 formach:

A. Metatagi w sekcji <*head*> z przykładami zamieszczonymi w listingu 1.

Listing 1 Fragment strony HTML zawierającej znaczniki <*meta*> spełaniające standard Schema.org. Źródło: opracowanie własne

```
<meta property="og:url"
content="https://domni.pl/sklep/umywalki/umywalki-stawiane-na-blat" />


<meta property="og:type" content="website" />


<meta property="og:title" content="Umywalki stawiane na blat - owalne,
okrągłe, prostokątne - Domni.pl" />


<meta property="og:description" content="Wybierz nowoczesne umywalki
stawiane na blat z szerokiej oferty artykuł&#243;w do wyjątkowych,
atrakcyjnych wnętrz." />


<meta property="og:image"
content="https://media.domni.pl/thumbnail/1370w/uploads/1/16/bez-
nazwy_090335877598.png" />


<meta property="og:image:width" content="1370" />


<meta property="og:image:height" content="969" />
```

B. Struktura json-ld w tagu <*script type=”application/ld+json”*> z przykładem zamieszczonym w listingu 2.

Listing 2 Fragment strony HTML zawierającej skrypt json-ld spełniający standard Schema.org. Źródło: opracowanie własne

```
<script type="application/ld+json">
{
    "@context": "http://schema.org",
    "@type": "Product",
```

```
    "offers": [{
          "@type": "Offer",
          "name": "Default Title",
          "availability":"https://schema.org/InStock",
          "price": 378.0,
          "priceCurrency": "PLN",
          "priceValidUntil": "2022-11-13",
          "sku": "7526",
          "url": "/products/szafa-(...)"
    }],
    "gtin13": "435211",
    "productId": "435211",
    "brand": {
          "name": "ChestChef"
    },
    "name": "Szafa pokojowa ChestChef",
    "description": "Futurystyczna forma połączona (...)",
    "category": "",
    "url": "/products/chest-chef",
    "sku": "7576",
    "image": {
          "@type": "ImageObject",
          "url": "https://cdn.shopify.com/s/(...)",
          "image": "https://cdn.shopify.com/s/(...)",
          "name": "Szafa pokojowa ChestChef Frony",
          "width": "1024",
          "height": "1024"
    }
}
</script>
```

## 4.2. Architektura MVC

Klasyczny wzorzec Model-Widok-Kontroler (Model-View-Controller) opisuje sposób tworzenia aplikacji ze środowiskiem graficznym (GUI), w podziale na odseparowane warstwy programowe. Każda z nich odznacza się inną funkcjonalnością. Dzięki podziałowi na 3 moduły osiągnięto wysokie stopień reużywalności kodu. Ich charakterystyka przedstawia się następująco: [23]

**A. Model (Model)**

Warstwa to odwzorowuje zbiór bytów, które istnieją w rzeczywistości i posiadają swoje atrybuty oraz zachowania. Cześć z nich może być odwzorowana przez więcej niż jeden obiekt tej klasy. Oprócz

klasycznych encji, w ich skład wchodzą także obiekty DTO (*Data Transfer Object*), a także zbiór klas klas, które przechowują struktury danych w Logice Zorientowanej Obiektowo (*Domain Drive Development*, DDD). [23]
Istnieją 2 rodzaje modeli:

**Pasywny**
Każda zmiana stanu następuje jest rezultatem wywołania żądania przez kontroler lub widok.

**Aktywny**
Przejścia pomiędzy stanami obiektu zachodzi automatycznie, jednak wymaga to implementacji podsystemów odpowiedzialnych za aktualizację interfejsu użytkownika, a także wysłanie notyfikacji do kontrolera.

**B. Widok (View)**

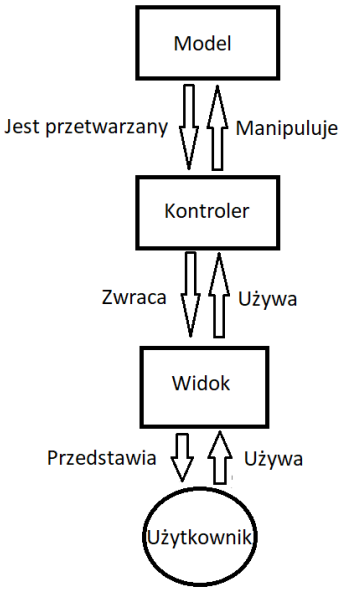
Komponent ten pełni funkcję pośrednika, pomiędzy użytkownikiem końcowym, a logiką aplikacji. W celu prezentacji danych używa on plików *template* opisujących strukturę i zachowanie elementów GUI, a także może być podzielony na podwidoki w celu zastosowania podziału ze względu na zakres informacji. Za ich pomocą użytkownik końcowy, wchodząc w interakcje z tą warstwą, wyzwala odpowiadające im działania w kontrolerze. [23]

**C. Kontroler (Controller)**

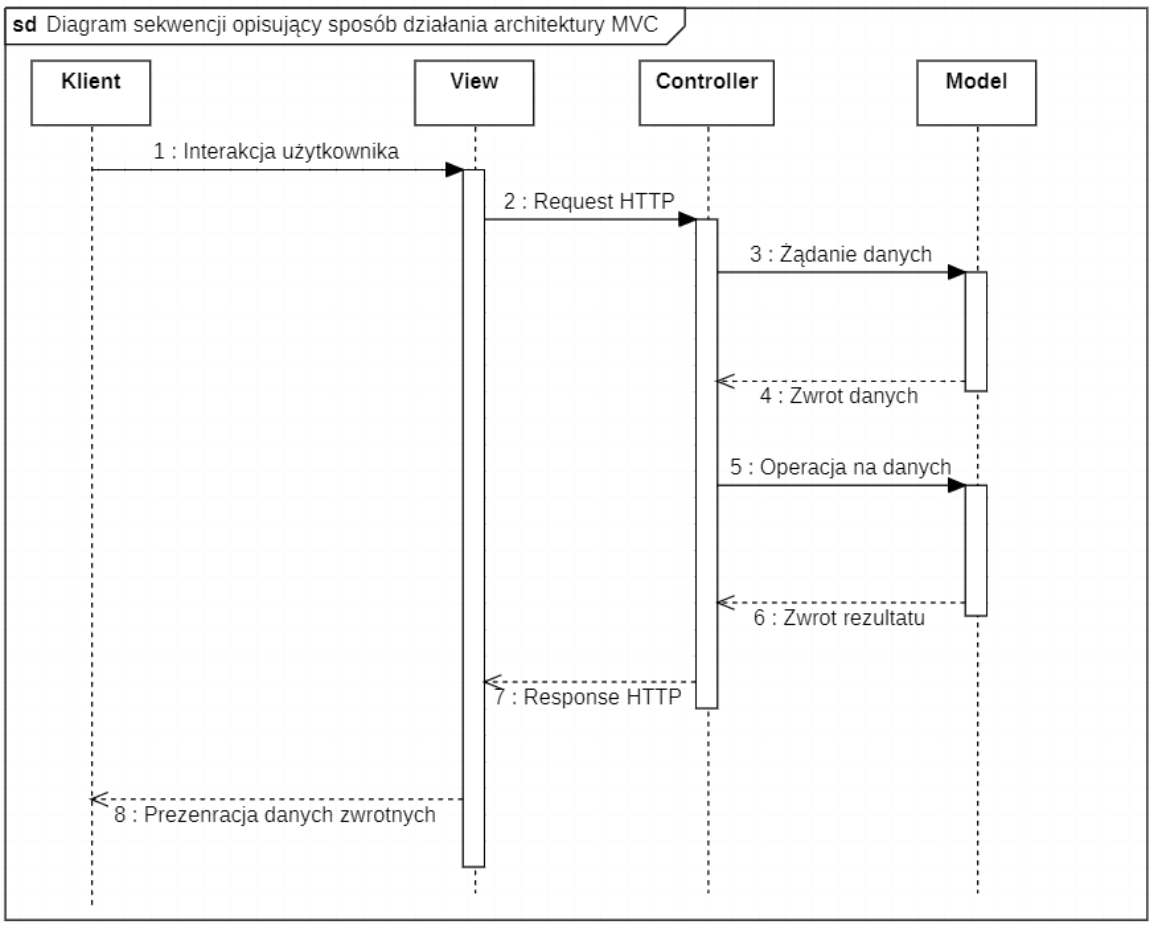
Funkcją kontrolera we wzorcu MVC jest wykonywanie akcji polegających na odbiorze, przetwarzaniu i analizie danych otrzymanych od użytkownika. Po wykonaniu tej akcji może odświeżyć widok, zmienić stany modeli, przekazywać modele do widoku oraz przenieść sterowanie na inny kontroler. Wzajemne relacje, pomiędzy poszczególnymi warstwami architektury MVC, przedstawiono na rysunku 12, zaś interakcje pomiędzy nimi – w formie diagramu sekwencji na rysunku 13. [23]

Do zbioru funkcjonalności tej warstwy należą:

- Przetwarzanie danych oraz wykonywanie operacji na modelach
- Analiza oraz przetworzenie danych wejściowych (*Request* - obiekt opisujący parametry żądania wejściowego do serwera)
- Zwrócenie rezultatu działania odpowiadającego wejściowemu *requestowi* (*Response* – będący modelem zawierającym odpowiedź zwrotną i status HTTP)
- Nanoszenie zmian w widokach, aktualizacja interfejsu
- Przenoszenie sterowania do innego kontrolera poprzez *redirect response* (Kod http 3xx)




Rysunek 12 Diagram przedstawiający relacje pomiędzy poszczególnymi warstwami architektury MVC. Źródło: [24]




Rysunek 13 Diagram sekwencji przedstawiający sposób interakcji pomiędzy użytkownikiem a komponentami MVC. Źródło: opracowanie własne

## 4.3. REST

Sposób komunikacji REST (ang. *Representational State Transfer*) [25], opracowany na potrzeby standaryzacji protokołu http, wprowadził zbiór zasad do architektury klient-serwer. Komunikacja z jego wykorzystaniem odbywa się w sposób bezstanowy.
REST przeznaczony jest dla protokołów HTTP/HTTPS oraz wykorzystuje dużą liczbę dostępnych formatów danych:

- JSON;
- HTML;
- XML;
- YAML;
- OctetStream, a także inne pliki, także w formacie plain\text.

Zastosowano w nim wsparcie dla SSL (*Secure Socket Layer)*. Każdy zestaw informacji jest nim określonym zasobem, do którego dostęp odbywa się z wykorzystaniem URI (Uniform Resource Identifier), który określa zbiór metod wykorzystywanych przy generowaniu żądań do serwera. [25]

### A. GET

Metoda GET pozwala na uzyskanie dostępu do określonego zasobu, z użyciem parametrów w notacji para klucz-wartość. Metoda ta pozwala na utworzenie żądania, w celu ich odczytu z zasobów serwera z wykorzystaniem ścieżki URL, która zawiera zbiór parametrów po znaku specjalnym „?” (Przykład w listingu 3). [25]

Listing 3 Przykład sparametryzowanego żądania GET. Źródło: opracowanie własne

```
/api/garden/tree?occassion=christmass&width=160
```

Przeznaczeniem żądań GET jest pobranie danych. Niewskazane jest wykorzystanie tego typu żądań do akcji modyfikujących dane (szczególnie informacje wrażliwe), ze względu na podatność na luki bezpieczeństwa. Są one przechowywane w historii przeglądarki internetowej. [25]

### B. POST

Standard REST specyfikuje sposób dostępu do serwera w celu modyfikacji zasobu (dostępnego przez pewien uniwersalny identyfikator). Służy do tego metoda POST, której parametry nie są przekazywane przez URL, a treść żądania (listing 4) w dowolnym formacie, który podlega serializacji (JSON, FormData, Graph QL). Requesty tego typu nie są wprost zapisywane w historii, jednak rejestrowane przez narzędzia dla developerów jak np. konsola przeglądarki. Poprzez konfigurację serwera możliwe jest określenie maksymalnego rozmiaru treści żądania. [25]

Listing 4 Przykład żądania POST widoczny w konsoli przeglądarki internetowej. Źródło: opracowanie własne

```
POST /api/garden/tree/145 http/1.1
Host: local.host.example
Body: {
   ”occassion” : ”christmass”,
```

```
   ”width” : 160
}
```

### C. PUT

Posiada ona podobną funkcjonalność co POST i pozwala na utworzenie lub zastąpienie istniejącego zasobu. Istotną różnicą jest idempotentność, co oznacza, że wielokrotne wywołanie żądania PUT zostanie przyjęte jak pojedyncze wywołanie, w przeciwieństwie do metody POST, gdy podczas jej realizacji powstanie N zasobów. Przykład żądania PUT przedstawiono w listingu 5. [25]

Listing 5 Przykład żądania PUT widoczny w konsoli przeglądarki internetowej. Źródło: opracowanie własne

```
PUT /api/garden/tree http/1.1
Host: local.host.example
Body: {
   ”name” : ”TheFirstTree”,
   ”occassion” : ”christmass”,
   ”width” : 160
}
```

### D. HEAD

Metoda ta umożliwia odczyt nagłówków wygenerowanych przez serwer w *response* do żądania. Pozwala to uniknąć np. pobierania dużego pliku przy sprawdzaniu jego typu czy rozmiaru. Przykładowe żądanie HEAD przedstawiono w listingu 6. [25]

Listing 6 Przykład żądania HEAD. Źródło: opracowanie własne

```
/api/garden/tree/145/thumbnail
```

### E. DELETE

Przeznaczeniem DELETE, w komunikacji za pomocą protokołu REST, jest usunięcie zasobu wskazanego przez parametry żądania, najczęściej poprzez unikalny identyfikator. Przykładowe żądanie DELETE przedstawiono w listingu 7. [25]

Listing 7 Przykład żądania DELETE. Źródło: opracowanie własne

```
DELETE /api/garden/tree/145 http/1.1
```

### F. OPTIONS

Specjalny typ *requestów*, który odpowiada za modyfikację parametrów połączenia. Jest także wykorzystywany przy komunikacji z udziałem CORS (*Cross-Origin Resource Sharing*). [25]

## 4.4. Kotlin

Kotlin [26] powstał jako język mający w założenie wprowadzenie pewnych usprawnień względem języka JAVA. Jest on statycznie typowany, wieloplatformowy i obiektowy. Jego działanie opiera się o maszynę wirtualną JAVA i jest interoperacyjny z tym językiem. Pewne cechy składniowe i semantyczne są podobne do Scali np.

- Typ zmiennej umieszczony jest po nazwie, oddzielony znakiem „:”;
- Wsparcie dla programowania proceduralnego z udziałem funkcji.

Język ten współpracuje z narzędziami kontroli wersji Apache Maven, Gradle. Jego głównym zastosowaniem jest możliwość tworzenia aplikacji na platformę Android. Doskonale sprawdza się także przy tworzeniu aplikacji w architekturze MVC. [26]

Listing 8 Przykład klasy pełniącej funkcji serwisu utworzonej w języku Kotlin

```
@Service
class SmartInvalidRequestValidator @Autowired constructor(
    private val validator: Validator
) {
    fun validate(request: Any): ValidationResult {
        var validationResult = ValidationResult()
        val violations = BeanPropertyBindingResult(request,
            request::class.toString())
        validator.validate(request, violations)
        if (violations.hasErrors()) {
            violations.allErrors.map {
                error -> error?.let { message ->
validationResult.addError(ErrorPayload(message.defaultMessage ?:
"undefined error")) }
            }
            validationResult.responseCode = 400
        }

        return validationResult
    }
}
```

Na przykładzie listingu 8 możemy zauważyć, że każda zmienna deklarowana jest słowem kluczowym *var/val*, a funkcja/metoda za pomocą klucza *fun*, stałe z wykorzystaniem frazy const.

Różnica pomiędzy zasadnością zastosowania *var* lub *val* wynika z tego, czy zmienna będzie podlegać zmianie jej wartości. Korzystanie ze słowa kluczowego „*var*” pozwala na dowoloną mutowalność danej zaalokowanej w pamięci w całym okresie działania programu. Inaczej w przypadku typu *val* – jej wartość nie może zostać zmodyfikowana za pomocą operatora przypisania (znak równości). Warto zaznaczyć, iż, nie w przypadku obiektów nie można zmienić referencji, jednak ich

wewnętrzne własności mogą zostać modyfikowane pod warunkiem, że modyfikator widoczności na to pozwala. [27]

W przypadku opracowanego prototypu, język ten bierze udział w tworzeniu części *backend* aplikacji a także wtyczki na system mobilny Android.

## 4.5. Spring MVC

Spring Web MVC [28] jest jedną z części składowych całego *framework*a Spring Framework. Jest on komponentem webowym pozwalającym na tworzenie aplikacji webowych w architekturze Model-View-Controller, przy pomocy mechanizmu *Dependency Injection*. Określa on logiczny podział klas na grupy funkcjonalne z wykorzystaniem adnotacji. Wśród nich możemy wyróżnić:

- Component – stanowi informację dla frameworka o tym, że dana klasa może być wstrzykiwana poprzez dependeny injection;
- Controller (oraz RestController) – jest to zbiorem klas, których zadaniem jest odbiór przesłanych danych oraz zwrot odpowiedzi przy użyciu mapowania adresów URI. Każda metoda kontrolera zwraca odpowiednio przygotowany obiekt ModelAndView. Przykładowy kontroler zaprezentowano w ramach listingu 10;
- Service – wskazuje grupę klas, których instancje odpowiadają za działanie logiki biznesowej. Przykład zamieszczono w listingu 9;
- Repository – są to komponenty, które pośredniczą w wymianie danych pomiędzy aplikacją, a bazą danych;
- Configuration – Obok plików xml odpowiada za konfigurację aplikacji.

Jego działanie opiera się o istnienie tzw. beanów, których definicja przedstawia się następująco:

*W Springu obiekty, które tworzą szkielet aplikacji i są zarządzane przez kontener Spring IoC, nazywane są fasolami. Bean to obiekt, który jest tworzony, składany i zarządzany w inny sposób przez kontener Spring IoC.*. [29]

Za przyjmowanie wszystkich żądań http odpowiada *servlet* o nazwie DispatcherServlet. Zarządza on odpowiednim zagospodarowaniem innych obiektów w celu przetwarzania requestów użytkownika. By móc powiązać adresy wprowadzone w przeglądarce internetowej z konkretnymi funkcjami klas Controller, HandlerMapping odczytuje parametry kolekcji *routes* z wykorzystaniem annotacji typu Mapping. [28]

Listing 9 Przykład klasy *frameworka* oznaczony jako *@Service*. Źródło: opracowanie własne

```
@Service
class UserRoleExtractorSrv {


    fun               getRoles(userAccount:               UserAccount):
MutableCollection<GrantedAuthority> {
        val roles = LinkedList<GrantedAuthority>()


        Hibernate.initialize(userAccount.userGroups)
```

```
        userAccount.userGroups.filter { it.deletedBy == null }.forEach
{ userGroup ->
            userGroup.roles.forEach {role -> roles.add(RoleDto(role as
String))}
        }

        roles.add(RoleDto(Role.ROLE_USER.toString()))
        return roles
    }
}
```

Listing 10 Przykład klasy oznaczony jako *@Controller*. Przedstawiono 2 punkty końcowe typu GET. Źródło: opracowanie własne

```
@Controller
class CategoryFrontController @Autowired constructor(
    private val securityManagerSrv: SecurityManagerSrv
) {
    @GetMapping("/front/categories/selection")
    fun getCategoriesSelection(): ModelAndView {
        val accountId = securityManagerSrv.authenticatedUser()?.id
        val result = ModelAndView("store/category/category-selection")
        result.addObject("account_id", accountId)
        return result
    }

    @GetMapping("/front/category/{id}/manager")
    fun getCategoryManager(@PathVariable id: Long): ModelAndView {
        val accountId = securityManagerSrv.authenticatedUser()?.id
        val result = ModelAndView("store/category/category-manager")
        result.addObject("id", id)
        result.addObject("account_id", accountId)
        return result
    }
}
```

Za pracę z DDL odpowiada Spring JPA oraz Hibernate. Dzięki nim możemy utworzyć klasy encji mapowane obiektowo relacyjnie (ORM) i nadać odpowiednie annotacje własnościom tych modeli określające to, jakiej kolumnie w systemie bazodanowym odpowiadają. Przykładem tego typu encji może być kod przedstawiony w listingu 11. Repozytoria posiadają funkcjonalność automatycznego tworzenia kwerend, na podstawie nazwy metody w klasie implementującej interfejs Repository. W celu zestawienia połączenia z SBD należy skonfigurować plik *application.properties*. [28]

Listing 11 Przykład klasy encji w *frameworku* Spring. Źródło: opracowanie własne

```
@Entity
open class UserKey(): ValidatableTrait {

    @Id
    @GeneratedValue
    open var id: Long? = null

    @ManyToOne(fetch = FetchType.LAZY)
    open var userAccount: UserAccount? = null

    @Column(nullable = true)
    open var key: String? = null

    @Column(nullable=true)
    open var validTo: LocalDateTime? = null

    @Column(nullable=true)
    open var startAt: LocalDateTime? = null

    override fun isValid(): Boolean {
        return validTo === null
    }

}
```

## 4.6. HTML

Język znaczników, który pozwala na tworzenie widoków stron internetowych jest HTML [31] (ang. *HyperText Markup Language*). Pozwala opisać strukturę informacji dokumentu hipertekstowego. Dzięki wykorzystaniu tagów możliwe jest nadanie znaczenia semantycznego fragmentom tekstu.

Znaczniki posiadają strukturę blokową, przy pomocy których definiowany jest wygląd dokumentu, zazwyczaj w oparciu o kaskadowe arkusze stylów (CSS oraz technologie pochodne jak LESS, SCSS). Poszczególnym elementom możemy przypisywać akcje przy pomocy języka skryptowego JavaScript oraz jego frameworków (stopniowo zastępowany przez TypeScript). Wyróżniamy miedzy innymi następujące typy treści: [32]

- Hiperłącza (znacznik *<a>*);
- Akapity (*<p>*);
- Nagłówki *(<h1-h6>);*

- List numerowane (<*ol*>) i nienumerowane (<*ul*>);
- Multimedia (np. <*img*>);
- Bloki sekcji (<*div*>).

Struktura dokumentu w języku HTML jest określona w następujący sposób:

- Istnieje sekcja metadanych (<*head*>) zawierająca znaczniki <*meta*> oraz odwołania do arkuszy css;
- Plik zawiera <*body*> w którym definiowana jest postać wizualna dokumentu;
- Każdy tag może zawiera atrybuty np. *id* (identyfikator elementu, najczęściej wykorzystywany przez język skryptowy js), *class* (klasyfikator elementów szeroko stosowany przy tworzeniu plików styli css).

Listing 12 Fragment formularza (<*form*>) pliku HTML przedstawiający podstawowe elementy służace do wprowadzania danych (np. <*input*>, <*select*>). Zastosowano Bootstrap 5 jako bibliotekę styli *css*. Źródło: opracowanie własne

```
<div class="row">
    <div class="col-3">
        <input type="text" name="item_fieldKey" class="form-control"
placeholder="Klucz">
    </div>
    <div class="col-3">
        <input type="text" name="item_fieldName" class="form-control"
placeholder="Nazwa">
     </div>
     <div class="col-4">
         <select name="item_fieldType" class="form-select" aria-
placeholder="Typ">
             <option value="required">Wymagane</option>
             <option value="optional">Opcjonalne</option>
         </select>
      </div>
      <div class="col-auto align-self-center justify-content-end d-flex
flex-fill">
           <input type="submit" value="Send"
                class="cursor-pointer apc-icon apc-icon__color-danger
apc-icon__minus"
           ></span>
      </div>
</div>
```

W przykładzie listingu 12, zawierającego jeżyk znaczników, można zaobserwować sposób definiowania poszczególnych elementów HTML. Wymagany jest tu: [32]

- Tag otwierający (np. <*div*>);
- Tag zamykający (np. </*span*>), który w przypadku wybranych znaczników może być pominięty (takim przykładem jest obrazem zawarty w <*img*>);
- Zawartość wewnątrz bloku.

Ponadto każdy element może zawierać wiele atrybutów, oprócz wspomnianych wcześniej id i *class*, są to między innymi: [32]

- style – pozwalający definiować style *css* właściwe tylko do wybranego elementu;
- placeholder – Opis pola typu *input*;
- title – Pozwalający zawrzeć dodatkowe informacje w postaci *tooltip*.

## 4.7. Javascript

Javascript (JS) [33] – Skryptowy język programowania stworzony przez firmę Netscape. Jego zastosowaniem są aplikacje internetowe, w których odpowiada za obsługę interakcji z użytkownikiem, a także komunikację z serwerem *backend* działając po stronie przeglądarki internetowej. W latach 90 XX wieku zdefiniowano dokumentacją określającą standard tego języka, nazwany później EcmaScript, rozwijany przez komisję TC39. W zakres jego funkcji wchodzi:

- Reagowanie na wystąpienie zdarzeń;
- Walidacje pól formularza;
- Tworzenie efektów wizualnych;
- Wysyłanie żądań do serwera WWW;
- Odbieranie komunikatów z części *backend* poprzez protokół SOAP.

JS oferuje wsparcie dla wielu bibliotek i rozszerzeń. Jako przykład można podać jQuery, który upraszcza tworzenie akcji wywoływanych przy interakcji z elementami GUI zawartymi w dokumencie HTML. Na jego podstawie powstały liczne *frameworki* rozszerzające podstawowe funkcje js o reaktywność (np. React, Angular, Vue) i stanowość (np. Vuex). Możliwe jest stosowanie tego języka po stronie serwerowej przy użyciu Node.js, a także Ringo. Skrypty js są osadzane na stronie za pomocą znacznika <*script*>. Przykładem wykorzystania zewnętrznego skryptu może być listing 13. [33]

Listing 13 Sposób podłączenia zewnętrznego skryptu JS. Źródło: opracowanie własne

```
<script type=”text/javascript” src=”…”></script>
```

Javascript charakteryzuje się następującymi cechami:

- Wieloparadygmatyzm – Zawiera w sobie instrukcje do programowania obiektowego, funkcyjnego i imperatywnego. Funkcje definiowane są w sposób przedstawiony w listingu 14;
- Skryptowość – Jest on interpretowany przez przeglądarkę i wykonywany niezależnie od innych aplikacji;
- Typowanie dynamiczne – Typy zmiennych nie są określane przez interpreter przy tworzenia/inicjalizacji zmiennej, a podczas przypisywania do niej wartości w czasie wykonywania skryptu;

- Wieloplatformowość – Działają prawidłowo z różnymi przeglądarkami w wielu typach systemów (desktopowych i mobilnych).

Listing 14 Przykład funkcji napisanej w języku Javascript. Źródło: opracowanie własne

```
function getUrlParameter(sParam) {
    let sPageURL = window.location.search.substring(1),
        sURLVariables = sPageURL.split('&'),
        sParameterName,
        i;

    for (i = 0; i < sURLVariables.length; i++) {
        sParameterName = sURLVariables[i].split('=');

        if (sParameterName[0] === sParam) {
            return sParameterName[1] === undefined ? true : decodeURIComponent(sParameterName[1]);
        }
    }
    return false;
}
```

## 4.8. VUE

Jednym z frameworków opartych o JS jest Vue [34]. Przeznaczony jest do budowania kompleksowych interfejsów użytkownika. Wykorzystuje on zarówno HTML, CSS jak i JS, z użyciem modelu deklaratywnego oraz opartego na komponentach programowania. Posiada dwie główne cechy:

- Wsparcie dla *renderowania* deklaratywnego – Stantardowy *syntax* HTML został rozszerzony o szablony pozwalając tworzyć wyjściowe dane HTML. Modyfikacja widoku następuje przy zmianie stanu JavaScript. [34]
- Obsługa reaktywności – zmienne zdefiniowane w komponencie Vue podlegają automatycznemu śledzeniu zmian ich stanu. Jeśli dojdzie do aktualizacji jej wartości, struktura DOM zostanie zaktualizowana. [34]

Vue może zostać użyty do realizacji poniższych założeń:

- Osadzenie komponentów webowych na dowolnej stronie;
- Tworzenie aplikacji typu Single Page (zmiana podstron odbywa się bez przeładowania całości - SPA);
- Rozwijanie serwisów renderowanych po stronie serwera (SSR);

- Rezponsywność – co oznacza, że aplikacja powinna być dostępna na szerokiej gamie urządzeń, a także w terminalach i WebGL.

W celu utworzenia komponentu Vue (*Single-File-Component*) definiujemy w nim szablon oparty o 3 bloki: <*template*>, <*script*>, <*style*>.

W sekcji <*template*> umieszczane zostają znaczniki HTML odpowiadające za wygląd, zaś w sekcji <*style*> dodatkowe style css/less/scss. Pomiędzy takiem <*script*>, a </*script*> wykonywane są akcje importu dodatkowych skryptów, styli i bibliotek, a także określane reguły działania komponentu. [34]

Po słowie kluczowym export umieszczany jest obiekt w następującej postaci (wersja podstawowa) przedstawionej w listingu 15.

Listing 15 Definicja komponentu Vue. Źródło: opracowanie własne

```
export default {
    name: 'ComponentName',
    components: {}
    data() {
        return {…}
    },
    methods: {
       …
    },
    computed: {
       …
    },
    mounted() {
       …
    }
};
```

W Listingu 15 przedstawiono niektóre własności i metody obiektu komponentu Vue. Są to między innymi:

- Data – jest funkcją, która zwraca zestaw wewnętrznych zmiennych komponentu;
- Methods – zawiera funkcje wykorzystywane w SFC;
- Computed – stanowi sekcję wyliczanych własności – czyli funkcje bazujące na pozostałych zmiennych, które są wywoływane w obszarze tego komponentu jako jego własność (co narzuca fakt, że nie mogą być wprowadzane argumenty wejściowe);
- Mounted – stanowi zbiór instrukcji wywołanych w momencie powołania i zamontowania instancji SFC.

## 4.9. PostgreSQL

PostgreSQL [35] jest systemem zarządzania obiektowo-relacyjną bazą danych. Oprócz przechowywania danych w tabelach określonych relacjami oraz udostępnieniu DML, DQL, DDL pozwala na wprowadzenie obiektów, klas i dziedziczenia. Jest on oparty o licencję typu *Open Source*. Oprócz cech takich jak wieloplatformowość i obiektowość charakteryzuje się:

- Możliwością zapisu danych z ograniczeniem tablicy o maksymalnym rozmiarze 32TB, rzędu – 1.6TB, a pola o limicie 1GB;
- Wprowadzeniem indeksowania GiST (oferując wydajne algorytmy do wyszukiwania i sortowania);
- Wsparciem procedur składowanych w PL/pgSQL, PL/Python, PL/Perl, PL/Tcl oraz w formie ciągu parametryzowanych instrukcji SQL;
- Partycjonowaniem;
- Rozszerzoną obsługą więzów spójności.

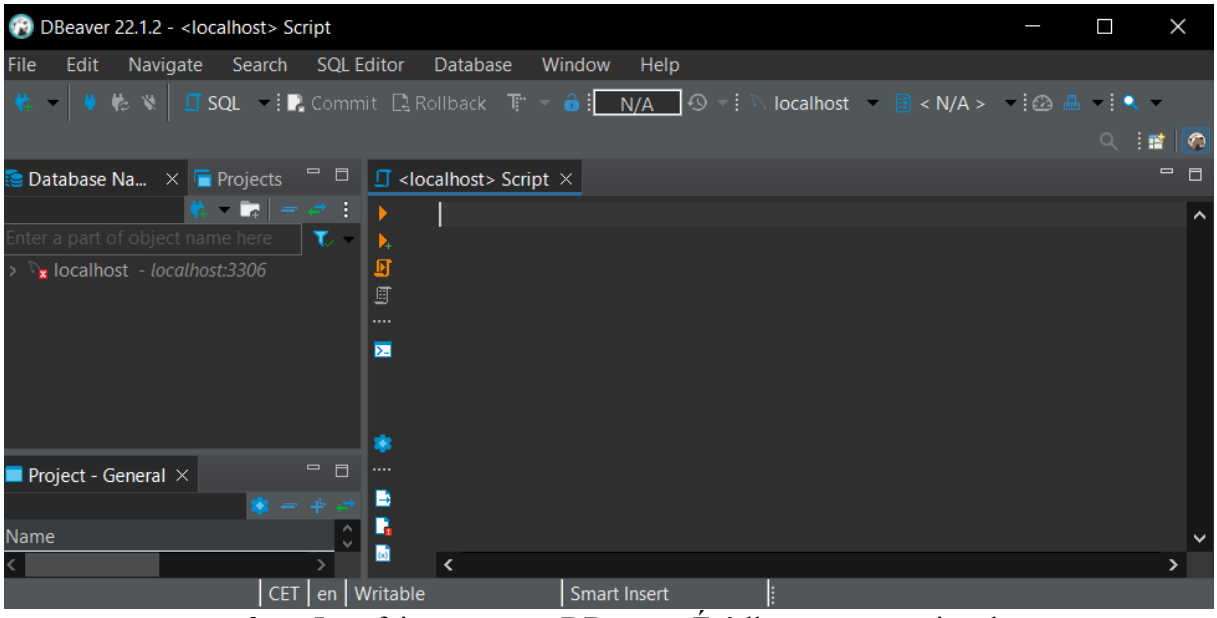
W bazie tej zastosowano kilka rodzajów indeksów zakładanych zarówno na kolumnie tabeli czy widoku zmaterializowanego. Przewidziano w nim obsługę wielu typów indeksów jak B-drzewo, Hash, GiST, BRIN i GIN. W skład PostgreSQL wchodzą wyzwalacze uruchamiane automatycznie przed lub bo operacjach typu DML (*Data Manipulation Language*). Oprócz standardowych w języku SQL typów danych, można przechowywać też typy JSON, UUID, XML czy geometryczne (point, line, path). Oferuje wsparcie dla funkcji okienkowych (np. RANK, DENSE_RANK, ROW_NUMBER). [35]

Przykładem zestawu instrukcji w dialekcie PostgreSQL może być poniższy listing 16:

Listing 16 Przykład instrukcji SELECT w PostgreSQL. Źródło: opracowanie własne

```
SELECT * FROM (
                SELECT *, DENSE_RANK() OVER (
                    PARTITION BY list_id, item_id, product_uid
                    ORDER BY created_at DESC
                ) AS action_order
                FROM product_inst
                WHERE supplier_id = 6
                    AND created_at >= '2020-12-16'
                    AND created_at <= '2021-12-15'
) AS event_data_prepared
```

W celu bieżącej obsługi bazy danych, w projekcie zastosowano oprogramowanie DBeaver (przedstawione na rysunku 14), które oferuje graficzne narzędzie do operacji DML, DQL, DDL czy też tworzenia procedur składowanych PL/pqSQL



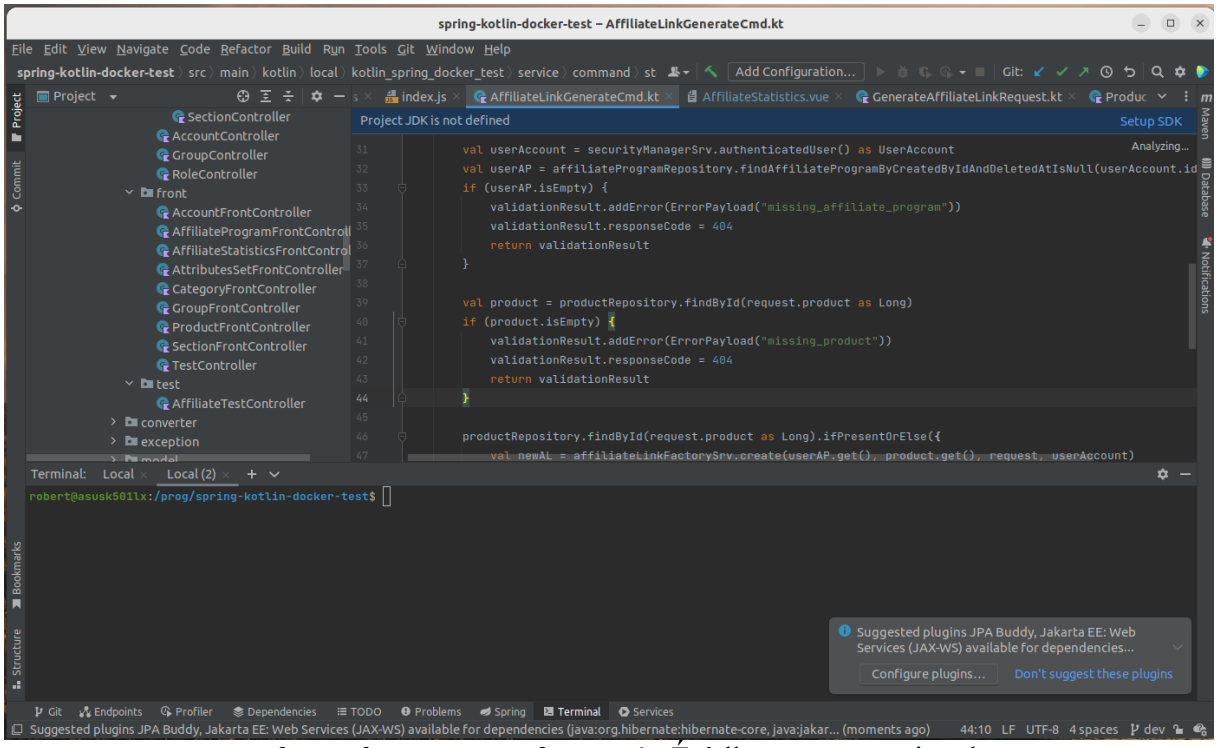
Rysunek 14 Interfejs programu DBeaver. Źródło: opracowanie własne.

## 4.10.Intelij IDEA

InteliJ IDEA [36] wchodzi w skład zestawu narzędzi przeznaczonych dla programistów rozwijanych przez firmę JetBrains, powstałą w Czechach w 2000 r. Jest komercyjnym środowiskiem do tworzenia oprogramowania w językach opierających się na maszynie wirtualnej JVM (Java, Kotlin). Aplikacja ta dostępna jest w wersji 30 dniowej do przetestowania. Narzędzie jest otwarte na modyfikację poprzez dodatkowe pluginy, które umożliwiają pracę także z innymi językami programowania. W standardzie tego IDE jest współpraca z:

- Językami opartymi o maszynę wirtualną Javy (Kotlin, Scala);
- Plikami hipertekstowymi HTML oraz stylami CSS, a także skryptami i frameworkami JavaScript;
- Formatami wymiany danych XML/XLS oraz JSON;
- Językami Python, Ruby, Groovy czy Perl;
- Kwerendami i instrukcjami SQL.

W skład IDEA wchodzi integracja z licznymi frameworkami i technologiami jak JSP, Spring, Hibernate/JPA, JSF, WebServices, JUnit, AJAX oraz narzędziami Git, Apache Maven, Apache Ant, JUnit, Gradle. Pozwala na uruchomienie serwera Tomcat, a także usług kontenerowych w oparciu o środowisko Docker. Narzędzia wchodzące w skład pakietu JetBrains obługują zarówno platformy Windows, macOS jak i systemy operacyjne z jądrem Linux. Interfejs użytkownika tego IDE zorganizowany jest w formie podokien, które przedstawiono na rysunku nr 15. [36]



Rysunek 15 Ekran IDE InteliJ IDEA. Źródło: opracowanie własne

## 4.11.Android Studio

W związku z potrzebą opracowania nowoczesnego środowiska programistycznego na platformę Android powstał Android Studio [37]. Jest on rozwinięciem omawianego w poprzednim punkcie Intelij IDEA. Kładzie ono nacisk na 3 języki programowania: Java, Kotlin, a także natywne programowanie w C++. W skład tego IDE wchodzi emulator systemów Android (przy czym dostępne są różne wersje API Android w szerokiej gamie wirtualnych urządzeń). W celu kreowania widoków aplikacji zostało udostępnione narzędzie w trypie WYSIWYG, dzięki którym dane sekcje wybranego *template* XML mogą być edytowane poprzez listę zagnieżdżoną, edytor atrybutów oraz mechanizm *drag&drop*. Twórcy przewidzieli możliwość importu starszych projektów utworzonych w oprogramowaniu Eclipse.

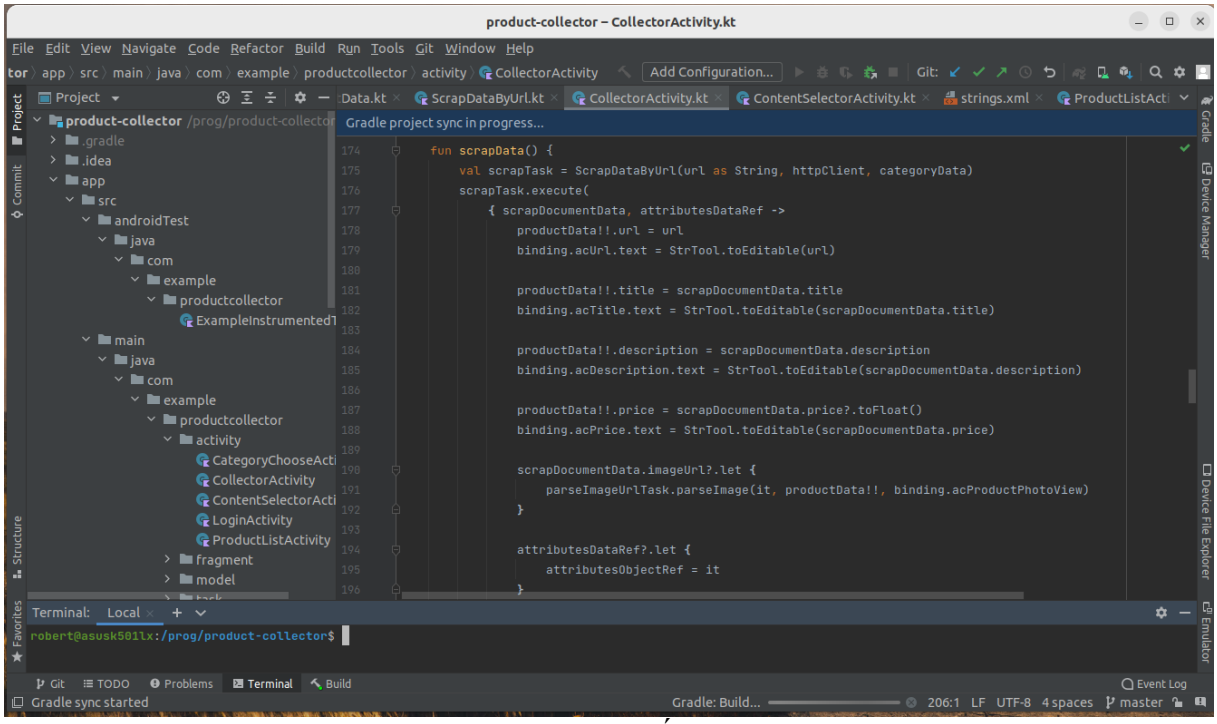
Kreator konfiguracji obecny w oprogramowaniu ściśle współpracuje z Gradle.

Aplikacje na platformę Android składają się kilku rodzajów klas [38]:

- Aktywności (*Activities*) – Implementują one interfejs Activity. Sterują widokami oraz obsługują akcje użytkownika;
- Fragmenty (*Fragments*) – Odpowiadający za wybrany komponent GUI. Klasy implementujące interfejs Fragment są zarządzane poprzez *FragmentManager*;
- Adapter – Pełni rolę pomostu pomiędzy kolekcjami danych, a widokami Adaptera. Służy do zarządzania listami, tablicami, a także innymi rodzajami kolekcji renderującac tylko te elementy, które są widoczne na danym ekranie;
- Widget – Odpowiadają za możliwość tworzenia *widgetów* GUI;
- Services – Klasy, których zadaniem jest wykonywanie procesów biznesowych;
- Intents – Stanowią one żądanie do wywołania nowej aktywności, pełnią rolę eventów;

- Content Provider – Dbają o dostęp do treści zapisanych w różnych rodzajach mediach;
- BroadCast Receiver – Ich zadaniem jest odbiór Intent-ów rozgłoszeniowych i ich odpowiednia obsługa.

Konfiguracja, a także widoki skorelowane z aktywnościami, fragmentami i adapterami są określone w plikach XML. Interfejs użytkownika tego IDE zorganizowany jest w formie podokien przedstawiono na rysunku nr 16. [37]



Rysunek 16 Ekran Android Studio. Źródło: opracowanie własne

# 5. Specyfikacja wymagań

Rozdział ten skupia w sobie zbiór wymagań projektowych, a także opis sposobu konstrukcji modeli danych oraz procesów, które będą w tym systemie obsługiwane.

## 5.1. Zarys

Aplikacja ma na celu dostarczenie rozwiązania, które oferuje przyjazne tworzenie list zakupowych tworzonych przez projektantów (między innymi architektów) oraz udostępnianie usług sieci afiliacyjnej. Obecne rozwiązania oferują proste listy zakupowe lub interaktywne listy w obrębie danego, dedykowanego portalu lub sieci afiliacyjnej, bez możliwości integracji tych rozwiązań ze sobą. Zaproponowane rozwiązanie pozwala na utworzenie oprogramowania składającego z zestawu usług:

1. aplikacja serwerowa;
2. aplikacja webowa;
3. wtyczka do przeglądarek (rozwiązanie dla głównej przeglądarek na rynku: Chrome);
4. aplikacja mobilna pełniąca rolę wtyczki.

Cechą tego rozwiązania, według koncepcji, jest intuicyjność ux, ułatwienie pracy projektantów, a także odciążenie klientów od dodatkowych czynności. W związku z różnoraką charakteryzacją produktów aplikacja pozwoli na tworzenie zestawu atrybutów oraz domyślnych tagów html/atrybutów tagów (pola jeśli są dostępne wypełniałyby się automatycznie) typów tekstowych, zdjęć (ładowanie zdjęć z src lub background-image w wyjątkowych przypadkach) oraz komponentu GUI (Komponent WebView w aplikacji android z referencją na adres URL przekazany przez przeglądarkę internetową lub toolbar w przypadku klasycznych wtyczek), który pozwoli na wybranie zasobów tekstowych/multimedialnych, w celu uzupełnienia informacji o produktach według określonych przez kategorię atrybutów (przed analizą użytkownik miałby możliwość wyboru listy oraz sekcji).

Rozwiązanie pozwala pracować z dowolnymi portalami sklepów i w szczególnych przypadkach pozwala wypełnić formularz ręcznie – w przypadku produktów, które nie są dostępne drogą sklepów internetowych. W tej koncepcji możemy wyróżnić 3 aktorów biznesowych, którzy biorą udział w tym procesie – sprzedawca (właściciel portalu internetowego), pośrednik (np. projektant wnętrz, który kompletuje wyposażenie łazienki) oraz klient – osoba, która dokonuje zakupu produktów. W celach komercyjnych aplikacja umożliwia dodatkowo utworzenie sieci afiliacyjnej, w której właściciele strony internetowej będą mogli tworzyć programy partnerskie (i wprowadzić ewentualne zmiany do swojego systemu w celu obsługi zdarzenia zakupu po stronie ich serwisów), a projektant – zarabiać na prowizji od wskazanych przez siebie produktów (które zostały zakupione w wykorzystaniem wygenerowanego linku).

## 5.2. Aktorzy biznesowi

Przy pracy z oprogramowanym prototypem udział bierze 4 aktorów systemowych:

### A. Sprzedawca

Sprzedawca jest właścicielem sklepu, który zgłasza się z zamiarem uczestnictwa w procesach afiliacyjnych. W tym celu tworzy program partnerski uzyskując unikatowy klucz API do komunikacji z serwerem. W ramach uczestnictwa zgadza się odstąpić część zysków z produktów poleconych przez projektantów jako prowizję na poziome, który on sam określił. Po jego stronie spoczywa odpowiedzialność za zamieszczenie na stronie kodów integrujących stronę zewnętrzną z opracowanym prototypem.

### B. Projektant

Projektant jest osobą, która na zlecenie klientów tworzy wizualizacje oraz projekty architektury wnętrz. W tym celu tworzy dla użytkownika końcowego listy zakupowe wymaganego wyposażenia, które realizuje on samo lub klient zależnie od rezultatu ustaleń pomiędzy oboma aktorami. Przeszukuje on treści internetowe i za pomocą wtyczki dodaje dany produkt do listy. Ma prawo zarządzać zestawami atrybutów specyfikującymi produkty zależnie od kategorii i przeznaczenia. Może także przeglądać zestawienia dotyczące wykorzystania danych towarów. Do każdego projektu może on dołączać komentarze i wizualizacje. W przypadku produktów, które są udostępniane w ramach programu partnerskiego generowany jest link afiliacyjny z unikalnym identyfikatorem polecającego jako parametr GET w URL.

### C. Klient

Klient ma wgląd do udostępnionych projektów poprzez architektów wnętrz. Może wykonać export do pliku csv, a także wnosić komentarze. Wykonuje on transakcję zakupu (lub udziela zgody na wydelegowanie tej operacji projektantowi) wykorzystując wygenerowane przez system linki afiliacyjne.

### D. Administrator techniczny

W jego gestii jest zarządzanie kontami użytkowników oraz wprowadzanie bieżących korekt w danych przetwarzanych w systemie, bądź pomocy użytkownikom zewnętrznym w integracji z systemem.

## 5.3. Wymagania projektowe – funkcjonalne

Poniżej przedstawiono pełną listę wymagań funkcjonalnych postawionych prototypowemu systemowi:

**Aktor systemu: Właściciel sklepu**

W1. System musi umożliwić rejestrację/logowanie się do systemu przez właścicieli sklepów;

W2. System musi umożliwić założenia programu partnerskiego w celu oferowania usług sieci afiliacyjnej;

W3. System musi umożliwić wygenerowanie kodu źródłowego w celu integracji systemu z aplikacją zewnętrzną, co pozwoli na to, że:

- Klient po wejściu do sklepu przez link afiliacyjny umieszczony na liście zakupowej wywołuje zdarzenie systemowe „rozpoczęto transakcję zakupu wskazanego produktu", którego odbiorcą jest aplikacja zewnętrzna;
- Po dokonaniu transakcji aplikacja zewnętrznej generuje zdarzenie „potwierdzenia transakcji", którego odbiorcą jest system;
- W przypadku braku dokonania transakcji w ciągu 3 dni następuje zmiana statusu na „nie zrealizowano".

W4. System musi umożliwić wygenerowanie statystyk odnośnie sprzedaży produktów za pośrednictwem systemu, a także jaką część sprzedaży właściciel sklepu internetowego jest należny projektantom jako prowizja.

**Aktor systemu: Projektant**

W5. System musi umożliwić rejestrację/logowanie się do systemu przez projektantów;

W6. System musi umożliwić definiowanie list zakupowych (wraz z sekcjami). Listy mogą być prywatne lub współdzielone;

W7. Projekt może być kojarzone z zestawem atrybutów;

W8. Zestaw atrybutów musi być elastyczny w tworzeniu tzn. dostarczać możliwość utworzenia dowolnego pola tekstowe, wyboru, selekcji, obrazkowego wraz ze kojarzonym meta znacznikiem (lub dowolnym niespecyficznym znacznikiem innym niż <meta>) umieszonym na dowolnej stronie internetowej sklepu, który określa daną cechę produktu;

W9. Wtyczka do systemu musi zapewnić możliwość kolekcjonowania kluczowych informacji o produkcie za pomocą technologii webscrapingowych;

W10. Wtyczka do systemu musi zapewnić automatyczny odczyt danych cech produktów z dowolnej strony internetowej;

W11. W przypadku braku danego znacznika system pozwala na interakcję ze stroną w celu zaznaczenia i załadowania danych (tekstowych, graficznych itp) do tego pola;

W12. Projektant musi posiadać wgląd do listy sklepów, które oferują sieć linków afiliacyjnych;

W13. Projektant musi posiadać wgląd do statystyk udostępnionych linków do produktów (wraz z uzupełnionymi danymi o produktach) wraz z informacją o przychodzie z prowizji;

W14. System musi umożliwić eksport listy zakupowej do zestawienia pdf lub csv w celu przesłania klientowi np. pocztą e-mail;

W15. System musi umożliwić wgląd do globalnej listy produktów, w której zawarte będą statystyki odnośnie wykorzystania danego produktu na listach.

**Aktor systemu: Klient**

W16. System musi umożliwić rejestrację/logowanie się do systemu przez projektantów;

W17. System musi umożliwić podgląd listy zakupowej udostępnionej przez projektanta;

W18. System musi umożliwić eksport listy do zestawienia pdf lub csv;

W19. System musi umożliwić dodanie uwag w formie komentarza do udostępnionej listy zakupowej.

## 5.4. Wymagania projektowe – niefunkcjonalne

Zbiorem wymagań niefunkcjonalnych ujętych podczas procesu projektowania architektury zostały punkty wymienione w tabeli 1.

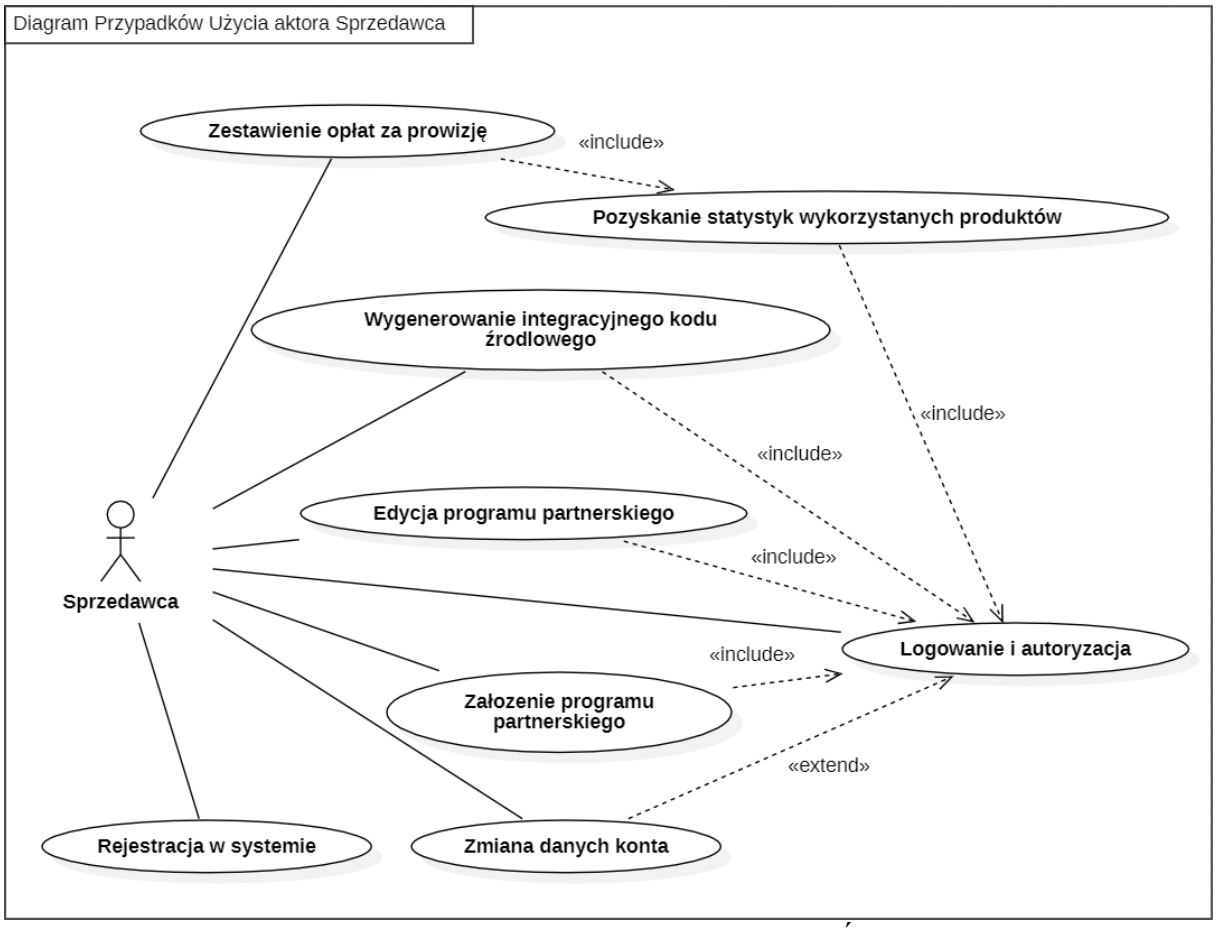
Tabela 1 Specyfikacja wymagań niefunkcjonalnych. Źródło: opracowanie własne

| Identyfikator | Wymaganie | Miara |
|---|---|---|
| F1 | Zgodność strony ze standardami: HTML5, CSS3, DOM2 | W3C Validator |
| F2 | Praca pod rygorem minimum 300 użytkowników online | Dostępność usługi |
| F3 | Zdolność do utrzymania funkcjonalności systemu po awarii zasilania - UPS | Dostępność usługi |
| F4 | Aplikacja mobilna powinna być kompatybilna z wersją min SDK 22 (Wersja API Android 5.1) | Kompatybilność starszych urządzeń |
| F5 | Zapewnienia szyfrowanego kanału komunikacji | Bezpieczeństwo |

## 5.5. Przypadki użycia – diagram

W poniższych diagramach *Use Case* przedstawiono zbiór akcji możliwych do wykonania przez aktorów biznesowych systemu z podziałem na role użytkownika. Relacje pomiędzy czynnościami są opisane stereotypami: *<<include>>*, *<<extend>>*.

### 5.5.1 Sprzedawca

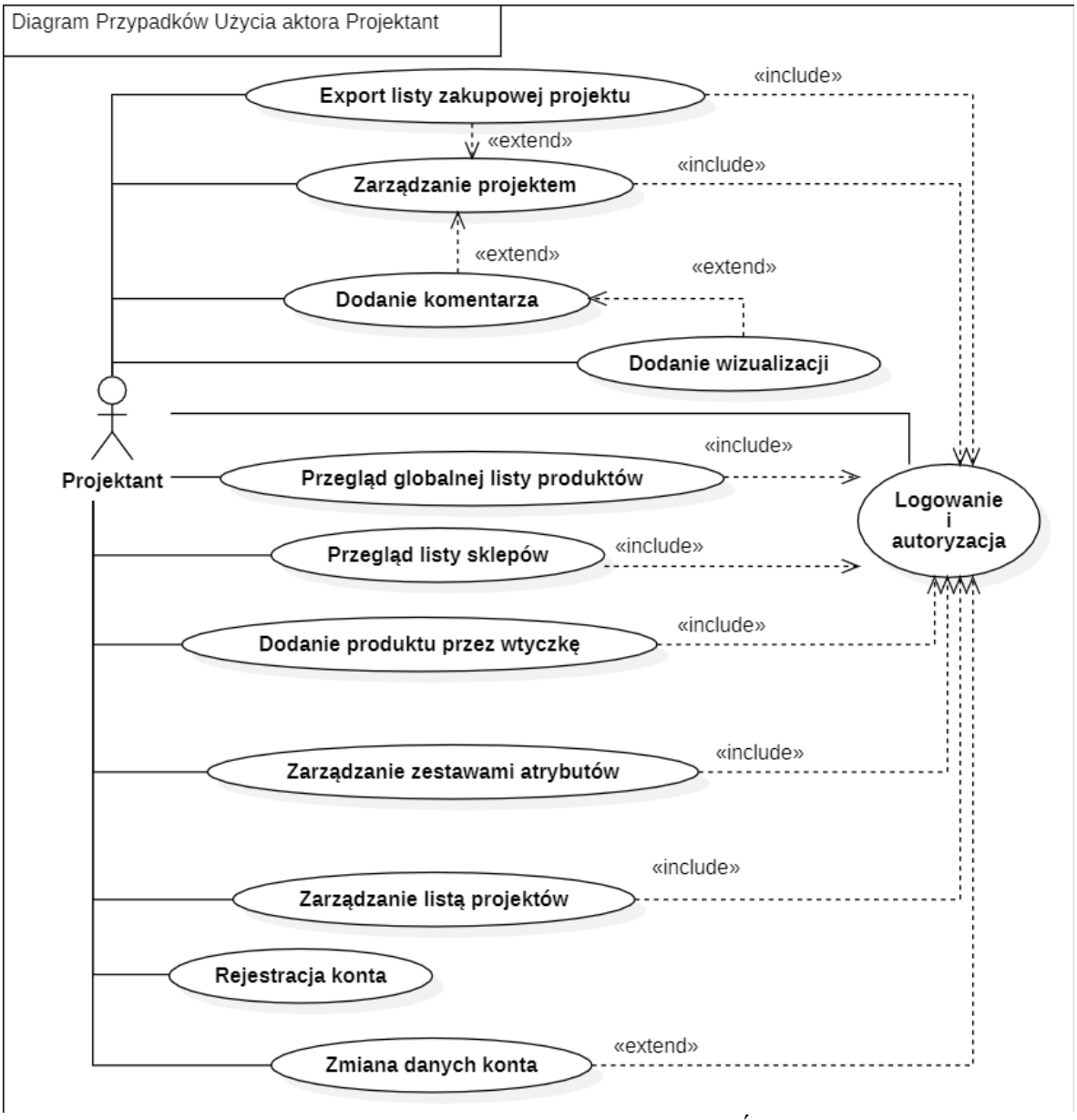



Rysunek 17 Diagram przypadków użycia aktora "Sprzedawca". Źródło: opracowanie własne

W skład funkcjonalności tego użytkownika wchodzą następujące czynności z diagramu na rysunku nr 17:

- Rejestracja konta w systemie oraz autoryzacja;
- Możliwość zmiany danych konta;
- Zakładanie oraz edycja programów partnerskich;
- Uzyskanie kodu integracyjnego;
- Tworzenie zestawów opłat i statystyk.

### 5.5.2 Projektant

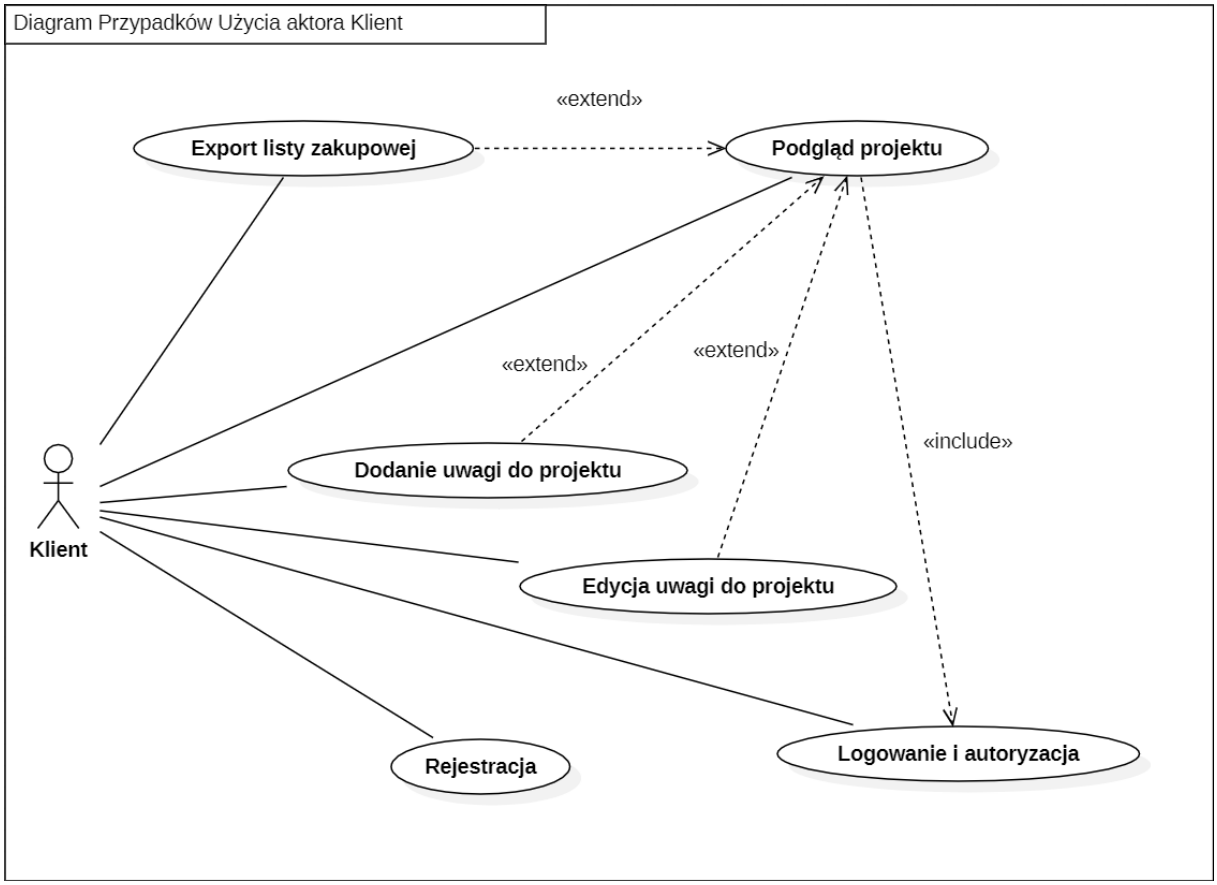



Rysunek 18 Diagram przypadków użycia aktora "Projektant". Źródło: opracowanie własne

W skład funkcjonalności tego użytkownika wchodzą następujące czynności z diagramu na rysunku 18:

- Rejestracja konta w systemie oraz autoryzacja;
- Możliwość zmiany danych konta;
- Zarządzenie listą projektów oraz produktami na nim obecnymi, możliwość jej udostępnienia klientom/współpracującym projektantom;
- Kreowanie własnych zestawów atrybutów;
- Dodawanie produktów poprzez wtyczkę;
- Przegląd globalnej listy produktów oraz sklepów;
- Dodawanie wizualizacji oraz komentarzy do projektów;
- Eksport zestawienia do pliku .csv.

### 5.5.3 Klient

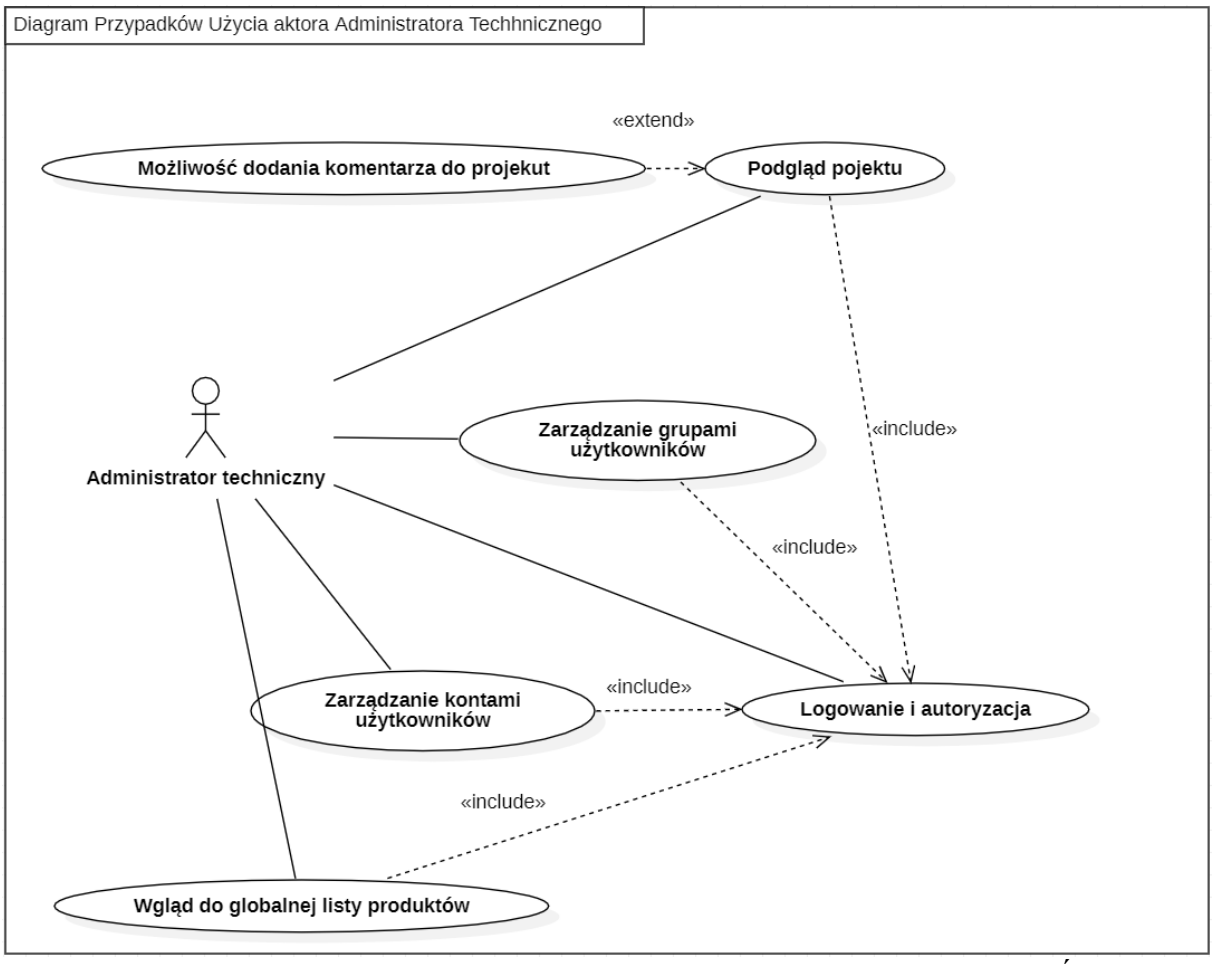



Rysunek 19 Diagram przypadków użycia aktora "Projektant". Źródło: opracowanie własne

W skład funkcjonalności tego użytkownika wchodzą następujące czynności z diagramu na rysunku 19:

- Rejestracja konta w systemie oraz autoryzacja;
- Możliwość zmiany danych konta;
- Przegląd udostępnianych projektów oraz listy produktów na nim się znajdującej;
- Dodawanie uwag/komentarzy do projektów;
- Eksport zestawienia do pliku .csv.

### 5.5.4 Administrator techniczny




Rysunek 20 Diagram przypadków użycia aktora "Administrator techniczny". Źródło: opracowanie własne

W skład funkcjonalności tego użytkownika wchodzą następujące czynności z diagramu przedstawionego na rysunku 20:

- Rejestracja konta w systemie oraz autoryzacja;
- Możliwość zmiany danych konta;
- Administracja kontami użytkowników;
- Zarządzanie grupami;
- Przegląd projektów oraz listy produktów na nim się znajdującej;
- Dodawanie uwag/komentarzy do projektów;
- Nanoszenie zmian merytorycznych w bazie danych na żądanie klientów w wyjątkowych sytuacjach.

## 5.6. Specyfikacja niektórych przypadków użycia

Opis interakcji odbywających się w ramach systemu pomiędzy aktorami, a systemem został przedstawiony w poniższych podrozdziałach w formie tabel określających specyfikację przypadków użycia. Zawierają one nazwę, listę aktorów, warunek końcowy, a także scenariusze podstawowe i alternatywne, jeśli taka potrzeba zaistniała.

### 5.6.1 Moduł logowania i autoryzacji

Liczba przypadków użycia realizowanych przez prototyp jest relatywnie duża w związku z czym przedstawiono tylko najbardziej istotne specyfikacje use-case z punktu widzenia danego modułu. Ich opis, w ramach modułu „Logowanie i autoryzacja", przedstawiono na tabelach 2-4.

Tabela 2 Przypadek użycia "Logowanie". Źródło: opracowanie własne

| Nazwa | Logowanie |
|---|---|
| Aktor(rzy) | Sprzedawca, Projektan, Klient, Administrator |
| Warunek końcowy | Dostęp do funkcjonalności aplikacji |
| Scenariusz główny | 1. Przedstawienie formularza logowania przez system – start;<br>2. Wprowadzenie poświadczeń przez aktora;<br>3. Zweryfikowanie danych przez system;<br>4. W przypadku udzielenia poprawnych danych przekierowanie użytkownika do właściwej części aplikacji – koniec; |
| Scenariusz alternatywny – Niepoprawny login lub hasło | 4. W przypadku udzielenia niepoprawnych danych odświeżenie stanu strony;<br>5. Poinformowanie użytkownika o wystąpieniu problemu z autentykacją – koniec. |

Tabela 3 Przypadek użycia "Zmiana danych". Źródło: opracowanie własne

| Nazwa | Zmiana danych |
|---|---|
| Aktor(rzy) | Sprzedawca, Projektant, Klient, Administrator |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór przycisku „Konto" w górnym menu użytkownika – start;<br>2. Przedstawienie przez system formularza do wprowadzenia zmodyfikowanych danych;<br>3. Wprowadzenie danych przez użytkownika;<br>4. Przetworzenie i walidacja danych z pkt. 3;<br>5. W przypadku powodzenia operacji widok zostaje odświeżony – koniec; |
| Scenariusz alternatywny (autoryzacja nieudana) | 5. W przypadku błędu wskazanie nieprawidłowości użytkownikowi – koniec. |

Tabela 4 Przypadek użycia "Dodanie nowej grupy". Źródło: opracowanie własne

| Nazwa | Dodanie nowej grupy |
|---|---|
| Aktor(rzy) | Administrator |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór przycisku grupy – start;<br>2. Przedstawienie użytkownikowi listy grup;<br>3. Wprowadzenie nazwy nowej grupy, zaakceptowanie wyboru przyciskiem „Dodaj”;<br>4. Walidacja nazwy przez serwer;<br>5. W przypadku sukcesu dodanie nowego elementu do listy;<br>6. Wybranie nowo dodanej grupy;<br>7. Wyświetlenie pełnego formularza edycji;<br>8. Wprowadzenie danych przez użytkownika;<br>9. Zapisanie danych przez system;<br>10. W przypadku powodzenia operacji przekierowanie do listy grup – koniec; |
| Scenariusz alternatywny (nieprawidłowa nazwa lub konflikt z istniejącą grupą) | 5. W przypadku błędu wyświetlenie odpowiedniego komunikatu – koniec; |
| Scenariusz alternatywny (nieprawidłowe dane w formularzu) | 10. W przypadku błędu wyświetlenie odpowiedniego komunikatu;<br>11. Oznaczenie pól ze złymi danymi kolorem czerwonym – koniec. |

### 5.6.2 Moduł kreatora list produktów

Opis istotnych przypadków użycia z modułu „Kreator list produktów” przedstawiono na tabelach o numerach: 5, 6, 7.

Tabela 5 Przypadek użycia "Dodanie produktu przez wtyczkę". Źródło: opracowanie własne

| Nazwa | Dodanie produktu poprzez wtyczkę |
|---|---|
| Aktor(rzy) | Projektant |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wywołanie ikony rozszerzenia w ekranie przeglądarki na stronie wybranego produktu – start;<br>2. Przedstawienie formularza dodawania produktu po prawej stronie wraz z danymi częściowo uzupełnionymi przez wtyczkę na podstawie schema.org oraz microdata;<br>3. Wprowadzenie przez użytkownika brakujących danych do formularza;<br>4. Uzupełnienie bądź korekcja wartości dodatkowych atrybutów<br>5. Wybór przycisku „Dodaj” przez projektanta;<br>6. Weryfikacja i walidacja danych przez systemach; |

| | |
|---|---|
| | 7. W przypadku poprawnego zapisu wyświetlenie odpowiedniego alertu. Jeśli przeciwnie – wyświetlenie komunikatu błędu – koniec; |
| Scenariusz alternatywny (z przeglądem komentarzy i wizualizacji) | 1. Wywołanie ikony rozszerzenia w ekranie przeglądarki na stronie wybranego produktu – start;<br>2. Przedstawienie formularza dodawania produktu po prawej stronie wraz z danymi częściowo uzupełnionymi przez wtyczkę na podstawie schema.org oraz microdata;<br>3. Wybranie zakładki „komentarze i wizualizacje”;<br>4. Przegląd wizualizacji przez użytownika;<br>5. Weryfikacja zgodości dodawanego produktu z założeniami w projekcie. Możliwość zakończenia procesu;<br>6. Wprowadzenie przez użytkownika brakujących danych do formularza;<br>7. Uzupełnienie bądź korekcja wartości dodatkowych atrybutów;<br>8. Wybór przycisku „Dodaj” przez projektanta;<br>9. Weryfikacja i walidacja danych przez systemach;<br>10. W przypadku poprawnego zapisu wyświetlenie odpowiedniego alertu. Jeśli przeciwnie – wyświetlenie komunikatu błędu – koniec. |

Tabela 6 Przypadek użycia "Dodanie projektu". Źródło: opracowanie własne

| Nazwa | Dodanie projektu |
|---|---|
| Aktor(rzy) | Projektant |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór sekcji „listy” – start;<br>2. Wprowadzenie nazwy nowej grupy, zaakceptowanie wyboru przyciskiem „Dodaj”;<br>3. Walidacja nazwy przez serwer;<br>4. W przypadku powodzenia zwrócenie nowego elementu do widoku. Powrót do kroku 2;<br>5. Wybranie nowego projektu kontrolką „zarządzaj”;<br>6. Rozwiniecie zakładki „edycja projektu” przez użytkownika;<br>7. Wprowadzenie nazwy, opisu, zestawu atrybutów przez użytkownika; Modyfikacja listy użytkowników, którym udostępniono projekt;<br>8. Aktualizacja danych „live” przez system;<br>9. Dodanie sekcji do projektu – koniec. |

Tabela 7 Przypadek użycia "Dodanie komentarza do projektu". Źródło: opracowanie własne

| Nazwa | Dodanie komentarza do projektu |
|---|---|
| Aktor(rzy) | Projektant |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór sekcji „listy” – start;<br>2. Przedstawienie przez system listy projektów;<br>3. Przejście do interesującego użytkownika projektu;<br>4. Wyświetlenie listy dodanych produktów;<br>5. Przełączenie zakładki komentarze przez użytkownika; |

| | |
|---|---|
| | 6. Przedstawienie wcześniej dodanych komentarzy;<br>7. Wybór przycisku „Dodaj Nowy Komentarz”;<br>8. Zmiana statusu widoczności formularza do dodawania nowego komentarzu;<br>9. Wypełnienie formularzu nowego komentarza oraz wybór przycisku „Zapisz”;<br>10. Zapis nowego obiektu do bazy – koniec; |
| Scenariusz alternatywny (z wizualizacją) | 1. Wybór sekcji „listy” – start;<br>2. Przedstawienie przez system listy projektów;<br>3. Przejście do interesującego użytkownika projektu;<br>4. Wyświetlenie listy dodanych produktów;<br>5. Przełączenie zakładki komentarze przez użytkownika;<br>6. Przedstawienie wcześniej dodanych komentarzy;<br>7. Wybór przycisku „Dodaj Nowy Komentarz”;<br>8. Zmiana statusu widoczności formularza do dodawania nowego komentarzu;<br>9. Wypełnienie formularzu nowego komentarza;<br>10. Dodanie obrazków w postaci załączników jako wizualizacje projektu;<br>11. Zapis nowego obiektu oraz załączników do bazy – koniec. |

### 5.6.3 Moduł afiliacji i integracji

Opis istotnych przypadków użycia z modułu „Afiliacje i integracje”, przedstawiono na tabelach o numerach: 8, 9, 10.

Tabela 8 Przypadek użycia "Założenie programu partnerskiego". Źródło: opracowanie własne

| Nazwa | Założenie programu partnerskiego |
|---|---|
| Aktor(rzy) | Sprzedawca |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór przycisku „konto” w górnym menu – start;<br>2. Przedstawienie ekranu edycji danych użytkownika;<br>3. Wybór pola wprowadzania „Dodaj program partnerski”;<br>4. Wyświetlenie formularzu programu partnerskiego przez system;<br>5. Wypełnienie wymaganych danych przez użytkownika;<br>6. Zlecenie zapisu z wykorzystaniem przycisku „Zapisz”;<br>7. W przypadku powodzenie operacji odświeżenie ekranu. Błąd walidacji któregokolwiek pola powoduje koniec procesu i zwrócenie komunikatu do frontu;<br>8. Przedstawienie krótkiego kodu integracyjnego do zamieszczenia na stronie sprzedawcy w bloku <code></code>. – koniec. |

Tabela 9 Przypadek użycia "Generowanie linku afiliacyjnego". Źródło: opracowanie własne

| Nazwa | Generowanie linku afiliacyjnego |
|---|---|

| | |
|---|---|
| Aktor(rzy) | 1. Projektant; |
| Warunek wstępny | 1. Poprawna autentykacja i autoryzacja;<br>2. Istnienie programu partnerskiego założonego przez aktora „Sprzedawca” na produkty oferowane przez wybraną firmę; |
| Scenariusz główny | 1. Wybór przycisku „listy” w menu górnym – start;<br>2. Przedstawienie przez system spisu dostępnych projektów;<br>3. Wybór interesującego użytkownika projektu;<br>4. Wyświetlenie menedżera projektu wraz ustawieniami, sekcjami, a także komentarzami;<br>5. Wskazania konkretnego produktu przez projektanta, kliknięcie w przycisku wygeneruj link;<br>6. Uzupełnienie pola „link” o URL zmodyfikowany o parametr get informujący o id polecającego jako identyfikator programu partnerskiego;<br>7. Udostępnienie wygenerowanych w ten sposób linków klientowi – koniec; |
| Scenariusz alternatywny (linki wygenerowane w zestawieniu csv) | 5. Wysłanie żądania pobrania pliku csv z listą produktów w sekcjach;<br>6. Przygotowanie pliku przez system wraz z linkami afiliacyjnym;<br>7. Zwrócenie użytkownikowi odpowiedzi jako załącznik .csv – koniec. |

Tabela 10 Przypadek użycia "Pobieranie zestawienia statystyk". Źródło: opracowanie własne

| | |
|---|---|
| Nazwa | Pobieranie zestawienia statystyk |
| Aktor(rzy) | Sprzedawca, Projektant |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór przycisku „statystyki” w menu górnym – start;<br>2. Przedstawienie przez system ekranu statystyk wraz z filtrami po dacie początkowej i końcowej okresu rozliczeniowego;<br>3. Wybór odpowiedniej daty „od”, „do” przez użytkownika;<br>4. Zwrócenie przygotowanie wyników przez system;<br>5. Wyświetlenie zestawienia transakcji – koniec:<br>A. Zrealizowanych dzięki linkowi afiliacyjnemu wygenerowanego przez aktora „projektant”;<br>B. Zarejestrowane w ramach programu partnerskiego założonego na aktora „sprzedawca”. |

### 5.6.4 Moduł atrybutów

Opis dwóch najbardziej istotnych przypadków użycia z modułu „Atrybuty produktów” przedstawiono na tabelach o numerach: 11, 12.

Tabela 11 Przypadek użycia "Tworzenie nowego zestawu atrybutów". Źródło: opracowanie własne

| | |
|---|---|
| Nazwa | Tworzenie nowego zestawu atrybutów |

| | |
|---|---|
| Aktor(rzy) | Sprzedawca |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wybór elementu „Zestawy atrybutów” w menu górnym – start;<br>2. Przedstawienie przez system listy zestawów atrybutów dodanych przez użytkownika;<br>3. Przejście do formularza dodawania zestawu atrybutów przez użytkownika;<br>4. Prezentacja ekranu przez system, podzielonego na 2 sekcje – dane o zestawie oraz pola w nim zawarte;<br>5. Uzupełnienie nazwy, klucza oraz typu atrybutu przez projektanta;<br>6. Dodanie do listy nowego atrybutu z możliwością wprowadzenia: podtypu atrybutu oraz powiązanych znaczników html;<br>7. Wprowadzenie informacji o kolejnych N atrybutach przez aktora procesu;<br>8. Wybranie przycisku „zapisz” przez użytkownika;<br>9. Walidacja wprowadzonych danych w systemie oraz wykonanie operacji zapisu;<br>10. W przypadku powodzenie przekierowanie na listę zestawów atrybutów – koniec; |
| Scenariusz alternatywny (z wizualizacją) | 10. W przypadku błędu wyświetlenie informacji o przebiegu walidacji przez system;<br>11. Zapis aktualnego stanu edytora przez system – koniec. |

Tabela 12 Przypadek użycia "Pobieranie wartości atrybutów przez wtyczkę/aplikację Android". Źródło: opracowanie własne

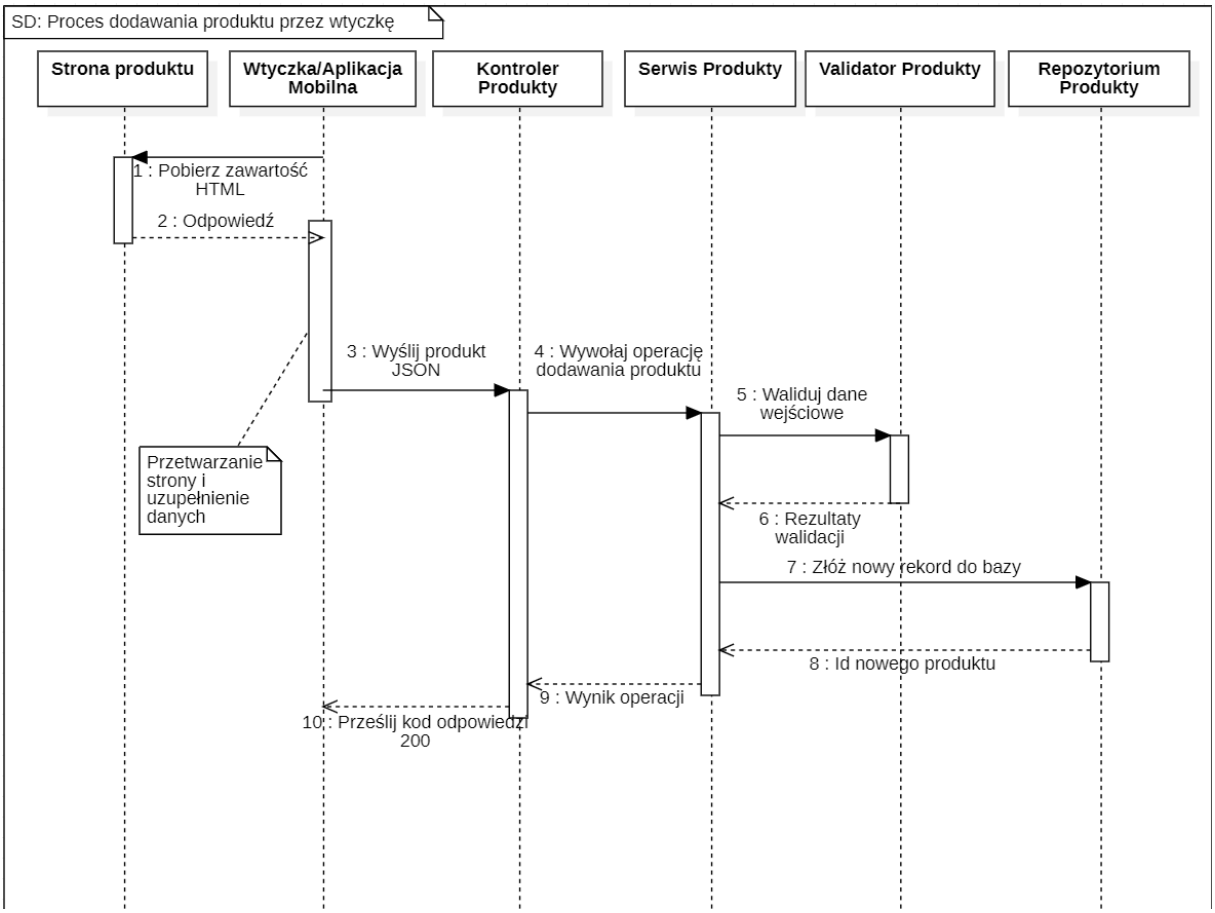
| | |
|---|---|
| Nazwa | Pobieranie wartości atrybutów przez wtyczkę/aplikację Android |
| Aktor(rzy) | Sprzedawca |
| Warunek wstępny | Poprawna autentykacja i autoryzacja |
| Scenariusz główny | 1. Wywołanie ikony rozszerzenia w ekranie przeglądarki na stronie wybranego produktu – start;<br>2. Przedstawienie formularza dodawania produktu po prawej stronie wraz z danymi częściowo uzupełnionymi przez wtyczkę na podstawie schema.org oraz microdata;<br>3. Pobranie zestawu atrybutów skojarzonego z aktualnie wybraną kategorią i sekcją;<br>4. Ustawienie wartości dodatkowych atrybutów poprzez parsowanie i wyszukiwanie zawartości skojarzonych z nimi znaczników HTML na stronie;<br>5. Uzupełnienie wartości pozostałych atrybutów przez użytkownika poprzez 2 metody: reczne wprowadzanie/wybór selecta lub poprzez zaznaczenie fragmentu na stronie i wybraniu przycisku ulokowanego obok pola którego zadaniem jest wkopiowanie zaznaczonego tekstu do pola atrybutu – koniec. |

## 5.7. Specyfikacja wybranych procesów biznesowych

Poniższe podrozdziały zawierają diagramy sekwencji, których rolą jest przedstawienie kolejności czynności i zdarzeń odbywających się pomiędzy modułami.

### 5.7.1 Proces dodawania produktu z wykorzystaniem wtyczki

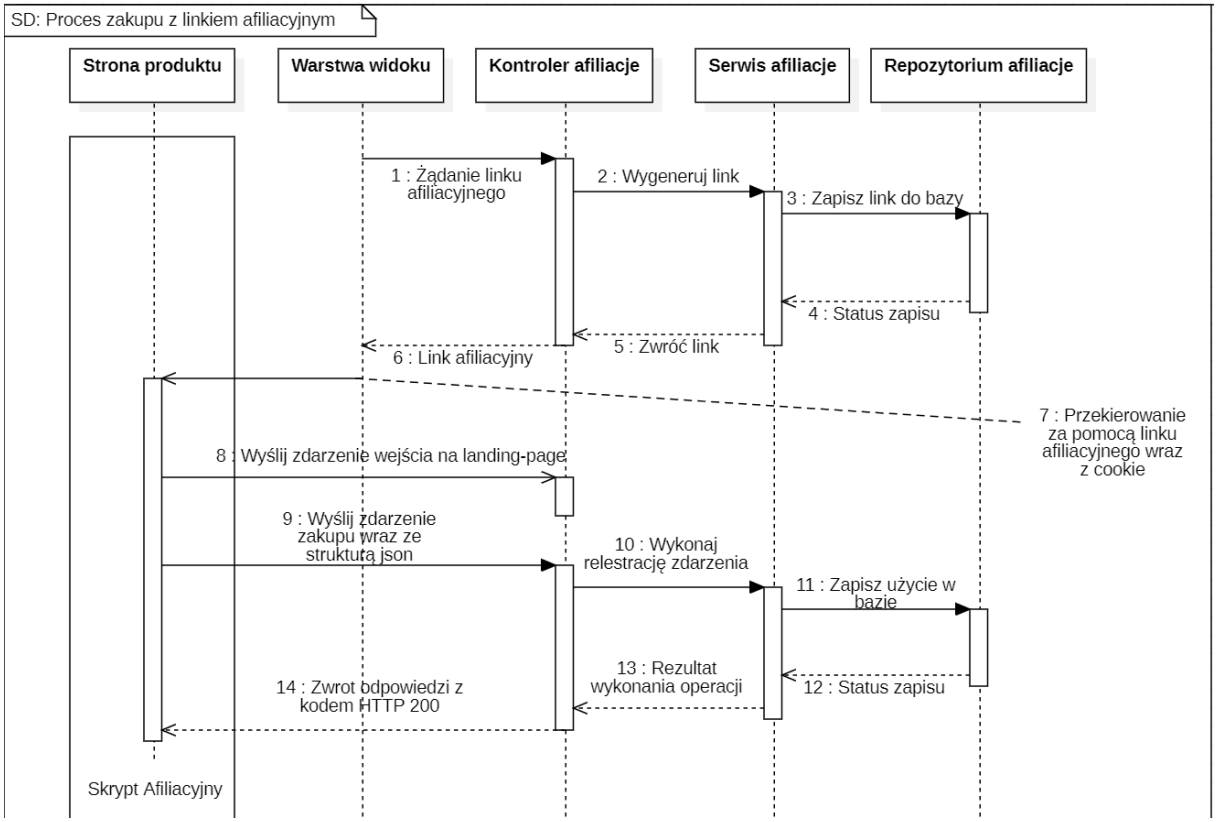
Rysunek nr 21 przedstawia interakcje zachodzące pomiędzy częściami prototypu, w momencie dodawania nowego produktu przez wtyczkę. Całość opiera się o współpracę wtyczki z częścią serwerową, której architektura zbudowana jest zgodnie z MVC.




Rysunek 21 Diagram sekwencji procesu dodawania produktu z wykorzystaniem wtyczki

### 5.7.2 Proces zakupu produktu z wykorzystaniem linku afiliacyjnego

Rysunek nr 22 przedstawia interakcje zachodzące pomiędzy częściami prototypu, w momencie zakupu produktu z wykorzystaniem wygenerowanego linku afiliacyjnego. Tuż po jego utworzeniu, aplikacja przekierowuje użytkownika na stronę produktu. Następnie skrypt osadzony przez właściciela strony przejmuje sterowanie i rejestruje zdarzenia założenia cookie po wejściu na stronę *landing-page*, a także dokonania transakcji zakupu zakończonej powodzeniem.

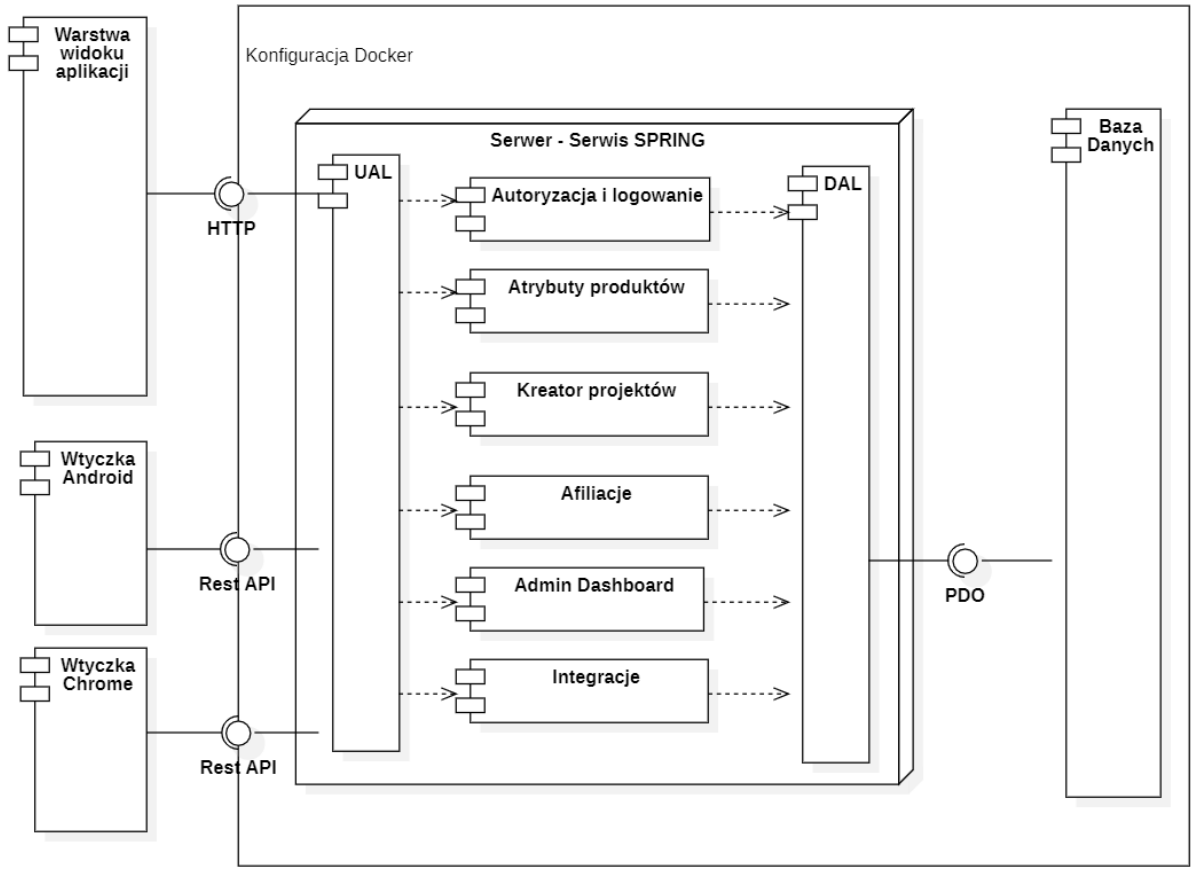



Rysunek 22 Diagram sekwencji procesu zakupu produktu z wykorzystaniem linku afiliacyjnego

## 5.8. Architektura systemu

Poniższe podrozdziały opisują przyjętą propozycję utworzenia architektury systemu, która składa się z komponentów, pakietów a także modelu danych przechowywanych w bazie danych aplikacji.

### 5.8.1 Diagram komponentów




Rysunek 23 Grafika przedstawiająca diagram komponentów systemu. Źródło: opracowanie własne

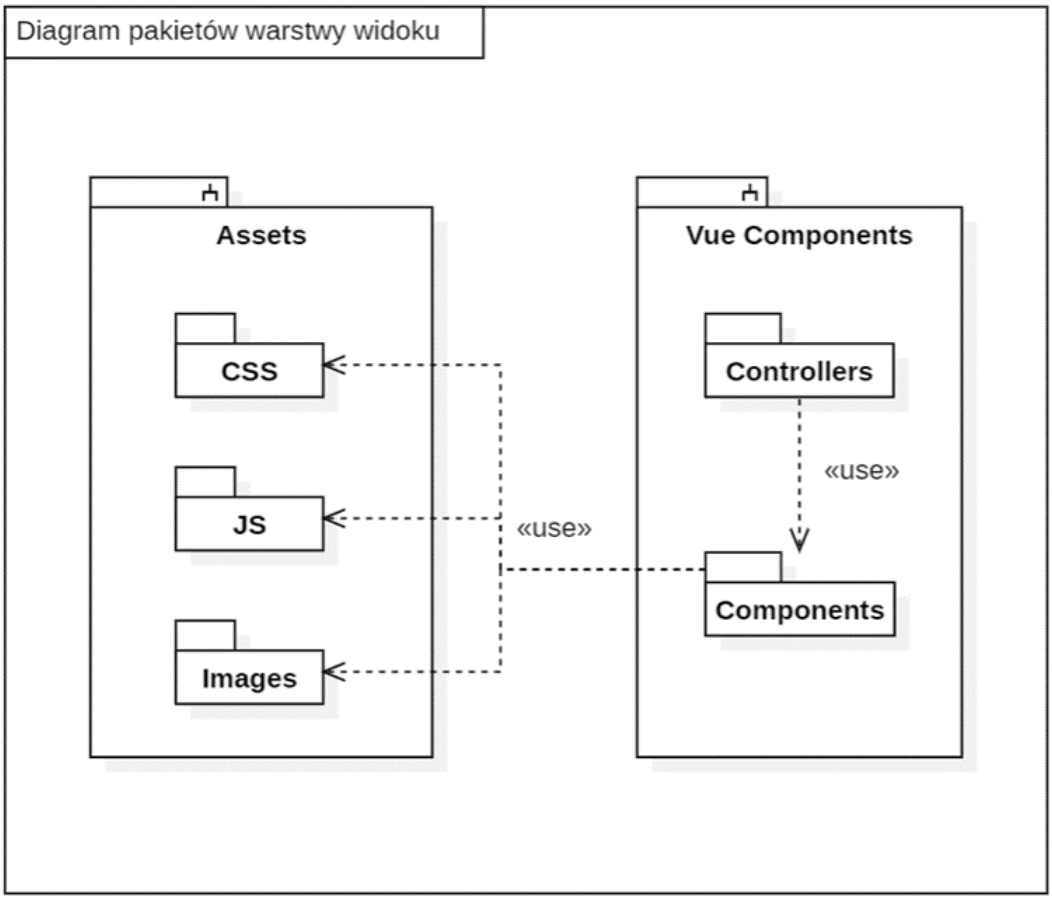
Na rysunku 23 można zauważyć, że aplikacja jest podzielona 3 główne części:

1. Kontener platformy Docker wraz z plikami serwerowymi oraz odseparowaną warstwą widoku, oraz obraz DBMS. Kanałem komunikacji modułów użytkownika końcowego i serwera jest protokół HTTP (port 8080) w oparciu o REST, zaś w celu uzyskania dostępu do bazy wykorzystano port 5432 oraz sterownik JDBC;
2. Wtyczka do kolekcji produktów na platformy Google Chrome;
3. Aplikacja do kolekcji produktów w systemie Android.

### 5.8.2 Diagram pakietów

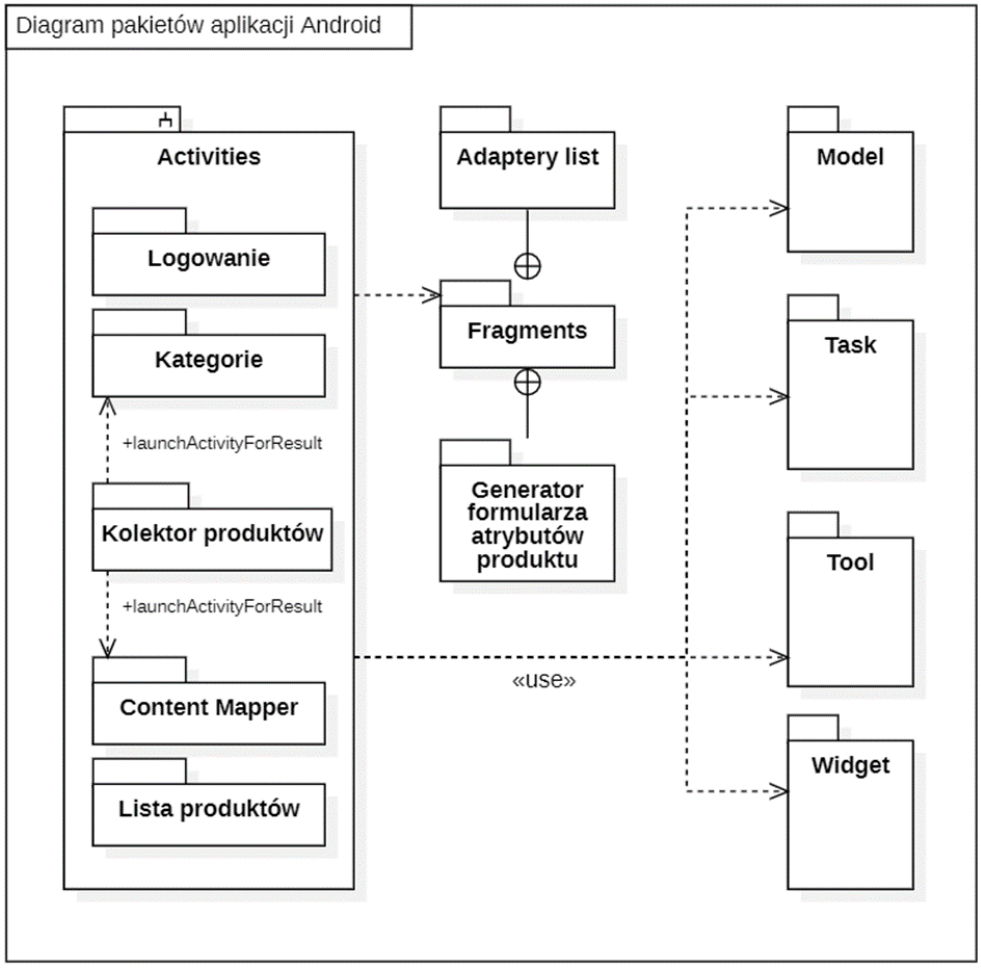
#### A. Warstwa widoku aplikacji

Rysunek 24 przedstawia diagram pakietów warstwy widoku która składa się podstawowych części wykorzystywanych przez biblioteki działającej we frontend np. Vue Framework. Są to paczki zasobów tj. style css, dodatkowe skrypty js oraz grafiki (elementy interfejsu w formacie png i svg).




Rysunek 24 Diagram pakietów warstwy widoku. Źródło: opracowanie własne

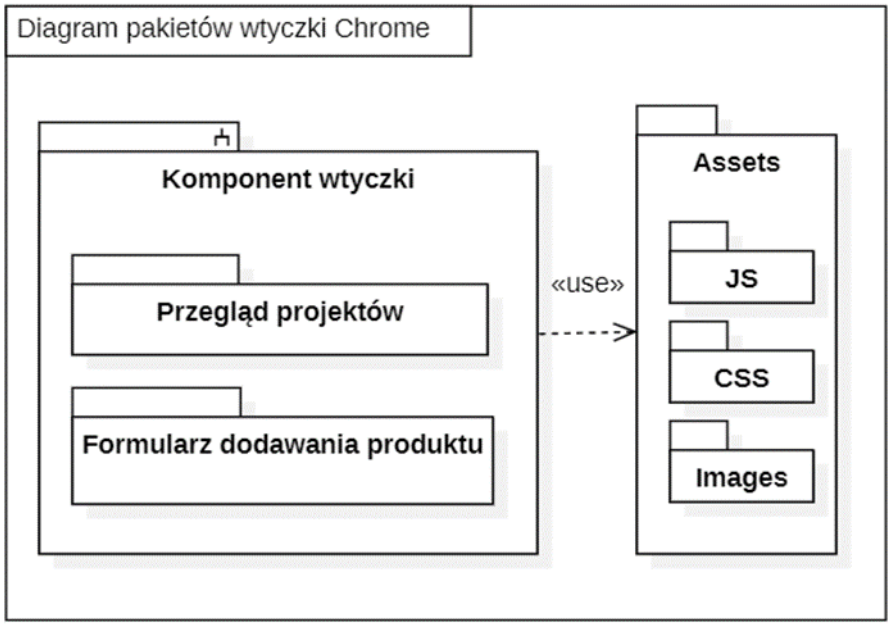
## B. Aplikacja Android




Rysunek 25 Diagram pakietów aplikacji na platformę Android. Źródło: opracowanie własne

Architekturę aplikacji mobilnej zrealizowano w oparciu o zasady tworzenia oprogramowania na platformę Android (Minimalny API Level przyjęto jako 22 co stanowi kompromis pomiędzy możliwościami implementacyjnymi nowoczesnych rozwiązań z zachowaniem obsługi dla 99% urządzeń [9]). Aktywności podzielono na ekrany spełniające daną paczkę funkcjonalności. Korzystają one z pozostałych pakietów podzielonych względem zastosowania i celu, np. klasy z pakietu *Task* są używane do wywołania działań w ramach odrębnego procesu jako *Kotlin Coroutines*. Diagram pakietów aplikacji mobilnej przedstawiono na rysunku 25.

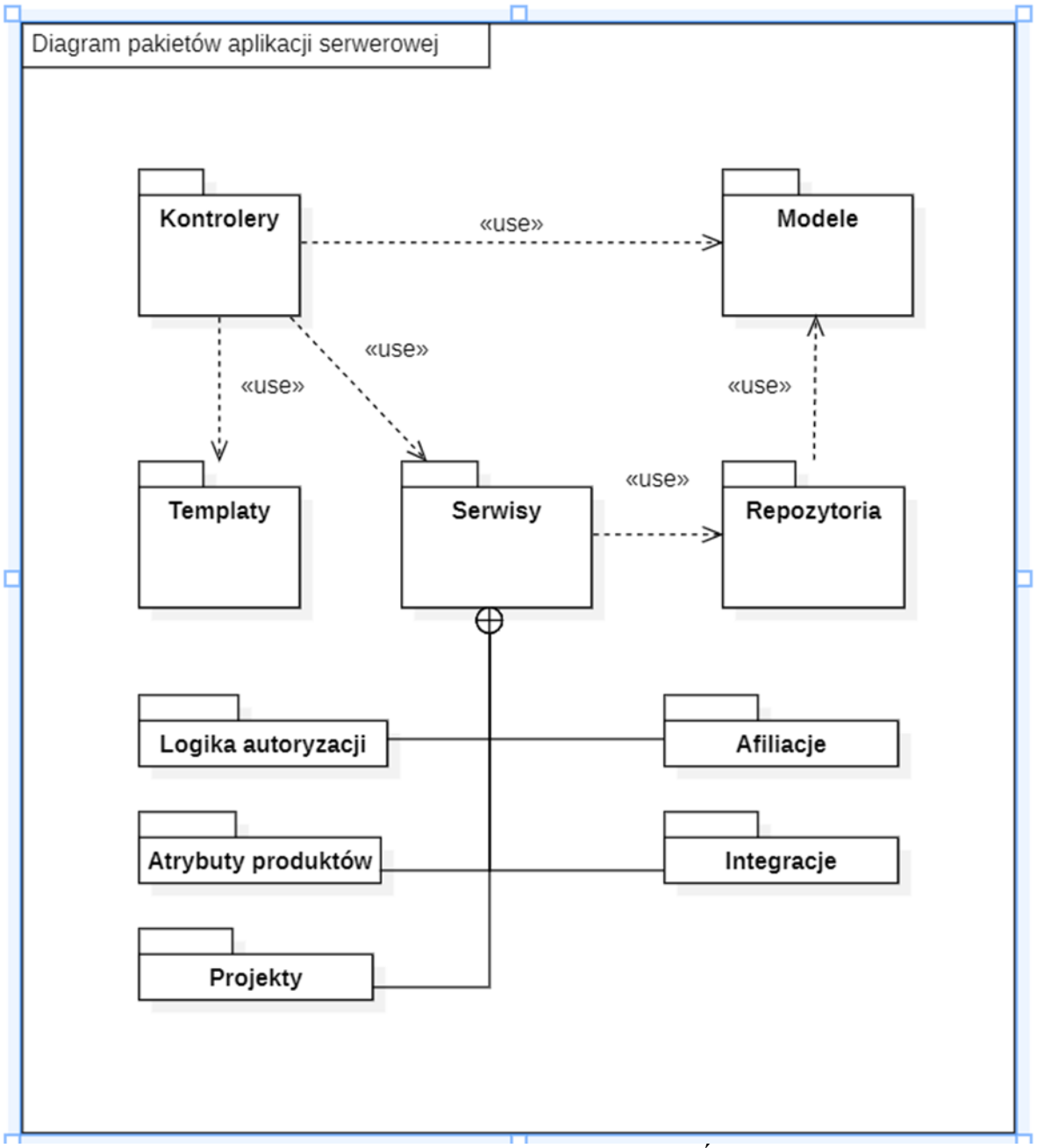
## C. Wtyczka Chrome




Rysunek 26 Diagram pakietów wtyczki Chrome. Źródło: opracowanie własne

Rysunek 26 przedstawia diagram pakietów wtyczki Chrome która podobnie jak warstwa widoku składa się ze styli css, dodatkowych skryptów js oraz grafik interfejsu użytkownika. *Core* wtyczki podzielono na 2 komponenty funkcjonalne.

### D. Serwer aplikacji internetowej

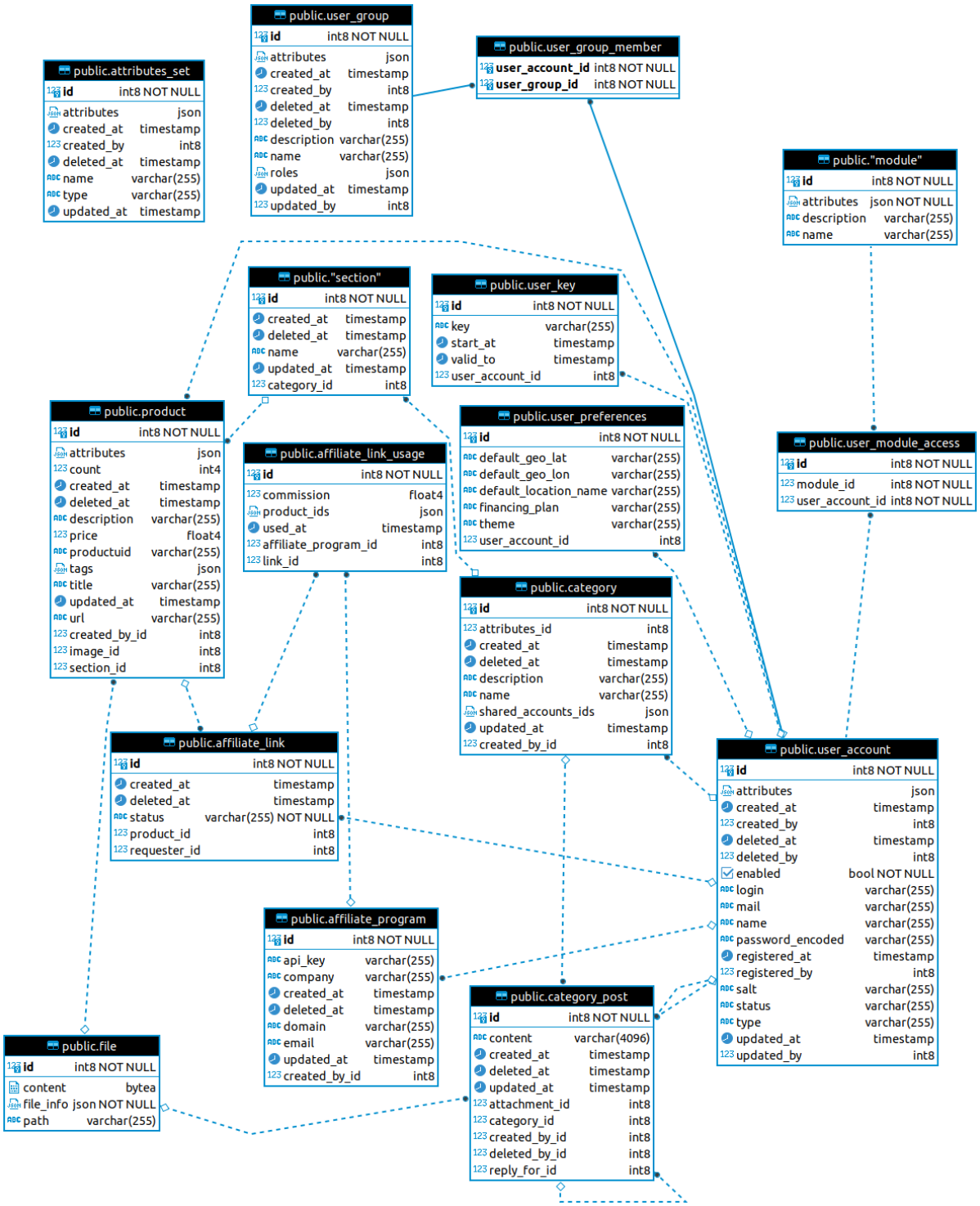



Rysunek 27 Diagram pakietów aplikacji po stronie serwera. Źródło: opracowanie własne

Diagram przedstawiony na rysunku 27 obrazuje sposób budowy aplikacji serwerowej opartej o *framework* Spring. Ich nieodłącznymi częściami są Repozytoria pozyskujące modele danych (encje) z bazy, które to później wykorzystywane są przez serwisy uruchomione w ramach kontrolera. Przewidziano tutaj redukcję kodu w kontrolerach do niezbędnych operacji, jak odbiór danych wejściowych, przekazanie odpowiedzi. Logika biznesowa została wydelegowana do serwisów.

### 5.8.3 Diagram ERD

Pełny diagram związków encji systemu przedstawiono na rysunku 28.




Rysunek 28 Diagram związków encji systemu. Źródło: opracowanie własne

# 6. Implementacja prototypu

Rozdział ten przedstawia proces wykonania pierwszej wersji aplikacji, przyjętą konfigurację systemu, zastosowany kod źródłowy kluczowy do spełnienia postawionych przed nim wymagań. Objaśnia jakie technologie zostały użyte, jaką rolę pełnią tabelę bazy danych, a także za co odpowiadają wybrane fragmenty kodu. Dokonano tutaj ekstrakcji kluczowych listingów z pominięciem algorytmów implementacyjnych powszechnie wykorzystywanych funkcjonalności jak np. logowanie użytkownika.

## 6.1. Architektura aplikacji – implementacja

W celu realizacji prototypu wykorzystano usługi maszyny Docker do definicji 2 serwisów połączonych wewnętrzną siecią „*backend*". Pozwoliły one utworzyć aplikację internetową będącą sercem całego rozwiązania.

Kontener o nazwie „*appserver*" zawiera kompletny projekt Spring, wraz z pobranymi przez narzędzie kontroli wersji (Maven) zależnościami. Podczas uruchamiania tego kontenera zachodzi ponowne przebudowanie i kompilacja solucji, a także wystawienie usługi na port 80. Drugi kontener (Listing 17) opisuje usługę bazy danych PostgreSQL.

Ponadto w celu realizacji funkcjonalności wtyczki opracowano 2 wersje programów – jedną jako rozszerzenie Chrome, a kolejną jako aplikację na system Android. Wtyczka Chrome współpracuje z Manifest 2 [39]. Standard ten określa sposób tworzenia rozszerzeń i dodatków przeglądarek internetowych (obsługiwany także przez inne popularne przeglądarki jak Firefox, Operę, Safari).

W dalszych rozdziałach opisano zastosowaną strukturę bazy danych, punkty końcowe API oraz sposoby implementacji wybranych funkcjonalności.

## 6.2. Konfiguracja bazy danych

W celu realizacji założeń projektu, wybrano rozwiązanie PostgreSQL, będące systemem zarządzania obiektowo-relacyjną bazą danych. Instrukcje, które użyto należą zarówno do zbioru *Data Manipulation Language, Data Definiton Language*, *Data Query Language*. Struktura bazy danych została wygenerowana za pomocą migracji, w oparciu o adnotacje ORM w encjach projektu.

Całość przechowywana jest w obrębie jednego *schema „public"*. Instancja serwera bazy danych została powołana jako jedna z usług Docker. Jej konfiguracja została przedstawiona w listingu 17.

Listing 17 Fragment zawartości pliku *docker-compose.yaml* opisujący usługę bazodanową. Źródło: opracowanie własne.

```
appdatabase:
  container_name: app-db
  image: postgres
  ports:
    - "5432:5432"
  environment:
    POSTGRES_DB: appserver
    POSTGRES_PASSWORD: <password>
```

```
    POSTGRES_USER: root
  volumes:
    - ./pgdata:/var/lib/postgresql/data
  networks:
    - backend
```

Komunikacja z serwerem odbywa się po porcie 5432. Za pomocą poświadczeń administracyjnych utworzono konto dedykowane dla komunikacji z aplikacją. W obrębie wewnętrznej sieci wirtualnej dostępny jest pod hostem o nazwie *appdatabase*.

## 6.3. Struktura bazy danych

W poniższym spisie przedstawiono budowę tabel bazy danych, funkcję i typy atrybutów wraz z informacją o atrybutach kluczowych. Struktura danych, zastosowana w bazie Postgres, została wygenerowana z użyciem adnotacji *frameworku* Hibernate. Dzięki wykorzystaniu tej biblioteki, wszelkie zmiany w strukturze DB są wprowadzane automatycznie. Następuje to po zmianie lub dodaniu adnotacji do atrybutu w klasie *Entity*.

### 6.3.1 Dane o użytkownikach

Tabela 13 Struktura tabeli *user_account*. Źródło: opracowanie własne

| Tabela | user_account | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator użytkownika |
| attributes | json | Zestaw dodatkowych atrybutów w formie JSON |
| created_at | timestamp without time zone | Timestamp daty utworzenia |
| created_by | bigint | Identyfikator osoby tworzącej konto |
| deleted_at | timestamp without time zone | Timestamp daty usunięcia |
| deleted_by | bigint | Identyfikator osoby usuwającej konto |
| enabled | boolean | Czy konto jest włączone/aktywne? |
| registered_at | timestamp without time zone | Timestamp daty rejestracji konta |
| updated_at | timestamp without time zone | Timestamp daty aktualizacji konta |
| updated_by | bigint | Identyfikator osoby aktualizującej dane o koncie |
| status | character varying | Status konta |
| type | character varying | Rodzaj konta |
| salt | character varying | Salt używany przez algorytm hashowania hasła |

| | | |
|---|---|---|
| login | character varying | Unikalny login użytkownika |
| mail | character varying | Mail użytkownika |
| name | character varying | Nazwa konta |
| password_encoded | character varying | Hasło w formie zakodowanej |

Tabela *user_account* (tabela nr 13) przechowuje informacje o kontach użytkowników w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator konta, atrybuty konta przechowywane w formacie JSON, daty utworzenia i usunięcia konta oraz identyfikatory osób, które te działania wykonały, informację o aktywności konta, datę rejestracji i ostatniej aktualizacji, aktualny status konta, typ konta, dodatkową wartość służącą do szyfrowania hasła, login użytkownika, adres e-mail oraz imię i nazwisko użytkownika, a także zaszyfrowane hasło.

Tabela 14 Struktura tabeli *user_group*. Źródło: opracowanie własne

| Tabela | user_group | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator grupy |
| updated_by | bigint | Identyfikator osoby aktualizującej dane o grupie |
| attributes | json | Zestaw dodatkowych atrybutów w formie JSON |
| created_at | timestamp without time zone | Timestamp daty utworzenia |
| created_by | bigint | Identyfikator osoby tworzącej grupę |
| deleted_at | timestamp without time zone | Timestamp daty usunięcia |
| deleted_by | bigint | Identyfikator osoby usuwającej grupę |
| updated_at | timestamp without time zone | Timestamp daty aktualizacji konta |
| roles | json | Pole zawiera kolekcję ról przydzielonych grupie |
| description | character varying | Opis grupy |
| name | character varying | Nazwa grupy |

Tabela *user_group* (tabela nr 14) przechowuje informacje o grupach użytkowników w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator grupy, identyfikatory osób, które aktualizowały lub utworzyły grupę oraz usunęły ją (jeśli została usunięta), dodatkowe atrybuty grupy przechowywane w formacie JSON, daty utworzenia i usunięcia grupy, daty ostatniej aktualizacji grupy, listę ról przydzielonych grupie (przechowywaną w formacie JSON), opis grupy oraz jej nazwę.

Tabela 15 Struktura tabeli *user_group_member*. Źródło: opracowanie własne

| Tabela | user_group_member | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| user_account_id | bigint | Kompozytowy PK, Identyfikator użytkownika |
| user_group_id | bigint | Kompozytowy PK, Identyfikator grupy |

Tabela *user_group_member* (tabela nr 15) przechowuje informacje o przynależności użytkowników do grup w systemie. Kolumny tabeli zawierają unikalne identyfikatory kont użytkowników i grup, do których należą użytkownicy.

Tabela 16 Struktura tabeli *user_session*. Źródło: opracowanie własne

| Tabela | user_session | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator sesji użytkownika |
| device_info | json | Dane o urządzeniu w formacie json |
| app_instance | character varying | Identyfikator instancji aplikacji (webowej, android i wtyczki na przeglądarkę Chrome) |
| user_key_id | bigint | FK, Identyfikator klucza autentykacji użytkownika |
| attributes | json | Dodatkowe atrybuty sesji w formacie json |

Tabela *user_session* (tabela nr 16) przechowuje informacje o sesjach użytkowników w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator sesji, informacje o urządzeniu, z którego korzysta użytkownik (przechowywane w formacie JSON), identyfikator instancji aplikacji, z której korzysta użytkownik, identyfikator klucza użytkownika oraz dodatkowe atrybuty sesji przechowywane w formacie JSON. Dzięki tej tabeli możliwa rejestracja logowania z urządzeń mobilnych bez plików cookie.

Tabela 17 Struktura tabeli *user_key*. Źródło: opracowanie własne

| Tabela | user_key | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator sesji użytkownika |
| start_at | timestamp without time zone | Data ważności klucza od |
| valid_to | timestamp without time zone | Data ważności klucza do |
| user_account_id | bigint | FK, Identyfikator konta użytkownika |

| | | |
|---|---|---|
| key | character varying | Wartość klucza w formie hash |

Tabela *user_key* (tabela nr 17) przechowuje informacje o kluczach użytkowników w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator klucza, datę rozpoczęcia ważności klucza, datę zakończenia ważności klucza, identyfikator konta użytkownika, do którego należy klucz oraz sam klucz. Za ich pomocą możliwe jest potwierdzenie poświadczeń z użyciem klucza API w kontekście sesji użytkownika zalogowanego z poziomu aplikacji mobilnej.

### 6.3.2 Projekty

Tabela 18 Struktura tabeli *file*. Źródło: opracowanie własne

| Tabela | file | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator pliku |
| content | bytea | Zawartość pliku |
| file_info | json | Metadane pliku |
| path | character varying | Ścieżka do pliku |

Tabela *file* (tabela nr 18) przechowuje informacje o plikach w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator pliku, jego zawartość przechowywaną w formacie bytea, dodatkowe informacje o pliku przechowywane w formacie JSON, ścieżkę do pliku. Tabela ta może być wykorzystywana do przechowywania plików w bazie danych lub do udostępniania ich użytkownikom.

Tabela 19 Struktura tabeli *category*. Źródło: opracowanie własne

| Tabela | category | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator projektu |
| created_at | timestamp without time zone | Data utworzenia projektu |
| shared_accounts_ids | json | Lista id kont użytkowników z dostępem do projektu |
| updated_at | timestamp without time zone | Data modyfikacji projektu |
| created_by_id | bigint | FK, Identyfikator tworzącego projekt |
| deleted_at | timestamp without time zone | Data usunięcia projektu |

| | | |
|---|---|---|
| attributes_id | bigint | FK, Identyfikator zestawu atrybutów |
| name | character varying | Nazwa projektu |
| description | character varying | Opis projektu |

Tabela *category* (tabela nr 19) przechowuje informacje o projektach w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator, datę utworzenia kategorii, identyfikatory kont, z którymi kategoria jest udostępniona (przechowywane w formacie JSON), datę ostatniej aktualizacji kategorii, identyfikator osoby, która utworzyła kategorię, datę usunięcia kategorii (jeśli została usunięta), identyfikator atrybutów kategorii, nazwę kategorii oraz jej opis. Do danego projektu mogą być przyporządkowane sekcje i produkty.

Tabela 20 Struktura tabeli *category_post*. Źródło: opracowanie własne

| Tabela | category_post | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator posta (komentarza do projektu/wizualizacji) |
| attachment_id | bigint | FK, Identyfikator załącznika |
| updated_at | timestamp without time zone | Data modyfikacji |
| category_id | bigint | FK, Identyfikator projektu |
| created_by_id | bigint | FK, Identyfikator osoby tworzącej |
| deleted_by_id | bigint | FK, Identyfikator osoby usuwającej |
| content | character varying | Treść |
| reply_for_id | bigint | FK, Identyfikator posta do którego tworzona jest odpowiedź |
| created_at | timestamp without time zone | Data utwotzenia |
| deleted_at | timestamp without time zone | Data usunięcia |

Tabela *category_post* (tabela nr 20) przechowuje informacje o komentarzach/wizualizacjach w projektach. Kolumny tabeli zawierają m.in. unikalny identyfikator postu, identyfikator załącznika (jeśli istnieje), datę ostatniej aktualizacji postu, identyfikator kategorii, do której należy post, identyfikator osoby, która utworzyła post, identyfikator osoby, która usunęła post (jeśli został usunięty), treść postu, identyfikator postu, na który jest odpowiedzią (jeśli jest odpowiedzią), datę utworzenia postu oraz datę usunięcia postu (jeśli został usunięty).

Tabela 21 Struktura tabeli *section*. Źródło: opracowanie własne

| Tabela | section | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator sekcji |
| name | character varying | Nazwa |
| category_id | bigint | FK, Identyfikator kategorii |
| deleted_at | timestamp without time zone | Data usuniecia |
| created_at | timestamp without time zone | Data utworzenia |

Tabela *section* (tabela nr 21) przechowuje informacje o sekcjach w projektach. Kolumny tabeli zawierają m.in. unikalny identyfikator sekcji, nazwę sekcji, identyfikator projektu, do której należy sekcja, datę usunięcia sekcji (jeśli została usunięta) oraz datę utworzenia sekcji.

Tabela 22 Struktura tabeli *product*. Źródło: opracowanie własne

| Tabela | product | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator produktu |
| productuid | character varying | Zewnętrzny, domenowy identyfikator produktu którym posługuje się baza sprzedawcy |
| created_by_id | bigint | FK, Identyfikator osoby tworzącej |
| image_id | bigint | FK, Id obrazka produktu (Tabela file) |
| section_id | bigint | FK, Identyfikator sekcji |
| price | real | Cena |
| tags | json | Tagi produktu |
| updated_at | timestamp without time zone | Data modyfikacji |
| url | character varying | Url produktu |
| title | character varying | Tytuł |
| description | character varying | Opis |
| attributes | json | Dodatkowe atrybuty produktu |

| | | |
|---|---|---|
| count | integer | Ilość |
| created_at | timestamp without time zone | Data utworzenia |
| deleted_at | timestamp without time zone | Data usuniecia |
| updated_at | timestamp without time zone | Data aktualizacji |
| name | character varying | Nazwa produktu |

Tabela *product* (tabela nr 22) przechowuje informacje o produktach w ramach projektu. Kolumny tabeli zawierają m.in. unikalny identyfikator produktu, unikalny identyfikator produktu, identyfikator osoby, która utworzyła produkt, identyfikator zdjęcia produktu (jeśli istnieje), identyfikator sekcji, do której należy produkt, cenę produktu, tagi produktu (przechowywane w formacie JSON), adres URL produktu, tytuł produktu, opis produktu, dodatkowe atrybuty produktu (przechowywane w formacie JSON), liczbę sztuk produktu w magazynie, datę utworzenia produktu, datę usunięcia produktu (jeśli został usunięty) oraz datę ostatniej aktualizacji produktu.

Tabela 23 Struktura tabeli *attributes_set*. Źródło: opracowanie własne

| Tabela | attributes_set | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Identyfikator zestawu atrybutów |
| attributes | json | Konfiguracja zestawu atrybutów w formie struktury json |
| created_by | bigint | FK, Identyfikator osoby tworzącej |
| created_at | timestamp without time zone | Data utworzenia |
| deleted_at | timestamp without time zone | Data usunięcia |
| updated_at | timestamp without time zone | Data aktualizacji |
| name | character varying | Nazwa |
| type | character varying | Rodzaj zestawu atrybutów |

Tabela *attributes_set* (tabela nr 23) przechowuje informacje o zestawach atrybutów w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator zestawu atrybutów, definicję atrybutów przechowywaną w formacie JSON, identyfikator osoby, która utworzyła zestaw atrybutów, datę utworzenia zestawu atrybutów, datę usunięcia zestawu atrybutów (jeśli został usunięty), datę ostatniej aktualizacji zestawu atrybutów, nazwę zestawu atrybutów oraz jego typ. Tabela ta jest wykorzystywana do zarządzania zestawami atrybutów w systemie, np. do ich tworzenia i przypisywania do projektów.

### 6.3.3 Programy partnerskie i afiliacje

Tabela 24 Struktura tabeli *affiliate_program*. Źródło: opracowanie własne

| Tabela | affiliate_program | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Identyfikator programu afiliacyjnego |
| updated_at | timestamp without time zone | Data modyfikacji |
| created_at | timestamp without time zone | Data utworzenia |
| deleted_at | timestamp without time zone | Data usuniecia |
| created_by_id | bigint | FK, Identyfikator konta twórcy |
| domain | character varying | Domena |
| email | character varying | E-mail |
| api_key | character varying | Klucz API do komunikacji z serwerem |
| company | character varying | Firma |

Tabela *affiliate_program* (tabela nr 24) przechowuje informacje o programach partnerskich w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator programu partnerskiego, datę ostatniej aktualizacji programu, datę utworzenia programu, datę usunięcia programu (jeśli został usunięty), identyfikator osoby, która utworzyła program, domenę programu, adres e-mail programu, klucz API programu, nazwę firmy będącej właścicielem programu. Tabela ta może być wykorzystywana do zarządzania programami partnerskimi w systemie, np. do rejestrowania nowych programów czy monitorowania ich aktywności.

Tabela 25 Struktura tabeli *affiliate_link*. Źródło: opracowanie własne

| Tabela | affiliate_link | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator linku afiliacyjnego |
| requester_id | bigint | FK, Id konta polecającego |
| status | character varying | Status linku afiliacyjnego |
| created_at | timestamp without time zone | Data utworzenia |
| deleted_at | timestamp without time zone | Data usunięcia |

| | | |
|---|---|---|
| product_id | bigint | FK, Identyfikator produktu |

Tabela *affiliate_link* (tabela nr 25) przechowuje informacje o linkach partnerskich w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator linku partnerskiego, identyfikator osoby polecającej, status linku, datę utworzenia linku, datę usunięcia linku (jeśli został usunięty) oraz identyfikator produktu, do którego jest przypisany link.

Tabela 26 Struktura tabeli *affiliate_link_usage*. Źródło: opracowanie własne

| Tabela | affiliate_link_usage | |
|---|---|---|
| Nazwa pola | Typ pola | Opis |
| id | bigint | PK, Unikalny identyfikator użycia linku afiliacyjnego |
| commission | real | Naliczona prowizja |
| link_id | bigint | FK, Identyfikator linku afiliacyjnego |
| affiliate_program_id | bigint | FK, Identyfikator programu afiliacyjnego |
| used_at | timestamp without time zone | Data użycia linku |
| product_ids | json | Lista identyfikatorów produktów z całej transakcji zakupu |

Tabela *affiliate_link_usage* (tabela nr 26) przechowuje informacje o wykorzystaniu linków partnerskich w systemie. Kolumny tabeli zawierają m.in. unikalny identyfikator wykorzystania linku, prowizję uzyskaną za jego wykorzystanie, identyfikator linku, identyfikator programu partnerskiego, do którego należy link, datę wykorzystania linku oraz identyfikatory produktów, do których odwołuje się link (przechowywane w formacie JSON). Tabela ta może być wykorzystywana do śledzenia poprawnie wykonanych transakcji zakupu oraz do rozliczania prowizji za ich wykorzystanie.

## 6.4. API

W celu realizacji zadań postawionych przed projektem przewidziano punkty końcowe API w technologii REST. Dane mogą być wymieniane i udostępnianie warstwie widoku aplikacji serwerowej, wtyczce na przeglądarkę Chrome, a także aplikacji mobilnej. Wykorzystują one kontrolery, a także powiązane z nimi serwisy oraz repozytoria do bazy danych. Komunikacja odbywa się z wykorzystaniem danych w formacie JSON.

Każdy zbiór danych zwracany jest w formacie przedstawionym w listingu 18.

Listing 18 Struktura odpowiedzi każdego żądania API w aplikacji. Źródło: opracowanie własne

```
{
    model_object: Any?,
    errors: List<String>,
    response_code: Int
}
```

W skład API wchodzą między innymi czynności i akcje zawarte w tabelach 27-33.

Tabela 27 Opis metody POST api/internal/account. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST api/internal/account |
| Parametry w URL | |
| Parametry w JSON | {<br>login: String?,<br>password_plain: String?,<br>name: String?,<br>mail: String?,<br>attributes: Map<String, Any?>,<br>user_groups: List<Long> --- Lista ID grup, do których konto ma być przypisane<br>} |
| Opis | Dodanie nowego konta użytkownika. |
| Przykładowe rezultaty | W przypadku udanej akcji dodania użytkownika:<br>{<br>model_object: 131 --- Id nowo dodanego użytkownika,<br>errors: [],<br>response_code: 200<br>}<br><br>W przypadku wystąpienia błędu walidacji:<br>{ |

| | |
|---|---|
| | model_object: null,<br>errors: [‘Pole login nie może być puste’],<br>response_code: 400<br>} |

Tabela 28 Opis metody POST /api/internal/group/{id}. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST /api/internal/group/{id} |
| Parametry w URL | Id – id istniejącej grupy |
| Parametry w JSON | {<br>name: String? = null<br>description: String? = null<br>attributes: Map<String, Any?><br>roles: List<Any><br>} |
| Opis | Edycja istniejącej grupy użytkownika. |
| Przykładowe rezultaty | W przypadku udanej akcji edycji grupy:<br>{<br>model_object: 131 --- Id zedytowanej grupy,<br>errors: [],<br>response_code: 200<br>}<br><br>W przypadku wystąpienia błędu walidacji:<br>{<br>model_object: null,<br>errors: [‘Pole nazwa nie może być puste’],<br>response_code: 400<br>} |

Tabela 29 Opis metody POST /api/internal/product. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST /api/internal/product |
| Parametry w URL | |
| Parametry w JSON | {<br>title: String?,<br>description: String?,<br>tags: List<String>,<br>section: Long?,<br>attributes: HashMap<String, Any?>,<br>url: String?,<br>price: Float?,<br>count: Int?,<br>product_uid: String?,<br>image: String?,<br>image_data: Map<String, Any?>,<br>image_url: String?<br>} |
| Opis | Dodawanie produktu z poziomu wtyczki na przeglądarkę Chrome/aplikacji internetowej. |
| Przykładowe rezultaty | W przypadku udanej akcji edycji grupy:<br>{<br>model_object: 131 --- Id nowo dodanego produktu,<br>errors: [],<br>response_code: 200<br>}<br><br>W przypadku wystąpienia błędu walidacji:<br>{<br>model_object: null,<br>errors: [‘Nie określono sekcji’], |

| | |
|---|---|
| | response_code: 400<br>} |

Tabela 30 Opis metody POST /api/internal/post. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST /api/internal/post |
| Parametry w URL | |
| Parametry w JSON | {<br>id: Long?,<br>content: String?,<br>attachment: String?,<br>attachment_data: Map<String, Any?>,<br>category: Long?,<br>reply_for: Long?<br>} |
| Opis | Dodawanie komentarza/wizualizacji do projektu. |
| Przykładowe rezultaty | W przypadku udanej akcji edycji grupy:<br>{<br>model_object: {<br>id: 101,<br>content: „Komentarz nr 1”,<br>created_at: „2020-10-10 14:14:54”<br>},<br>errors: [],<br>response_code: 200<br>}<br><br>W przypadku wystąpienia błędu walidacji:<br>{<br>model_object: null,<br>errors: [‘Nieprawidłowy załącznik.’], |

| | |
|---|---|
| | response_code: 400<br>} |

Tabela 31 Opis metody POST /api/external/affiliate-link. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST /api/external/affiliate-link |
| Parametry w URL | |
| Parametry w JSON | {<br>product: Long?<br>} |
| Opis | Zlecenie wygenerowania linku afiliacyjnego do produktu. |
| Przykładowe rezultaty | W przypadku udanej akcji:<br>{<br>model_object: {<br>id: 200,<br>link: „example.domain/product/100/?affiliate_id=10094”<br>},<br>errors: [],<br>response_code: 200<br>}<br><br>W przypadku braku wskazanego produktu:<br>{<br>model_object: null,<br>errors: [‘Brakujący produkt.’],<br>response_code: 404<br>} |

Tabela 32 Opis metody POST /api/external/affiliate-link/action/use. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | POST /api/external/affiliate-link/action/use |

| | |
|---|---|
| Parametry w URL | |
| Parametry w JSON | {<br>link: Long?,<br>commission: Float,<br>productIds: MutableList<String>,<br>domain: String?<br>} |
| Opis | Rejestracja eventu „*purchase*” przez kod trackujący zintegrowany ze stroną sklepu. |
| Przykładowe rezultaty | {<br>model_object: 134 ------ Id zarejstrowanego użycia linku/transakcji<br>errors: [],<br>response_code: 200<br>} |

Tabela 33 Opis metody GET /api/internal/affiliate-statistics/{from}/{to}. Źródło: opracowanie własne

| | |
|---|---|
| Metoda | GET /api/internal/affiliate-statistics/{from}/{to} |
| Parametry w URL | from – data od której widoczne jest zestawienie statystyk<br>to – data graniczna statystyk |
| Parametry w JSON | |
| Opis | Pobranie statystyk na temat użycia linków afiliacyjnych w pewnym okresie czasu. |
| Przykładowe rezultaty | {<br>model_object: {<br>user: {},<br>affiliateProgram: {},<br>fromDatetime: „2021-11-23 00:00:00”,<br>toDatetime: „2025-11-23 00:00:00”,<br>userLinksData: {}, |

| | |
|---|---|
| | companyLinksData: {}<br>},<br>errors: [],<br>response_code: 200<br>} |

## 6.5. Wybrane aspekty implementacji aplikacji internetowej

W tym podrozdziale przedstawiono fragmenty kodu zaimplementowanego prototypu, które są najbardziej istotne z punktu widzenia kluczowych funkcjonalności aplikacji internetowej.

### A. Kod integracyjny

Ważną częścią aplikacji jest skrypt przystosowany do zamieszczenia na stronie zewnętrznej, którego rolą jest śledzenie ruchu użytkownika korzystającego z linku afiliacyjnego. Jego zawartość przedstawiono w listingu 19 oraz 20.

Listing 19 Zawartość afiliacyjnego skryptu js – klasy zdarzeń. Źródło: opracowanie własne

```
class AffiliateLinkManager {
    constructor() {
        this.blankCookieData = getAffiliateLinksCookie()
        this.apiKey = window.apiKey
    }

    save() {
        saveAffiliateLinksCookie(this.blankCookieData)
    }

    hasLink() {
        return this.blankCookieData.affiliateLink != null
    }

    addLink(id) {
        (…)
    }

    consumeLink(commission, productIds) {
        (..)
    }
}

class LandingPageEvent {
    support() {
        (…)
    }
```

```
    execute() {
        let linkIdGetParam = getUrlParameter("alid")
        if (! window.affiliateLinkManagerInstance.hasLink()) {

window.affiliateLinkManagerInstance.addLink(linkIdGetParam)
        }
    }
}

class PurchaseEvent {
    support() {
        (…)
    }

    execute() {
        let event = window.eventLayer.find(event => event.type ===
'purchase')
        let commision = event.commission
        let productIds = event.productIds

        if (window.affiliateLinkManagerInstance.hasLink()) {
            window.affiliateLinkManagerInstance.consumeLink(commision,
productIds)
        }
    }
}
```

Listing 20 Zawartość afiliacyjnego skryptu js – zarys ogólny. Źródło: opracowanie własne

```
function getUrlParameter(sParam) {
    let sPageURL = window.location.search.substring(1),
        sURLVariables = sPageURL.split('&'),
        sParameterName,
        i;

    for (i = 0; i < sURLVariables.length; i++) {
        sParameterName = sURLVariables[i].split('=');

        if (sParameterName[0] === sParam) {
            return sParameterName[1] === undefined ? true :
decodeURIComponent(sParameterName[1]);
        }
    }
    return false;
}

(…)

function getAffiliateLinksCookie() {
    let cookie = Cookies.get(cookieName)
    return cookie ? JSON.parse(cookie) : Object.assign({},
blankCookieData)
}

function saveAffiliateLinksCookie(data) {
    Cookies.set(cookieName, JSON.stringify(data))
}
```

```
class EventRegistry {
    getEvents() {
        return [new LandingPageEvent(), new PurchaseEvent()]
    }
}

$(document).ready(() => {
    window.eventLayer = window.eventLayer || [];
    window.affiliateLinkManagerInstance = new AffiliateLinkManager();

    (new EventRegistry()).getEvents().forEach(eventHandler => {
        if (eventHandler.support()) {
            eventHandler.execute()
        }
    });
})
```

Listing 20 zawiera kilka klas i funkcji związanych z zarządzaniem zdarzeniami afiliacyjnymi. Funkcja `getUrlParameter` służy do pobrania wskazanego (argumentem wejściowym) parametru z bieżącego adresu URL. Robi to poprzez najpierw pobranie ciągu zapytania GET bieżącego adresu URL (części po znaku ?), a następnie jego konwersję na tablicę par klucz-wartość poprzez podzielenie go na znak &. Parser kodu przechodzi przez tę tablicę, szukając klucza, który pasuje do parametru określonego jako argument dla funkcji. Jeśli znajdzie dopasowanie, zwraca odpowiadającą mu wartość lub `true`, jeśli wartość nie jest określona. Jeśli nie znajdzie dopasowania, zwraca `false`.

Funkcja `getAffiliateLinksCookie` pobiera ciasteczko o nazwie określonej w zmiennej `cookieName`, parsuje jego wartość jako JSON i zwraca wynikowy obiekt. Jeśli ciasteczko nie istnieje, zwraca obiekt z właściwością `affiliateLink` ustawioną na *null*. Funkcja `saveAffiliateLinksCookie` przyjmuje obiekt zawartości ciasteczka jako argument i zamienia go na ciąg JSON, zanim zapisze go jako ciasteczko o nazwie określonej w zmiennej `cookieName`.

Klasa `AffiliateLinkManager` (listing 19) służy do zarządzania linkami partnerskimi. Ma kilka metod:

- `save`: zapisuje bieżący stan obiektu `blankCookieData` do ciasteczka;
- `hasLink`: zwraca `true`, jeśli właściwość `affiliateLink` obiektu `blankCookieDat`a nie jest `null`, `false` w przeciwnym razie;
- `addLink`: wysyła żądanie HTTP GET do punktu końcowego prototypu odpowiadającego za rejestrację linków i ustawia właściwość `affiliateLink` obiektu `blankCookieData` na wartość właściwości `model_object` odpowiedzi, jeśli żądanie jest pomyślne. Jeśli `request` nie powiedzie się, wyświetla komunikat błędu dla każdego elementu własności `errors` odpowiedzi;
- `consumeLink`: wysyła żądanie HTTP POST do określonego adresu URL z ciałem żądania JSON zawierającym właściwości `link, commission, productIds` i `api_key`. Jeśli żądanie jest pomyślne, ustawia właściwość `affiliateLink` obiektu `blankCookieData` na `null` i zapisuje stan obiektu `blankCookieData` do ciasteczka. Jeśli żądanie nie powiedzie się, wyświetla komunikat błędu dla każdego elementu właściwości `errors` odpowiedzi.

Celem klasy *LandingPageEvent* jest rejestrowanie zdarzenia, gdy użytkownik trafia na stronę docelową z linku partnerskiego. Ma pojedynczą metodę *support*, która zwraca *true*, jeśli adres URL zawiera parametr alid, a *false* w przeciwnym razie. Ma również metodę *execute*, która, jeśli bieżąca właściwość *affiliateLink* obiektu *blankCookieData* jest *null*, wysyła żądanie HTTP GET do określonego adresu URL i ustawia właściwość *affiliateLink* obiektu *blankCookieData* na wartość właściwości *model_object* odpowiedzi, jeśli żądanie jest pomyślne.

Klasa *PurchaseEvent* służy do śledzenia zdarzenia, gdy użytkownik dokonuje zakupu. Ma pojedynczą metodę *support*, która zwraca *true*, jeśli zmienna *window.eventLayer* istnieje i zawiera obiekt o właściwości *type* o wartości *purchase* lub *false* w przeciwnym razie. Ma również metodę *execute*, która pobiera obiekt o właściwości *type* o wartości *purchase* z zmiennej *window.eventLayer*, pobiera jego właściwości *commission* i *productIds*, a następnie jeśli właściwość *affiliateLink* obiektu *blankCookieData* nie jest *null*, wysyła żądanie HTTP POST do określonego adresu URL z ciałem żądania JSON zawierającym te właściwości oraz właściwość link i *api_key*. Jeśli żądanie jest pomyślne, ustawia właściwość *affiliateLink* obiektu *blankCookieData* na *null* i zapisuje stan obiektu *blankCookieData* do ciasteczka. Jeśli żądanie nie powiedzie się, wyświetla komunikat błędu dla każdego elementu właściwości *errors* odpowiedzi.

Klasa *EventRegistry* ma jedynie metodę *getEvents*, która zwraca tablicę zawierającą obiekty klas *LandingPageEvent* i *PurchaseEvent*.

Na końcu pliku znajduje się sekcja zawierająca kod wywoływany po załadowaniu strony (po zakończeniu ładowania drzewa DOM). Tworzy ona instancję klasy *AffiliateLinkManager* i przypisuje ją do zmiennej globalnej *window.affiliateLinkManagerInstance*. Następnie tworzy instancję klasy *EventRegistry* i pobiera tablicę zdarzeń za pomocą jej metody *getEvents*. W kolejnym kroku przechodzi przez tę tablicę i dla każdego zdarzenia sprawdza, czy jest obsługiwane (poprzez wywołanie jego metody *support*), a jeśli tak, to wywołuje jego metodę *execute*.

Opisany kod stanowi mechanizm umożliwiający śledzenie i zarządzanie linkami partnerskimi na stronie internetowej. Używa on bibliotek *jQuery* i *js-cookie* do wysyłania żądań HTTP i przechowywania danych w ciasteczkach, a także do obsługi zdarzeń związanych z wejściem na stronę z linku partnerskiego i dokonaniem zakupu.

Eventem zwrotnym, który sprzedawca zobowiązany jest zamieścić wewnątrz znacznika script, przypisując pod zmienną *window.eventLayer*, jest kod w tagu script na stronie zakupu (*purchase page*) przedstawiony w listingu 21.

Listing 21 Zawartość struktury eventu zamieszczonego na stronie *purchase-page*. Źródło: opracowanie własne

```
<script type="application/javascript">
        window.eventLayer = [{
            type: "purchase",
            commission: float,
            productIds: []
        }]
</script>
```

W listingu nr 21 zastosowano tablicę *productIds*, która oznacza zbiór wewnętrznych id produktów zakupionych przez użytkownika w ramach transakcji zakończonej sukcesem.

**B. Edytor zestawu atrybutów**

Listing 22 Fragment kodu odpowiadający za komponent edytora atrybutów. Źródło: opracowanie własne

```
<template>
    (…)
</template>

<script>
    import AttributeItem from "../../components/editor/AttributeItem";

    (…)

    export default {
        name: 'AttributesSetForm',
        components: {
            AttributeItem
        },
        data() {
            return {
                attributes_set: Object.assign({}, emptyAttributesSet),
                items_schema: {},
                id: window.id,
                new_attribute_item: Object.assign({},
newAttributeItem)
            }
        },
        methods: {
            loadAttributesSchema() {
                (…)
            },
            save() {
                (…)
            },
            add() {
                (…)
            },
            edit() {
                (…)
            },
            validate() {
                (…)
            },
            redirectToListUrl() {
                (…)
            },
            loadItemsSchema() {
                let url = this.itemsSchemaUrl;
                $.get(url).then((response) => {
                    this.items_schema = response
                })
            },
            addNewAttribute() {
```

```
                if (!this.attributes_set.attributes.schema) {
                    this.attributes_set.attributes.schema = [];
                }


this.attributes_set.attributes.schema.push(this.new_attribute_item)
                this.new_attribute_item = Object.assign({},
newAttributeItem)
            },
            deleteAttribute(idx) {
                this.attributes_set.attributes.schema.splice(idx, 1)
            },
            addNewTagForAttribute(idx) {
                if (! this.attributes_set.
                   attributes.schema[idx].defaultTags) {
                    this.attributes_set.attributes.
                        schema[idx].defaultTags = []
                }

                this.attributes_set.attributes
                   .schema[idx].defaultTags
                   .push(Object.assign({}, newTag))
            },
            removeAttributeTag(idx, tagIdx) {
                this.attributes_set.attributes.schema[idx].
                   defaultTags.splice(tagIdx, 1)
            }
        },
        computed: {
            (…)
        },
        mounted() {
            this.loadItemsSchema()
            if (this.id) {
                this.loadAttributesSchema()
            }
        }
    };
</script>
```

Komponent (listing nr 23) w technologii Vue.js, jest szablonem, który służy do tworzenia lub edycji zestawu atrybutów. Formularz składa się z kilku sekcji, w tym sekcji do wprowadzania nazwy zestawu, sekcji do dodawania nowych atrybutów do zestawu oraz sekcji do edytowania istniejących atrybutów. Formularz zawiera również przyciski do zapisywania lub anulowania zmian. Komponent ma również kilka metod, takich jak *"addNewAttribute()"* i *"deleteAttribute()"* do dodawania lub usuwania atrybutów z zestawu oraz *"save()"* do zapisywania zmian w zestawie na serwerze. Ważną częścią jest implementacja różnych rodzajów atrybutów, których typ posiada własny dedykowany komponent uruchamiany przez komponent nadzorczy *AttributeItem*.

Przykładem takiego komponentu może być komponent *SelectItem*, którego zawartość została zaprezentowana w listingu 23.

Listing 23 Implementacja komponentu *SelectItem*. Źródło: opracowanie własne

```
<template>
(…)
</template>

<script>
    (…)

    export default {
        name: 'SelectItem',
        props: {
            item: {
                type: Object,
                default: {
                    fieldKey: null,
                    fieldName: null,
                    fieldType: "select",
                    fieldSubType: null,
                    model: null,
                    additionalAttributes: {}
                }
            }
        },
        data() {
            return {
                new_option: Object.assign({}, newOption)
            }
        },
        methods: {
            addOption() {
                if (! Array.isArray(this.item.model)) {
                    this.item.model = []
                }

                this.item.model.push(Object.assign({},
this.new_option))
            },
            deleteOption(idx) {
                this.item.model.splice(idx, 1)
            }
        },
        computed: {
        }
    };
</script>
```

W listingu nr 23 przedstawiono komponent, który służy do tworzenia elementu listy wyboru. Element składa się z dwóch sekcji - sekcji do dodawania nowych opcji oraz sekcji do edytowania istniejących opcji. W sekcji do dodawania nowych opcji użytkownik może wprowadzić klucz i opcję, a następnie dodać ją do listy za pomocą przycisku "plus". W sekcji do edytowania istniejących opcji użytkownik może edytować klucz i opcję lub usunąć opcję za pomocą przycisku "minus". Komponent ma również kilka metod, takich jak *"addOption()"* do dodawania nowych opcji oraz *"deleteOption()"* do usuwania istniejących opcji.

### C. Generowanie linków afiliacyjnych

Listing 24 Fragment serwisu odpowiadającego za generowanie linków afiliacyjnych. Źródło: opracowanie własne

```
@Service
class AffiliateLinkGenerateCmd @Autowired constructor(
    (…)
) {
    fun execute(request: GenerateAffiliateLinkRequest):
ValidationResult {
        val validationResult = validator.validate(request)
        if (! validationResult.isOk()) {
            return validationResult
        }

        val userAccount = securityManagerSrv.authenticatedUser() as
UserAccount
        val userAP =
affiliateProgramRepository.findAffiliateProgramByCreatedByIdAndDeletedA
tIsNull(userAccount.id as Long)
        if (userAP.isEmpty) {

validationResult.addError(ErrorPayload("missing_affiliate_program"))
            validationResult.responseCode = 404
            return validationResult
        }

        val product = productRepository.findById(request.product as
Long)
        if (product.isEmpty) {
            validationResult.addError(ErrorPayload("missing_product"))
            validationResult.responseCode = 404
            return validationResult
        }

        productRepository.findById(request.product as
Long).ifPresentOrElse({
            val newAL = affiliateLinkFactorySrv.create(userAP.get(),
product.get(), request, userAccount)
            affiliateLinkRepository.save(newAL)

            val resultMap: MutableMap<String, Any?> = hashMapOf(
                "id" to newAL.id,
                "link" to (newAL.product?.url ?: "")
            )

            validationResult.modelObject = resultMap
        }, {
            validationResult.addError(ErrorPayload("missing_product"))
            validationResult.responseCode = 404
        })

        return validationResult
    }
}
```

Kod pokazany w listingu 24 stanowi implementację klasy serwisu w języku Kotlin, która jest odpowiedzialna za generowanie linków partnerskich. Klasa zawiera kilka zależności, takich jak obiekty

do obsługi linków partnerskich, programów partnerskich, produktów oraz mechanizmów bezpieczeństwa. Klasa zawiera również metodę *"execute()"*, która jest odpowiedzialna za wywołanie walidatora i sprawdzenie, czy żądanie jest poprawne. Następnie metoda sprawdza, czy użytkownik posiada program partnerski oraz czy produkt o podanym ID istnieje. Jeśli oba warunki są spełnione, metoda tworzy nowy obiekt linku partnerskiego i zapisuje go w bazie danych a następnie na jego podstawie przygotowana zostaje struktura JSON odpowiedzi. W przeciwnym razie metoda zwraca odpowiedni kod błędu.

Metoda przyjmuje argument w postaci obiektu DTO, który w zasadzie zawiera jedno pole wskazujące na id produktu w projekcie użytkownika. Tworzony jest on przez kontroler wywołujący ten serwis. Kod tego obiektu przedstawiono poniżej (listing nr 25).

Listing 25 Przykład obiektu DTO odpowiadające za dostarczenie danych żądania przesłanych do serwera. Źródło: opracowanie własne

```
class GenerateAffiliateLinkRequest : Serializable, AutoMappableTrait {
    @NotNull
    var product: Long? = null

    companion object {
        fun create(product: Long): GenerateAffiliateLinkRequest {
            (...)
        }
    }
}
```

**D. Rejestracja produktu przez wtyczkę**

Klasa *ProductAddCmd* (listing nr 26) jest serwisem, który jest używany do dodawania nowego produktu. W celu dodania nowego produktu, klasa ta wywołuje kilka innych serwisów i repozytoriów, w celu zwalidowania i przetworzenia danych wejściowych oraz zapisania nowego produktu do bazy danych. Główne zadania tej klasy to: walidacja danych wejściowych za pomocą *validator*, pobranie sekcji produktu z repozytorium *sectionRepository*, utworzenie nowego produktu za pomocą *productFactorySrv*, zapisanie obrazka produktu do repozytorium *fileRepository*, oraz zapisanie nowego produktu do repozytorium *productRepository*.

Listing 26 Fragment serwisu *ProductAddCmd*. Źródło: opracowanie własne

```
@Service
class ProductAddCmd @Autowired constructor(
    private val productFactorySrv: ProductFactorySrv,
    private val sectionRepository: SectionRepository,
    private val productRepository: ProductRepository,
    private val securityManagerSrv: SecurityManagerSrv,
    private val validator: SmartInvalidRequestValidator,
    private val fileRepository: FileRepository
) {
    fun execute(request: AddProductRequest): ValidationResult {
        val validationResult = validator.validate(request)
        if (! validationResult.isOk()) {
            return validationResult
        }
```

```
        sectionRepository.findById(request.section as
Long).ifPresentOrElse({
            val userAccount = securityManagerSrv.authenticatedUser()
as UserAccount
            val product = productFactorySrv.create(request, it,
userAccount)

            if (request.image != null ||
!request.image_url.isNullOrEmpty()) {
                val imageData = File()
                imageData.path = request.image_url
                request.image?.let { file ->
                    val repFile = file.replace("\n", "")
                    val decoded: ByteArray =
Base64.getDecoder().decode(repFile);

                    val fileInfo = JSONObject()
                    fileInfo["name"] =
request.image_data.getOrDefault("name", null)
                    fileInfo["content_type"] =
request.image_data.getOrDefault("content_type", null)
                    fileInfo["original_filename"] =
request.image_data.getOrDefault("original_filename", null)
                    fileInfo["size"] = decoded.size
                    imageData.content = decoded
                    imageData.fileInfo = fileInfo
                }

                fileRepository.save(imageData)
                product.image = imageData
            }

            productRepository.save(product)
            validationResult.modelObject = product.id
        }, {
            validationResult.addError(ErrorPayload("missing_section"))
            validationResult.responseCode = 404
        })

        return validationResult
    }
}
```

## 6.6. Wybrane aspekty implementacji wtyczki do przeglądarki Chrome

W tym podrozdziale przedstawiono fragmenty kodu zaimplementowanego prototypu, które są najbardziej istotne z punktu widzenia kluczowych funkcjonalności wtyczki Chrome.

### A. Pobieranie wartości domyślnych atrybutów

Listing 27 Fragment funkcji *scrapData*. Źródło: opracowanie własne

```
scrapData(url) {
      $.ajax({
        type: "GET",
```

```
        crossDomain: true,
        url,
        success: (result) => {
          var resultScrapData = Object.assign({}, scrapData);
          resultScrapData.url = url

          $('[property="og:name"],
[property="og:title"]').each(function () {
            var value = $(this).attr("content")
            resultScrapData.title = value || resultScrapData.title
          })

          $('[property="og:image"]').each(function () {
            var value = $(this).attr("content")
            resultScrapData.image = value || resultScrapData.image
          })

          $('meta[property="og:price:amount"], meta[itemprop="price"],
meta[property="product:price:amount"]').each(
              function () {
                var value = $(this).attr("content")
                resultScrapData.price = (parseFloat(value) ?? value)
|| resultScrapData.price
              }
          )

          $('[property="og:description"]').each(function () {
            var value = $(this).attr("content")
            resultScrapData.description = value ||
resultScrapData.description
          })

          resultScrapData = this.parseSchema(resultScrapData)
          resultScrapData = this.scrapAttributes(resultScrapData)

          this.product.title = resultScrapData.title
          this.product.image_url = resultScrapData.image
          this.product.description = resultScrapData.description
          this.product.price = resultScrapData.price
          this.product.url = url
          this.product.attributes = resultScrapData.attributes

          console.log(this.product)
        }
      })
    },
```

Kod zawarty w listing nr 27 definiuje funkcję o nazwie `scrapData`, która przyjmuje jeden argument – url – który oznacza URL przetwarzanej strony produktu, który zostanie dodany za pomocą wtyczki. Następnie wywoływana jest metoda *ajax* z biblioteki *jQuery,* która wysyła żądanie HTTP typu GET do podanego url-a. W przypadku sukcesu, wywoływana jest funkcja `success`, która dostaje jako argument `"result"` - odpowiedź serwera. Następnie tworzony jest obiekt `resultScrapData`, który jest kopią obiektu `scrapData` i nadpisywana jest jego właściwość `url` na podany argument. Następnie wyszukiwane są elementy html za pomocą selektorów css i nadpisywane są odpowiednie właściwości obiektu `resultScrapData`. Na końcu obiekt `product` jest nadpisywany wartościami z obiektu `resultScrapData`.

Poniższe selektory odpowiadają za:

- `[property="og:name"],[property="og:title"]` – Nazwę produktu;
- `[property="og:image"]` – Obrazek poglądowy produktu;
- `meta[property="og:price:amount"],meta[itemprop="price"],meta[property="product:price:amount"]` – Cenę jednostkowa produktu;
- `[property="og:description"]` – Opis.

Jednak w wielu przypadkach, to jest nie wystarczające, przez co istnieje potrzeba uzupełnienia danych, które zawarte są w zamieszczonej na stronie strukturze LD-JSON (według standardu Schema.org), co przedstawiono w listingu nr 28. Przykładem tego typu danych może być struktura przedstawiona w ramach listingu nr 29.

Listing 28 Fragment funkcji *parseSchema*. Źródło: opracowanie własne

```
parseSchema(scrapData) {
      $('script[type="application/ld+json"]').each(function () {
        var rawJson = $(this).html()
        if (! rawJson) {
          return scrapData
        }
        var schemaJson = JSON.parse(rawJson)

        var schemaItems =
            (schemaJson["@context"] === "https://schema.ord" &&
schemaJson["@graph"]) ?
                schemaJson["@graph"] : [schemaJson]

        schemaItems.forEach((schemaItem) => {
          if (schemaItem["@type"] === "Product") {
            scrapData.title = schemaItem.name ?? scrapData.title
            scrapData.description = schemaItem.description ??
                scrapData.description
            scrapData.price = (schemaItem.offers &&
schemaItem.offers.price) ?
                parseFloat(schemaItem.offers.price) ??
schemaItem.offers.price :
                scrapData.price
            scrapData.image = Array.isArray(schemaItem.image) ?
                schemaItem.image[0] ?? scrapData.image :
                schemaItem.image
          }
        })
      })

      return scrapData
    },
```

Listing 29 Przykład danych schema.org w formacie json-ld umieszczonej na przetwarzanej stronie internetowej. Źródło: opracowanie własne

```
<script type="application/ld+json">
{
```

```
  "@context": "https://schema.org/",
  "@type": "Product",
  "name": "Example",
  "image": "https://example.domain/image/101",
  "description": "Desc",
  "brand": {
    "@type": "Brand",
    "name": "MainBrand"
  },
  "offers": {
    "@type": "Offer",
    "url": "",
    "priceCurrency": "PLN",
    "price": "145.5",
    "availability": "https://schema.org/InStock",
    "itemCondition": "https://schema.org/NewCondition"
  }
}
</script>
```

**B. Formularz atrybutów**

Analogicznie do konstruktora atrybutów, komponent wtyczki posiada zarządcę komponentów elementu formularza, odpowiadającego temu atrybutowi. W ciele wtyczki powołano `AttributesItemLoader`, który powołuje odpowiedni element GUI odpowiadający danemu rodzajowi atrybutu. Przykładem może być komponent `AttributesItemLoader` z listingu nr 30.

Listing 30 Zawartość komponentu wywołanego przez *AttributesItemLoader - SelectItem*. Źródło: opracowanie własne

```
<template is="div">
  <label class="text-black-50 text-uppercase small mt-3">{{
attributesSetItem.fieldName }}</label>

  <div class="row mt-1">
    <div class="col-12">
      <select class="w-100 pc-select" v-
model="attributesModel[attributesSetItem.fieldKey]">
        <option :value="null"
disabled>{{attributesSetItem.fieldName}}</option>
        <option
            v-for="option in attributesSetItem.model"
            :value="option.key"
            :key="option.key"
            v-text="option.label"
        ></option>
      </select>
    </div>
  </div>
</template>

<script>
export default {
  props: {
    attributesSetItem: {
      type: Object
    },
```

```
    attributesModel: {
      type: Object
    }
  }
}
</script>
```

W listing nr 30 przedstawiono sposób implementacji komponentu odpowiadającego za pole typu „lista wyboru”. Przyjmuje on 2 cechy „*props*”, które stanowią: opis jak przedstawić dane pole (*attributesSetItem*) oraz zbiór dodatkowych atrybutów produktów w formacie JSON (*attributesModel*).

Wersja bazowa zakłada istnienie poniższych typów atrybutów:

- *TextInput*;
- *ImageInput*, w 2 wariantach różniących się sposobem zapisu danych tj. zapis linku, a także przesłanie obrazu z dysku;
- *NumberInput*;
- *SelectInput*.

## 6.7. Wybrane aspekty implementacji aplikacji na system Android

W tym podrozdziale przedstawiono fragmenty kodu zaimplementowanego prototypu, które są najbardziej istotne z punktu widzenia kluczowych funkcjonalności aplikacji Android.

### A. Odczyt wartości dodatkowych atrybutów

Jedną z ważnych części aplikacji (zarówno aplikacja mobilna jak i wtyczka na przeglądarkę Chrome) jest możliwość automatycznego odczytu wartości dodatkowych atrybutów. Odbywa się to z wykorzystaniem zestawu skojarzonych tagów. Proces ten (listing nr 31) odbywa się z wykorzystaniem biblioteki *Jsoup* [10], która znacząco ułatwia przetwarzanie znaczników w otrzymanym dokumencie HTTP.

Listing 31 Fragment metody *scrapAttributes*. Źródło: opracowanie własne

```
fun scrapAttributes(doc: Document) {
        categoryDto?.attributes_set?.attributes?.schema?.forEach {
            it.defaultTags.forEach { tag ->
                val tagSelector: String = tag["name"] ?: ""
                val selectedTags = doc.select(tagSelector)
                selectedTags.forEach { htmlTag ->
                    attributesDataRef.addProperty(it.fieldKey ?: "",
                        htmlTag.attr(tag["attribute"] ?: ""))
                }
```

```
            }
        }
    }
```

### B. Ładowanie treści strony do komponentu WebView

W celu umożliwienia użytkownikowi wybrania danych fragmentów tekstu i zastosowania go jako wartość pewnego atrybutu, dodano komponent *`WebView`*, który to umożliwia. Kod odpowiadający za jego realizację został przedstawiony w listingu 32.

Listing 32 Fragment aktywności *ContentSelectorActivity*. Źródło: opracowanie własne

```
class ContentSelectorActivity : AppCompatActivity() {
    lateinit var binding: ActivityContentSelectorBinding
    lateinit var url: String

    var fieldName: String = ""

    @SuppressLint("SetJavaScriptEnabled")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding                                                    =
ActivityContentSelectorBinding.inflate(layoutInflater)

        url = intent.getStringExtra("url").toString()
        if (intent.hasExtra("field_name")) {
            fieldName = intent.getStringExtra("field_name").toString()
            binding.acsValueLayout.hint = fieldName
        }

        binding.acsPageWebView.settings.javaScriptEnabled = true
        binding.acsPageWebView.loadUrl(url)

        binding.acsGrabSelection.setOnClickListener {

binding.acsPageWebView.evaluateJavascript("(function(){return
window.getSelection().toString()})()", JavaScriptCallback(binding))
        }

        binding.acsAccept.setOnClickListener {
            val resultIntent = Intent()
            resultIntent.putExtra("result",
binding.acsValueEditText.text.toString())
            setResult(RESULT_OK, resultIntent)
            finish()
        }

        setContentView(binding.root)
    }

    class      JavaScriptCallback      (private      val      binding:
ActivityContentSelectorBinding) : ValueCallback<String> {
        override fun onReceiveValue(value: String?) {
```

```
            if (! value.isNullOrEmpty()) {
                binding.acsValueEditText.text = Editable.Factory

.getInstance().newEditable(value.dropLast(1).drop(1))
            }
        }
    }
}
```

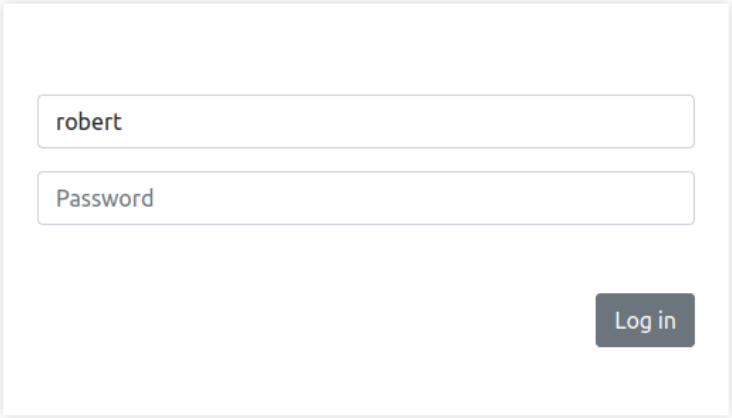
Jego zasada działania opiera się o wykorzystanie *callback* w języku js, na wybranym elemencie, który zwraca do aplikacji android rezultat w postaci zmiennej tekstowej. *Intent* uruchamiający to *activity* otrzymuje informację o tym, które pole będzie aktualizowane. *Property text* zmiennej `binding.acsValueEditText` zostaje aktualizowane o tą wartość i przedstawione użytkownikowi celem akceptacji lub zmiany zaznaczenia.

# 7. Prezentacja aplikacji

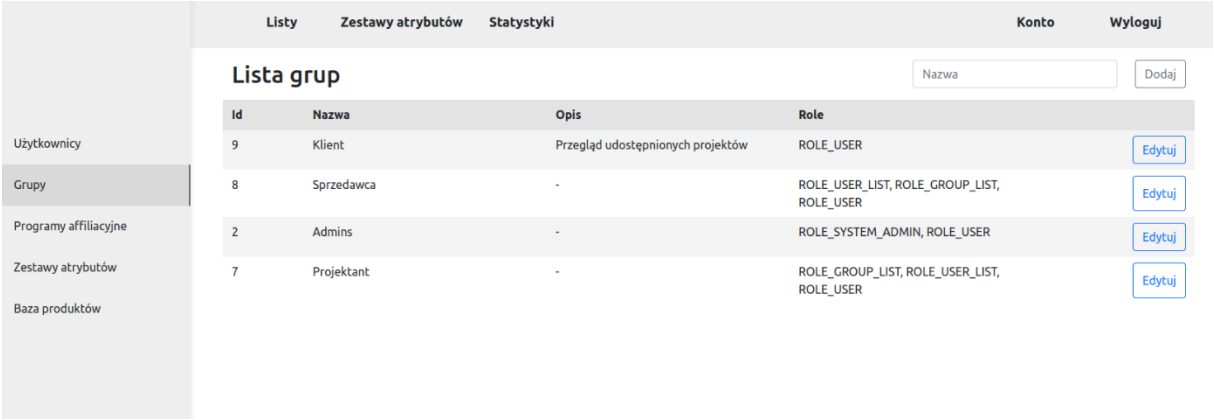
W tym rozdziale przedstawiono rezultaty opracowanego projektu oraz sposób obsługi aplikacji, zarówno od strony części serwerowej jak i wtyczki na przeglądarkę Chrome, a także aplikację na system Android. Opisano tutaj przykłady realizowania procesów biznesowych z wykorzystaniem prototypu.

## 7.1. Moduł logowania i autoryzacji

Dostęp do systemu odbywa się za pomocą zautoryzowanych kont. W celu zalogowania się system prezentuje poniższy modal logowania (rys. nr 29). Akcją alternatywną jest założenie nowego konta w systemie.



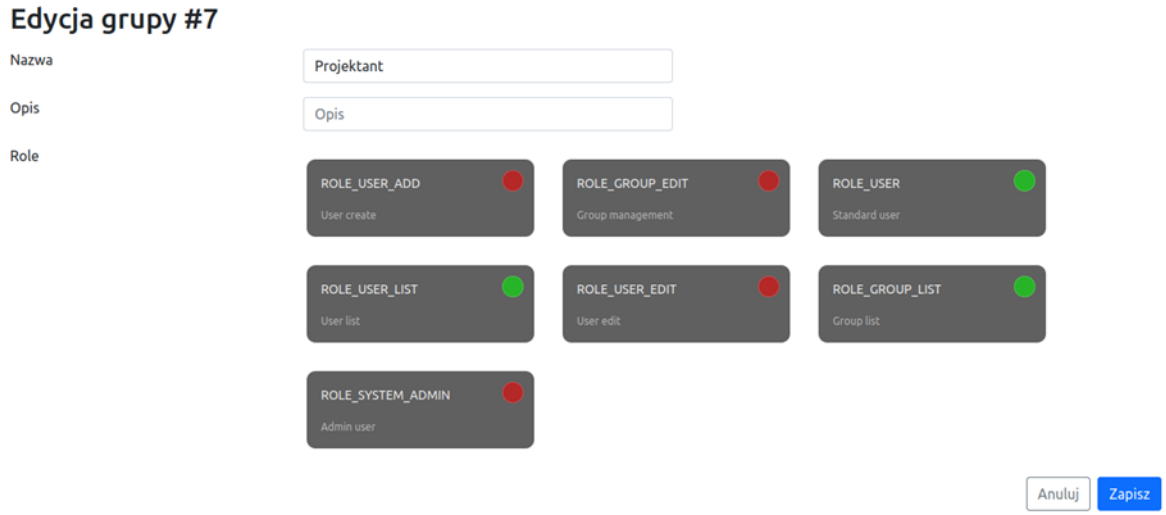

Rysunek 29 Formularz logowania do aplikacji. Źródło: opracowanie własne




Rysunek 30 Zrzut ekranu przedstawiający listę grup. Źródło: opracowanie własne

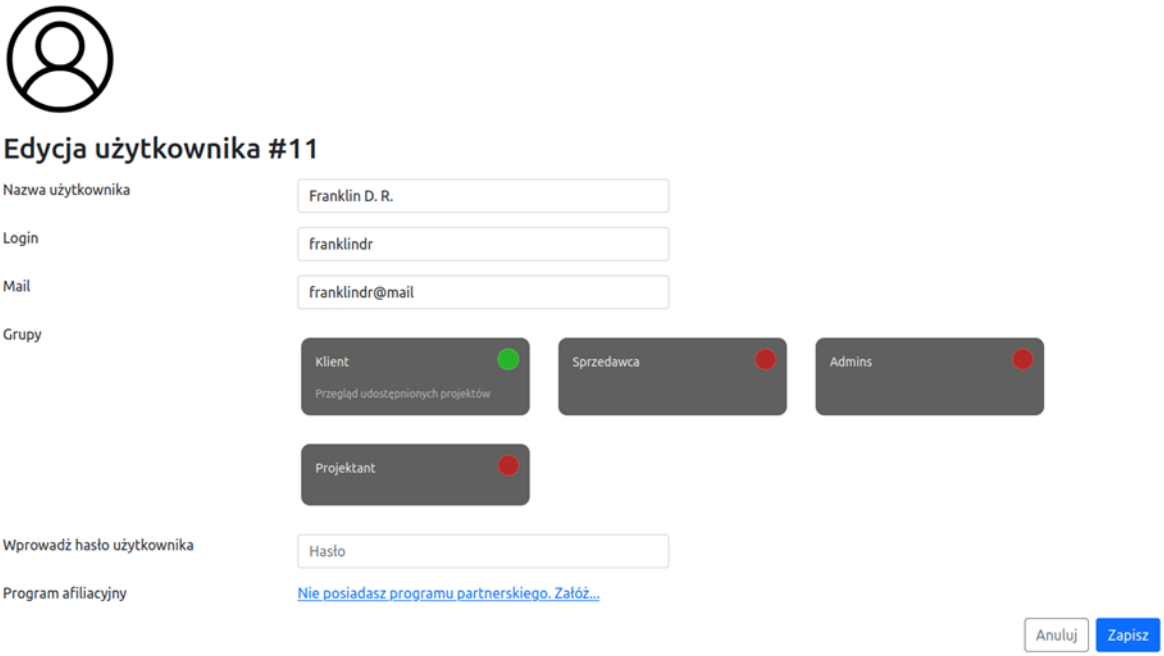
W zakres funkcji administratora systemu wchodzi możliwość definiowania grup, a także zarządzania nimi poprzez listę grup (rys nr 30). W celu dodania nowej grupy należy określić nazwę w polu „Nazwa”, a następnie kliknąć „Dodaj”. Jeśli proces przebiegł poprawnie na ekranie pojawi nowy

element listy z nowododaną grupą. By móc je edytować należy wybrać „Edytuj” dzięki czemu uzyskujemy dostęp do poniższego ekranu (rys. nr 31)




Rysunek 31 Zrzut ekranu przedstawiający formularz edycji grupy. Źródło: opracowanie własne

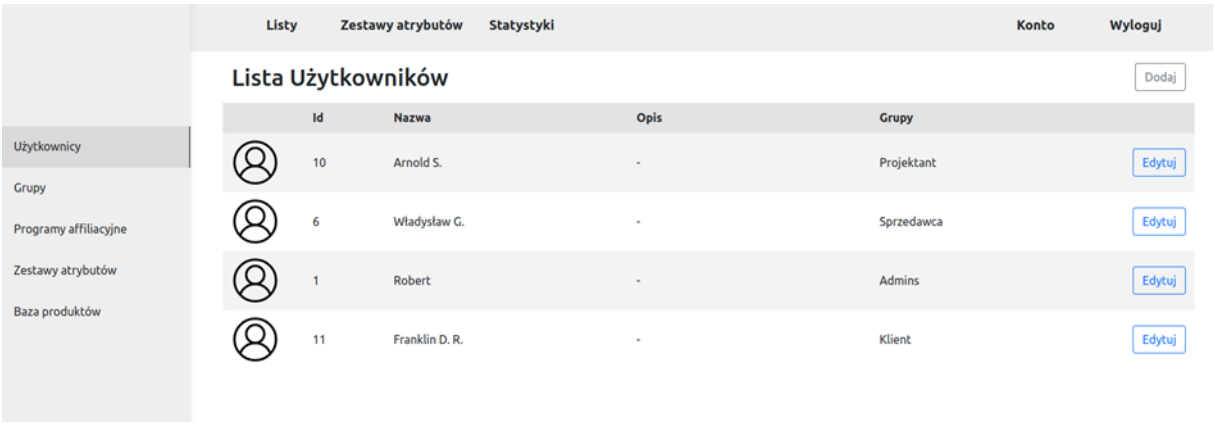
Formularza edycji użytkownika dostarcza administratorowi lub właścicielowi konta edycję podstawowych danych (obrazek nr 32).




Rysunek 32 Zrzut ekranu przedstawiający formularz edycji konta użytkownika. Źródło: opracowanie własne

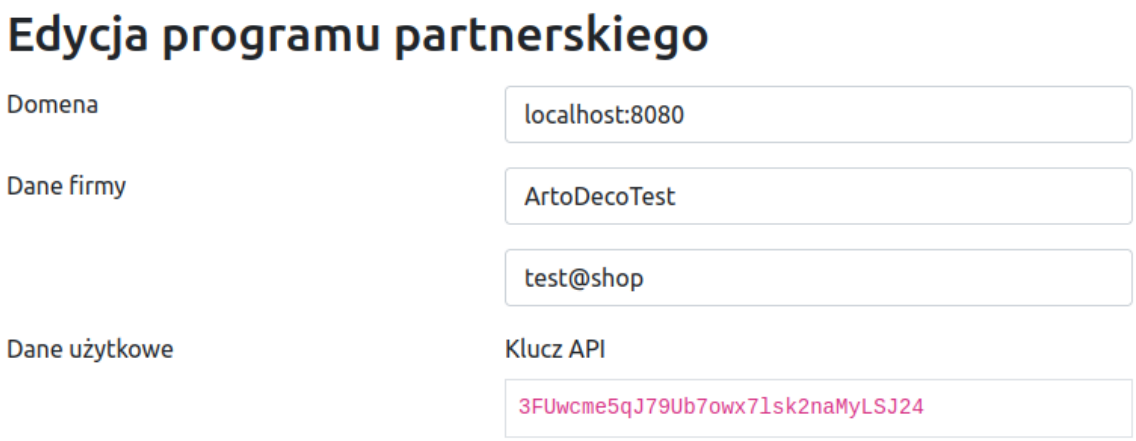
W przypadku gdy osobą modyfikującą dane jest administrator techniczny, może zmienić przypisane grupy, które wynikają z roli, którą objął użytkownik w czasie rejestracji. Na tym ekranie

aktor biznesowy sprzedawca ma prawo również wprowadzić dane programu partnerskiego (afiliacyjnego). Edycja użytkowników dostępna jest za pomocą panelu użytkownicy (rys nr 33)




Rysunek 33 Zrzut ekranu przedstawiający administratorską listę użytkowników w systemie

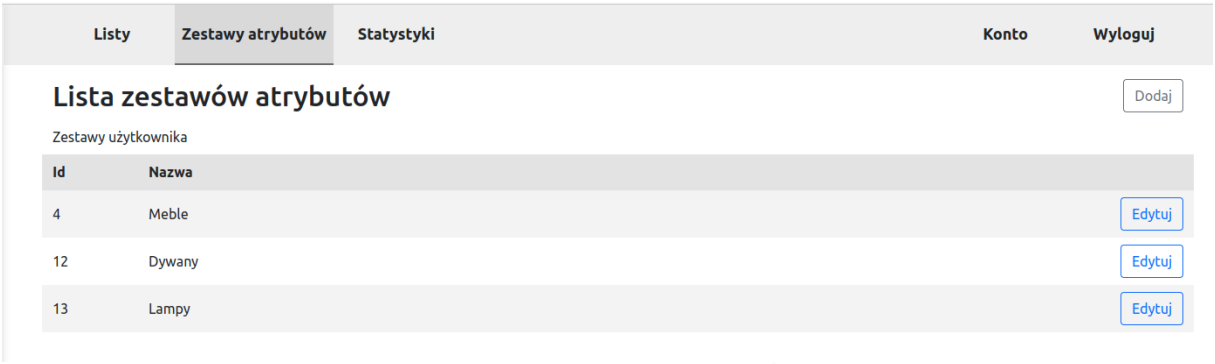
W ramach programu partnerskiego (edytor przedstawiony na rys nr 34) można definiować domenę sklepu, nazwę firmy, główny mail, a także pobierać wygenerowany klucz API.




Rysunek 34 Zrzut ekranu przedstawiający formularz edycji programu partnerskiego. Źródło: własne

## 7.2. Moduł atrybutów

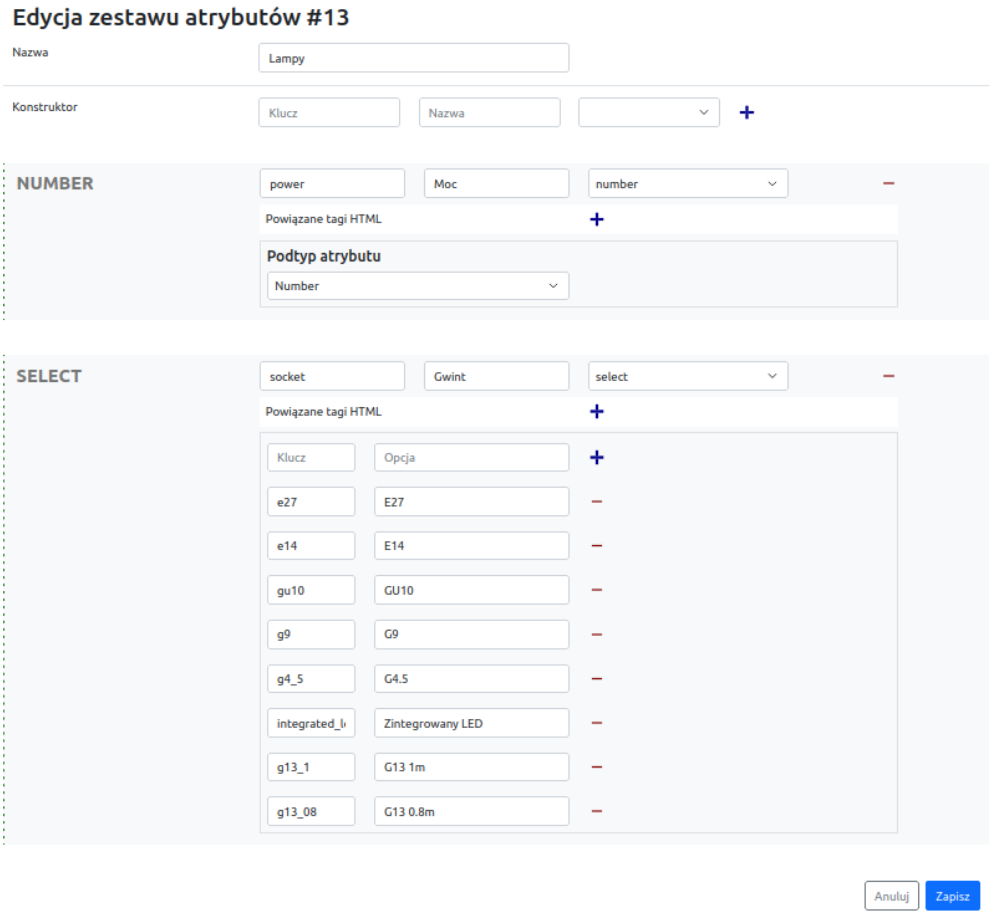
Zarządzanie zestawami atrybutów połączonymi z projektami odbywa się za pomocą listy przedstawionej poniżej (rys. nr 35)



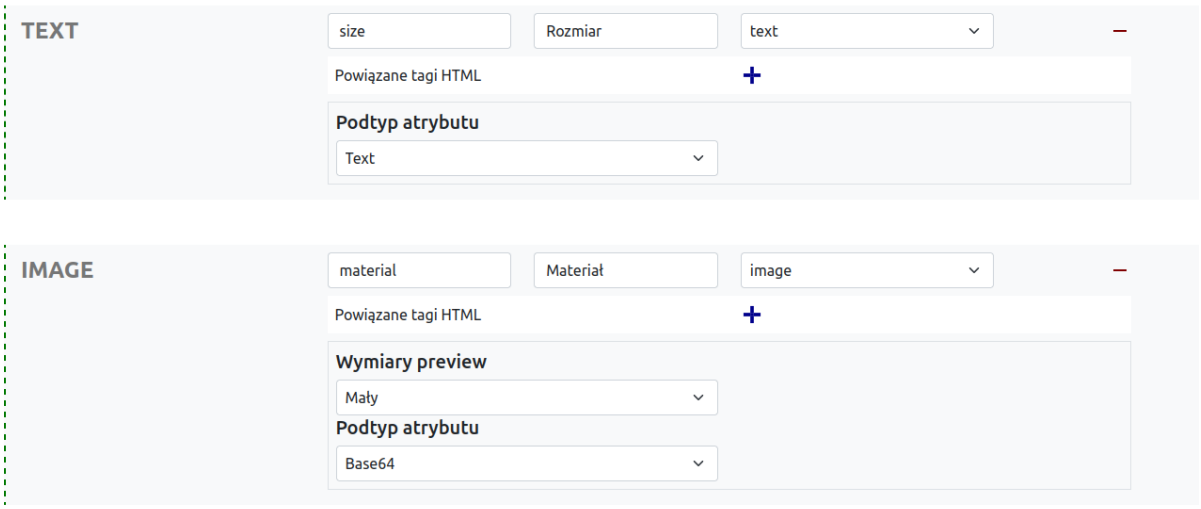

Rysunek 35 Zrzut ekranu listy zestawów atrybutów. Źródło: opracowanie własne

W celu zdefiniowania nowego zestawu należy użyć przycisku „Dodaj". Każdy zbiór atrybutów jest dostępny także do późniejszej edycji.

Atrybuty posiadają kilka predefiniowanych rodzajów. Są nimi typ liczbowy (posiada podtyp liczba całkowita, waluta), typ listy wyboru (oba rodzaje przedstawione na rys. nr 36), a także typ tekstowy (z podtypami prosta wartość oraz mail), czy też pole typu obrazek (rodzaj link oraz zapis w bazie jako base64) (rys. nr 37)



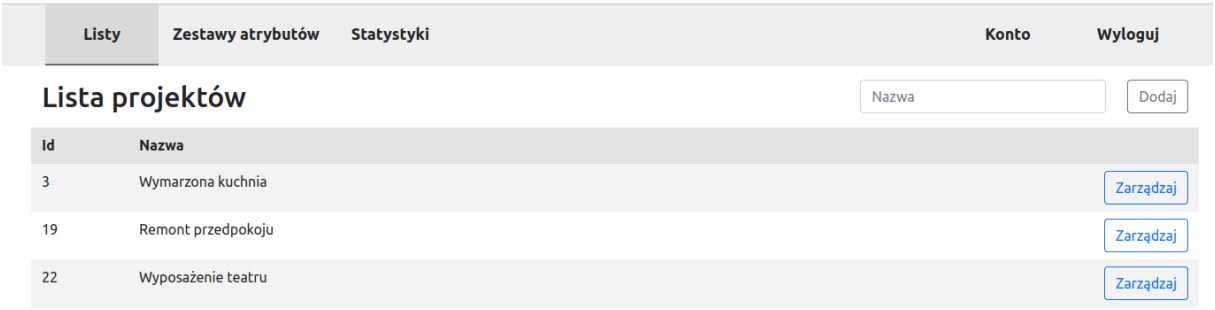

Rysunek 36 Zrzut ekranu edytora zestawu atrybutów. Źródło: opracowanie własne




Rysunek 37 Fragment ekranu przedstawiający 2 rodzaje konstruktora atrybutów nieujęte powyżej. Źródło: opracowanie własne

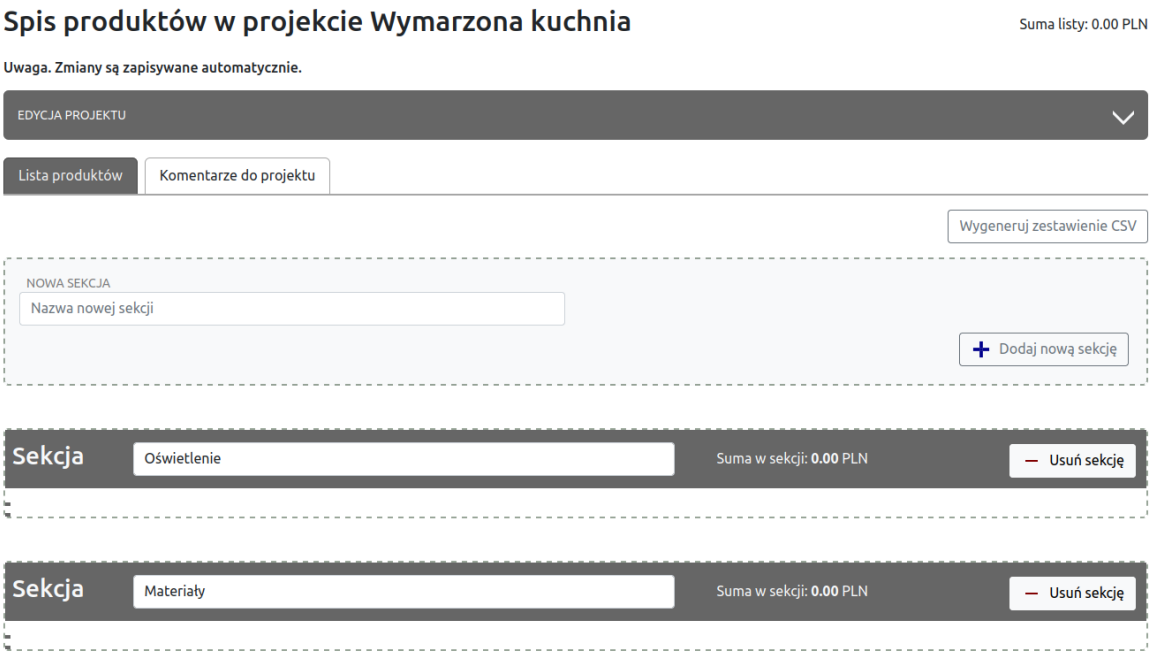
## 7.3. Moduł projektów

By móc tworzyć listy produktów a także uwag i wizualizacji należy wykorzystać przycisk „Listy". Ekran przedstawiający opisaną wcześniej zawartość został przedstawiony na rys. nr 38.




Rysunek 38 Zrzut ekranu przedstawiający listę projektów. Źródło: własne

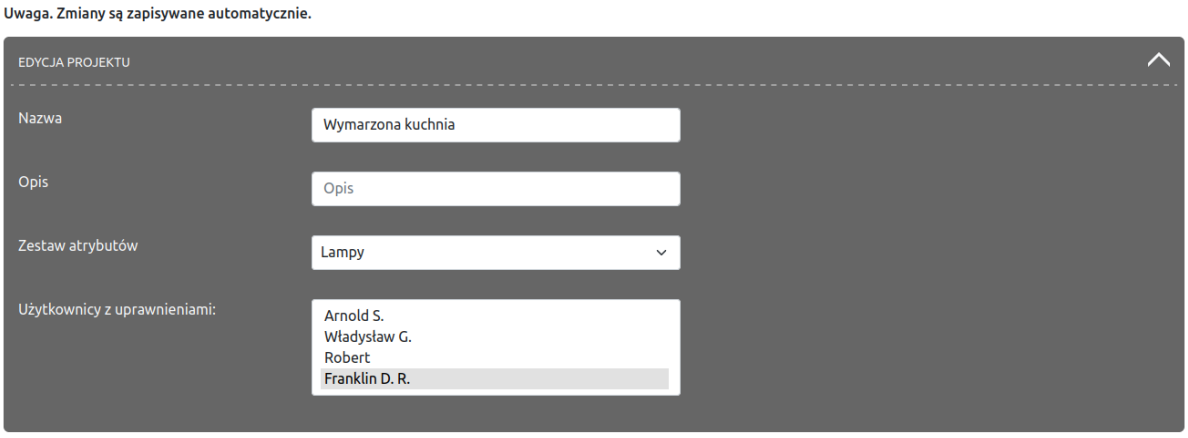
Gdy wybrany projekt zostanie oznaczony do edycji, system przedstawi zawartość jak na zdjęciu poniżej (rys. nr 39).




Rysunek 39 Pusty widok projektu, bez produktów. Źródło: własne

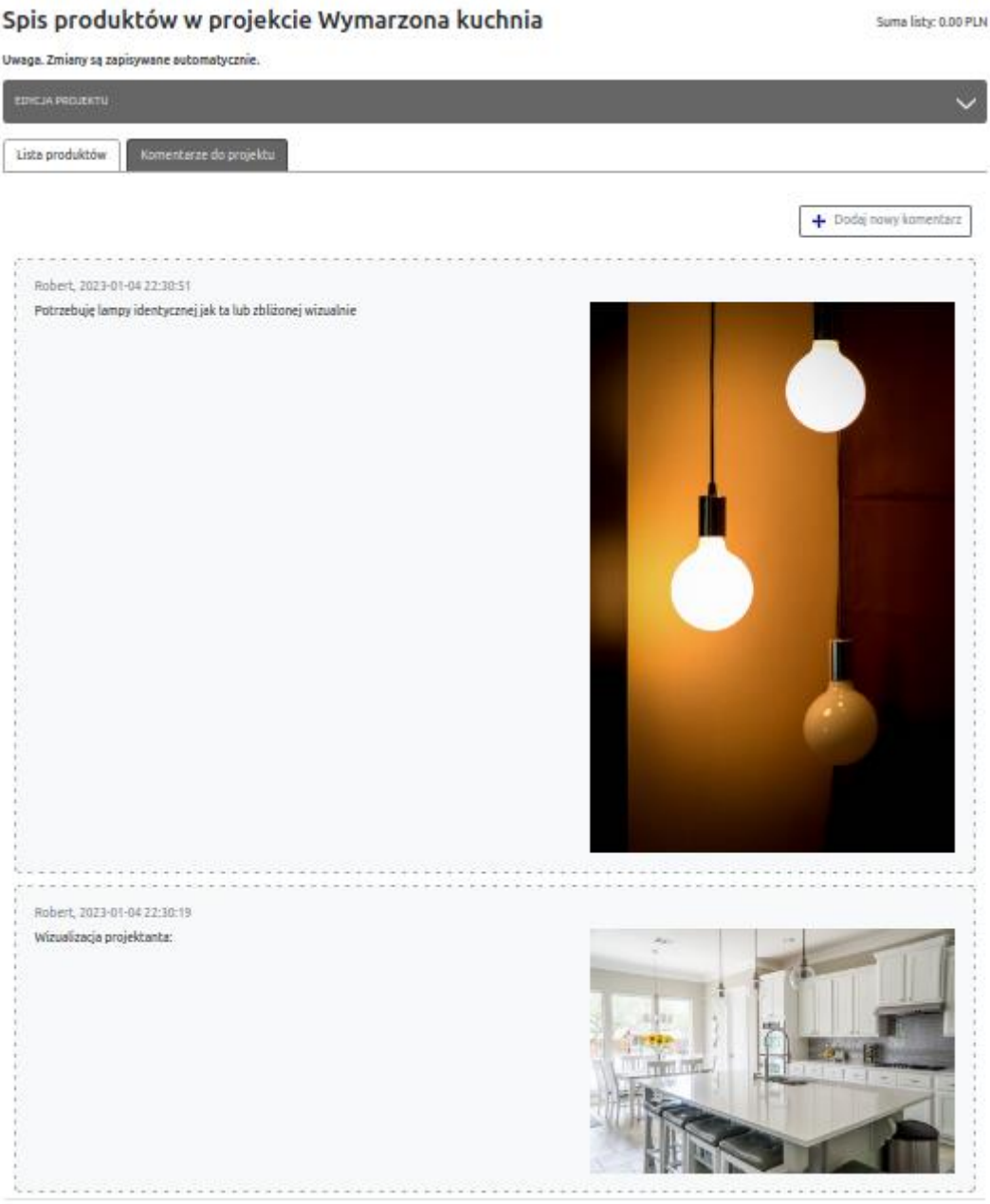
Projekt podzielony jest na 3 części:

- Ustawienia w bloku rozwijanym „Edycja projektu” (szerzej przedstawione na rys. nr 40);
- Zakładka „Lista produktów” – każdy zbiór produktów w projekcie może być organizowany w sekcje, które grupują produkty o podobnych cechach (mogą służyć jako kategorie produktów);
- Zakładka „Komentarze do projektu” – które zawierają uwagi klientów, a także wizualizacje pomocne w doborze produktów przez projektanta.



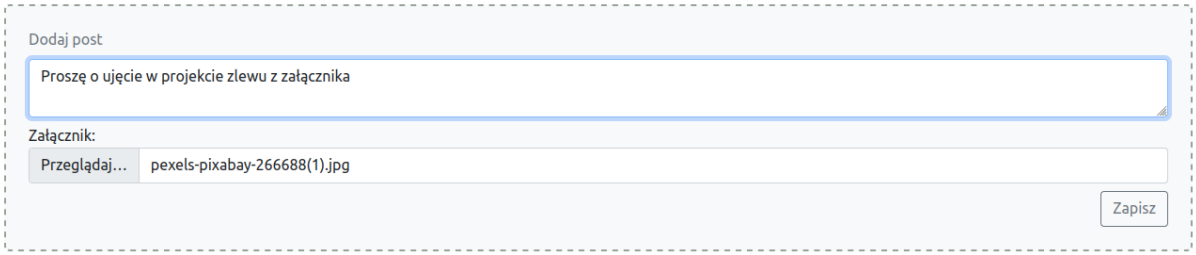

Rysunek 40 Sekcja edycji ustawień projektu. Źródło: własne

W ramach ustawień, projektant może zarządzać podstawowymi atrybutami, a także wybrać innych kreatorów lub klientów, którzy mają wgląd w zestawienie. Lista dodanych komentarzy została przedstawiona na rys. nr 41. Możliwość dodania nowego posta wraz z wizualizacją została zaprojektowana jako sekcja odkrywana po kliknięciu przycisku „dodaj nowy komentarz”.




Rysunek 41 Podgląd uwag do projektu. Źródło: opracowanie własne

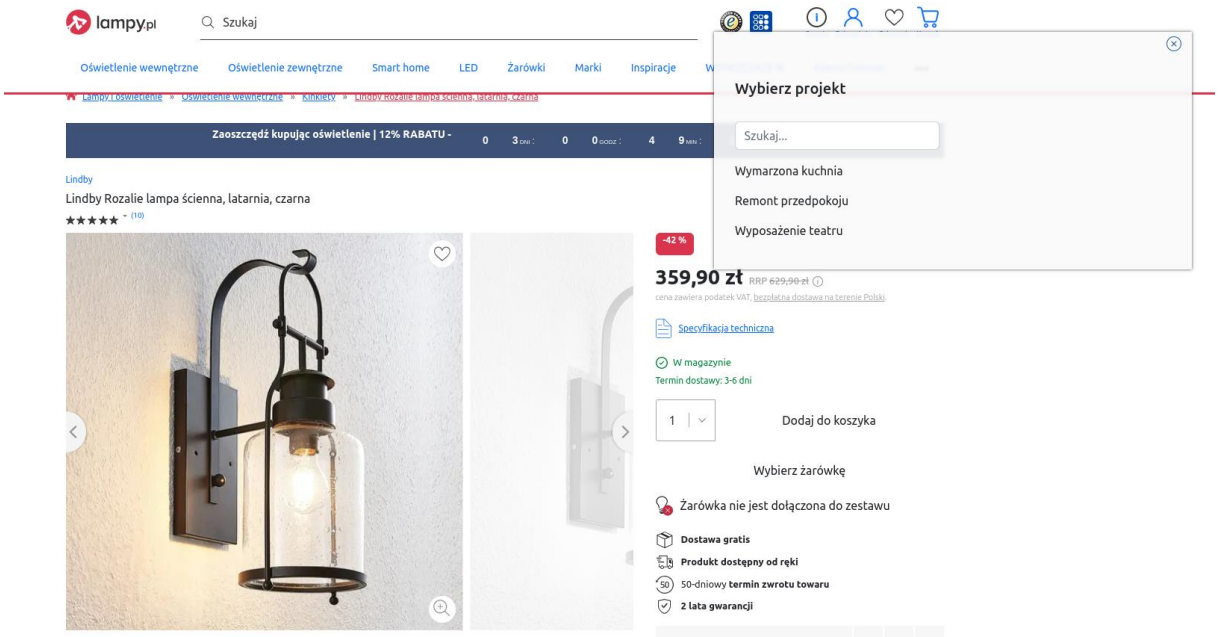
Formularz zawiera takie pola jak treść oraz załącznik w postaci pliku obrazkowego. (Rys. nr 42)




Rysunek 42 Formularz dodawania projektu/wizualizacji. Źródło: własne

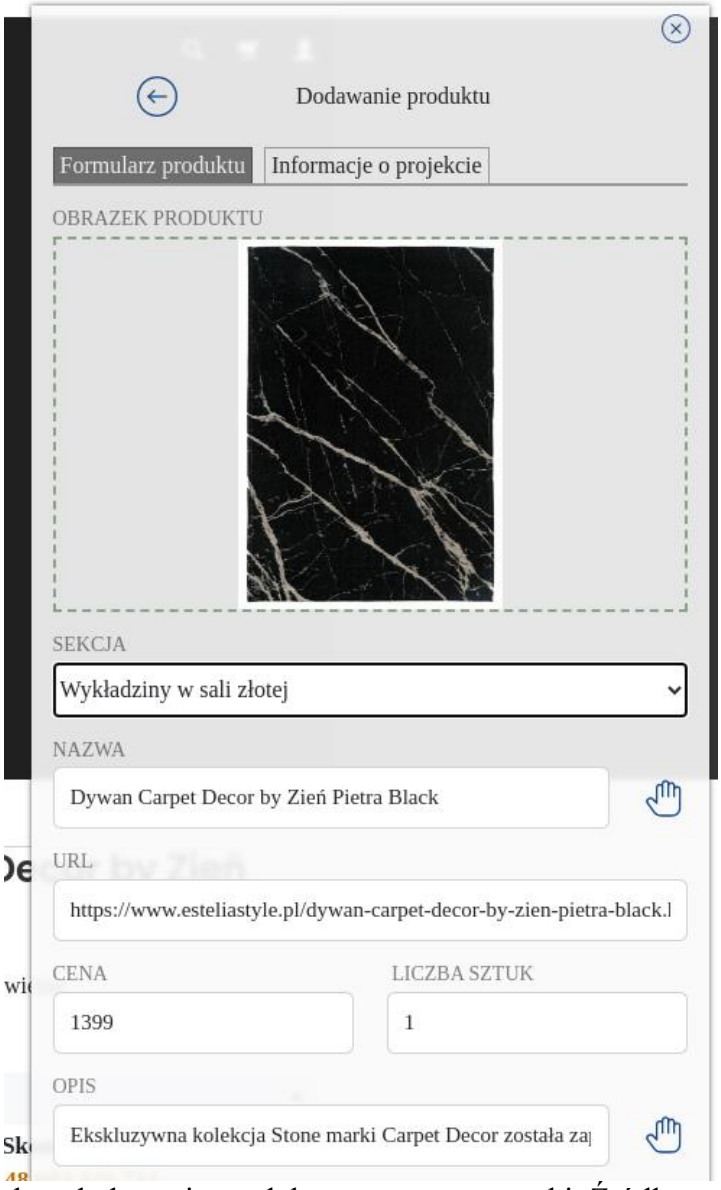
## 7.4. Dodawanie nowych produktów do projektu

Pierwszym krokiem, który musi zostać wykonany przez użytkownika jest instalacja wtyczki, a także poprawna autoryzacja w celu dostępu do serwera. Po wykonaniu tych czynności należy znaleźć interesujący produkt, a następnie uruchomić rozszerzenie na jego stronie. Po uruchomieniu zostaje przedstawiony modal wyboru projektu, do którego produkt będzie dodawany (przykład pokazany na rys. nr 43),




Rysunek 43 Moment po uruchomieniu wtyczki. Źródło: własne

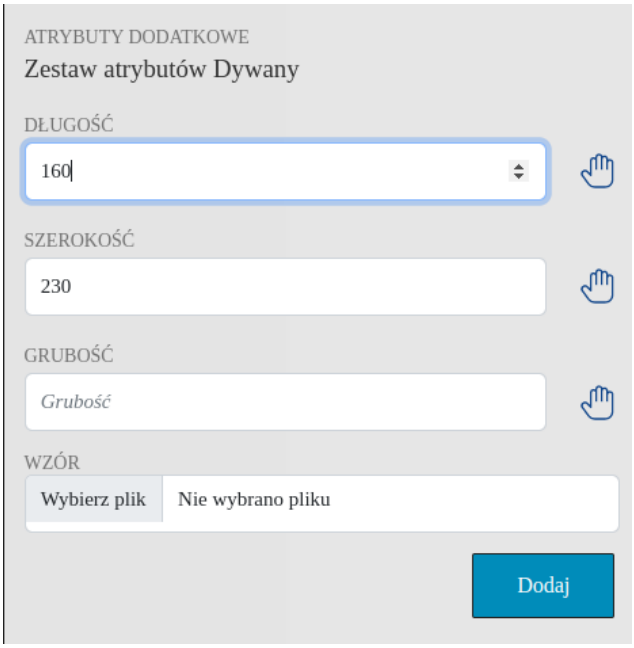
Po wybraniu projektu, następuje próba odczytania struktury schema.org zawartej na stronie, a także przetworzenia znaczników odpowiedzialnych za dodatkowe atrybuty. Jeśli dane zostaną poprawnie znalezione następuje aktualizacja tych pól, reszta pozostaje do uzupełnienia przez użytkownika (np. poprzez zaznaczenie tekstu i wybrania ikonki „łapki"). Jeżeli podobny produkt został już dodany do systemu, istnieje możliwość autouzupełnienia danych, z wykorzystaniem ikonki „odśwież", która pobiera dane tego produktu jednocześnie pozwalając na aktualizację pól, które się zmieniły od tego czasu. Przykład formularza wtyczki z uzupełnionymi danymi został przedstawiony poniżej (rys. nr 44):



Rysunek 44 Formularz dodawania produktu wewnątrz wtyczki. Źródło: opracowanie własne

Zrzut wycinka ekranu przedstawiony powyżej dotyczy pól podstawowych i uniwersalnych dla wszystkich produktów – są to:

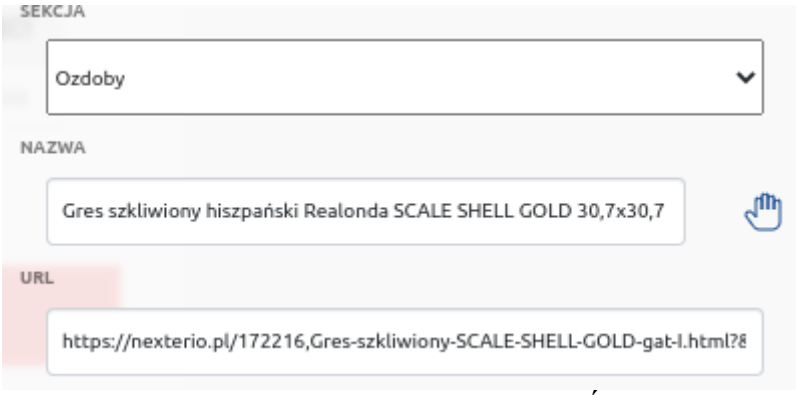
- Sekcja – wymagana;
- Nazwa – wymagana;
- Adres url – wymagany;
- Cena;
- Liczba sztuk – domyślnie „1”;
- Opis.



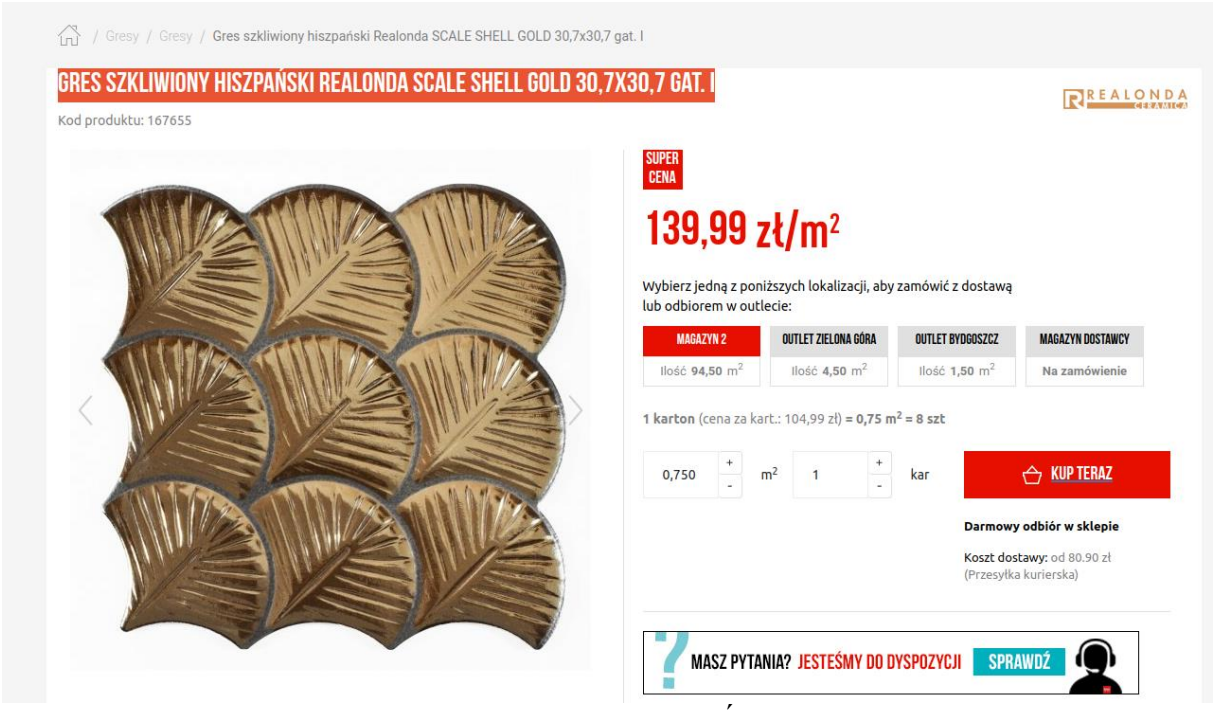

Rysunek 45 Sekcja atrybutów dodatkowych w formularzu. Źródło: własne

Sekcja „Atrybuty dodatkowe” zawiera formularz dla pól definiowanych przez projektanta. W przykładzie powyżej (rys. nr 45) zaprezentowano parametry typowe dla dywanów i ozdób tapicerowanych.

W celu uzupełnienia wartości pól, które nie mogą być odczytane poprzez webscraping, wystarczy zaznaczyć interesujący nas tekst ze strony (rys. nr 46), a następnie użyć przycisku z ikoną „łapki” (rys. nr 47).




Rysunek 46 Uzupełnianie wartości poprzez zaznaczenie. Źródło: opracowanie własne

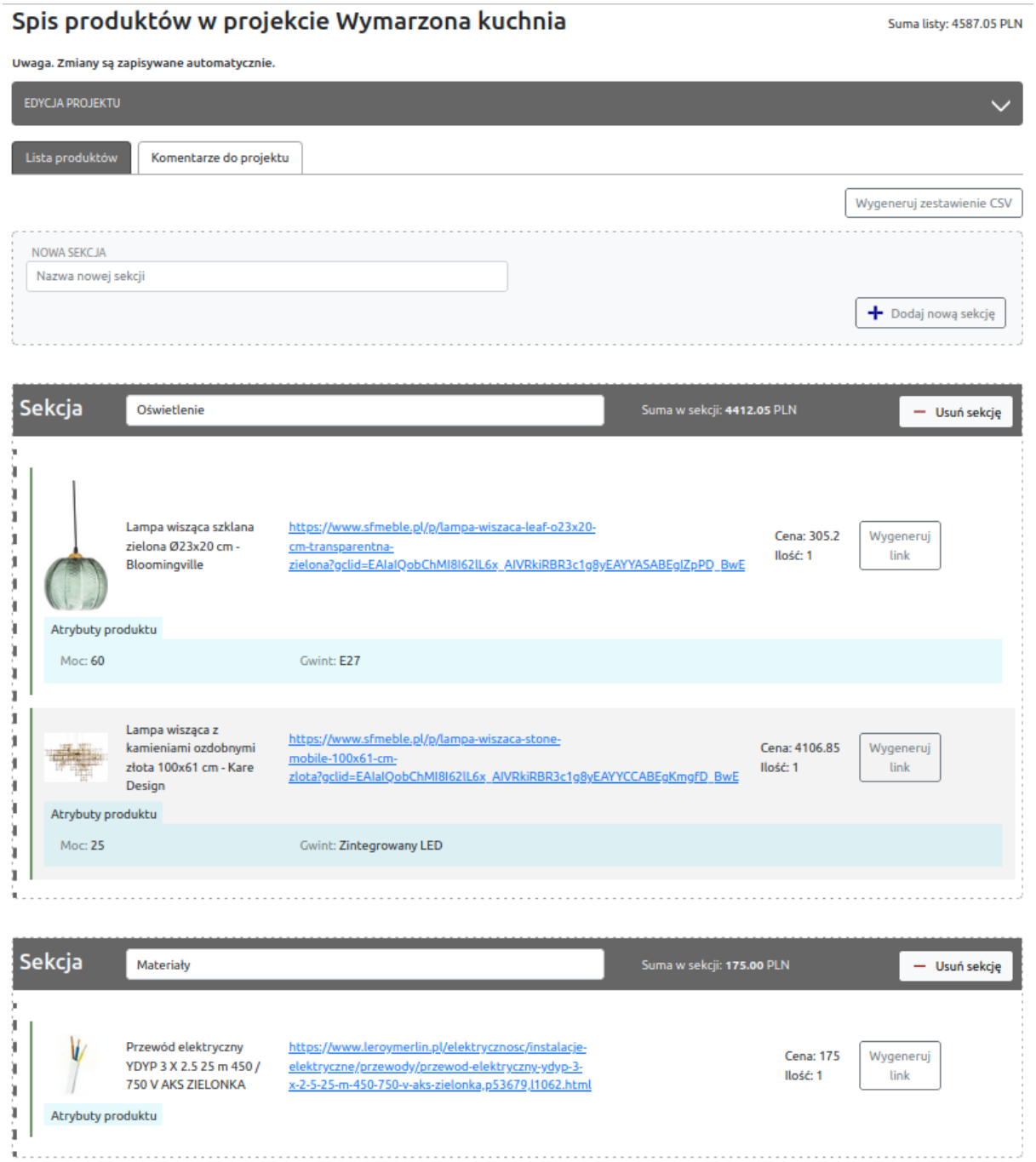



Rysunek 47 Strona przykładowego produktu. Źródło: opracowanie własne

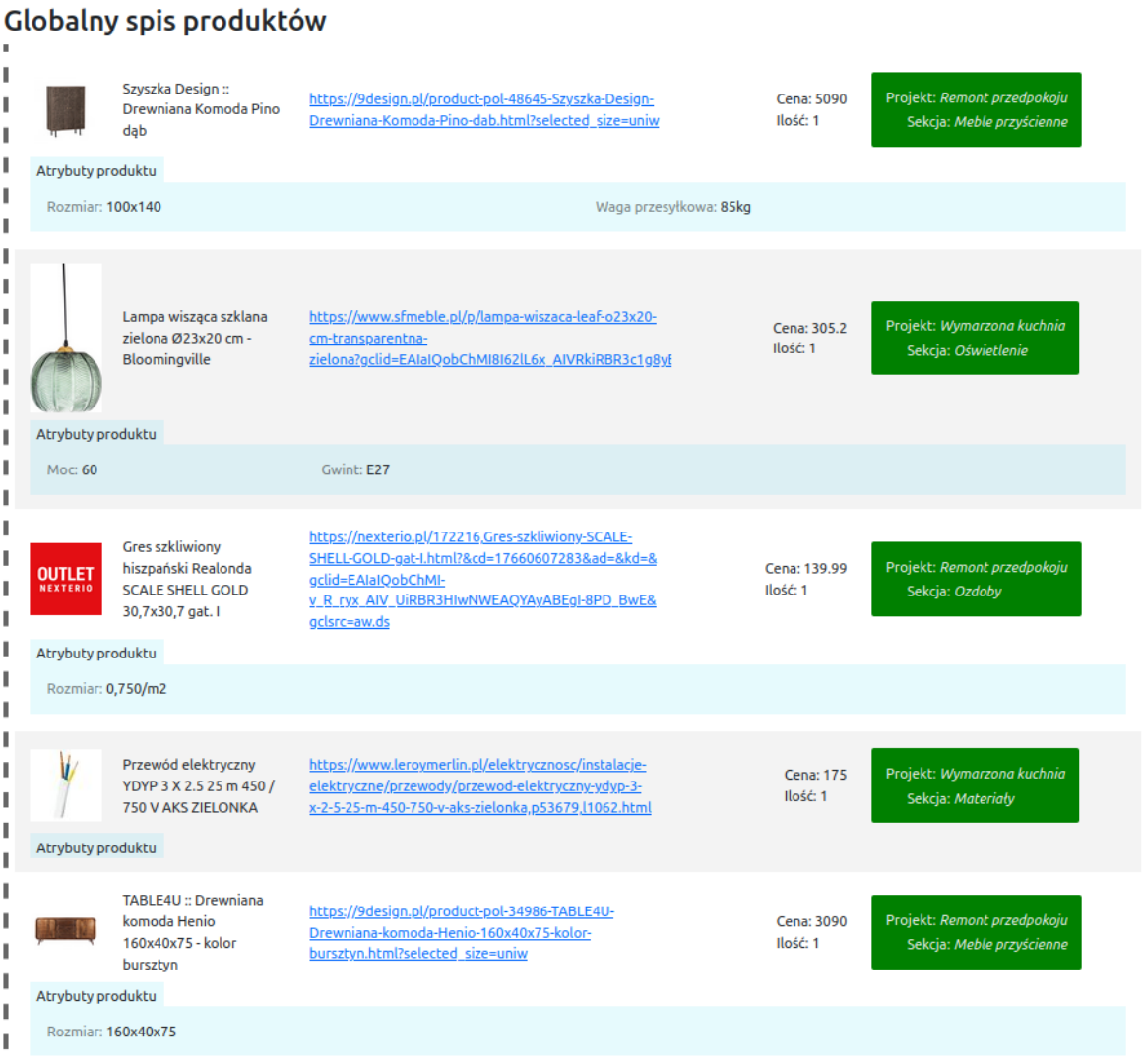
W celu szybkiego podglądu uwag oraz wizualizacji, dodano nową zakładkę we wtyczce, która pobiera odpowiednie dane według wybranego projektu. (rys. nr 48). Po wprowadzeniu kolekcji produktów z wykorzystaniem wtyczki, użytkownik może powrócić na edytor projektu (rys. nr 49). Informacje są przedstawione w sposób listy wierszy, które oprócz danych zawierają także przyciski do generowania linków afiliacyjnych. Użytkownik posiada możliwość pobrania zestawienia csv („Wygeneruj zestawienie csv") wraz z przygotowanymi linkami afiliacyjnymi. W każdej sekcji widnieje suma cen produktów (z ujęciem ilości sztuk). Projektant w oparciu o dodatkowe atrybuty może przygotować statystyki i zestawienia cenne dla klientów.

Osoby będące administratorami mają także możliwość przeglądu globalnej listy produktów (przykład na rys. nr 50), która dostarcza informacje na temat wykorzystania danych przedmiotów w projektach, a także sekcjach.


Rysunek 48 Sekcja uwag do projektu we wtyczce. Źródło: opracowanie własne



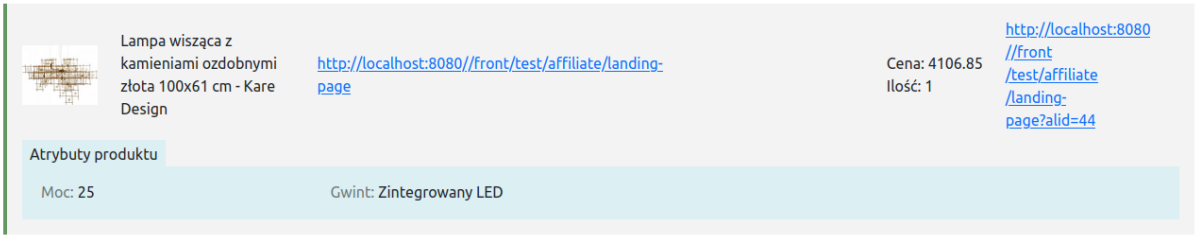

Rysunek 49 Widok projektu wraz z nowododanymi produktami. Źródło: opracowanie własne




Rysunek 50 Zrzut ekranu przedstawiający globalny spis produktów. Źródło: opracowanie własne

## 7.5. Moduł afiliacji

Warunkiem powodzenia procesu jest założenie programu afiliacyjnego przez sprzedawcę dla wskazanej domeny. Oznacza to wyrażenie chęci do wypłacania prowizji za wybór produktów, a co za tym idzie - zwiększanie wskaźników sprzedażowych. Projektant może zlecić wygenerowanie linku, a następnie skopiować zawartość i przekazać klientowi. Przykładowy link tego typu przedstawiony został na rys. nr 51.




Rysunek 51 Przykład wygenerowanego linku afiliacyjnego do strony z produktem. Źródło: opracowanie własne

W celu testów funkcjonalności oraz symulacji procesu w środowisku lokalnym, przygotowano 2 strony symulujące *landing-page* oraz *purchase-pase* zewnętrznego serwisu, który korzysta z dostarczonych kodów integracyjnych. Przykładowy kod *landing-page*, służący do testu obranego rozwiązania, został przedstawiony w listingu nr 33, a *purchase-page* – listing nr 34, co dało efekt w postaci widoków przedstawionych na 52 oraz 53. Jedyną czynnością, wymaganą do realizacji przez sprzedawcę, jest ustawienie klucza API, przed sekcją <*script*>, odnoszącą się udostępnionego kodu JS. W przypadku *purchase-page* – najważniejszym elementem jest przekazania eventu do warstwy *frontend* za pomocą `window.eventLayer`, który zostaje następnie przetworzony przez skrypt i wysłany do serwera.

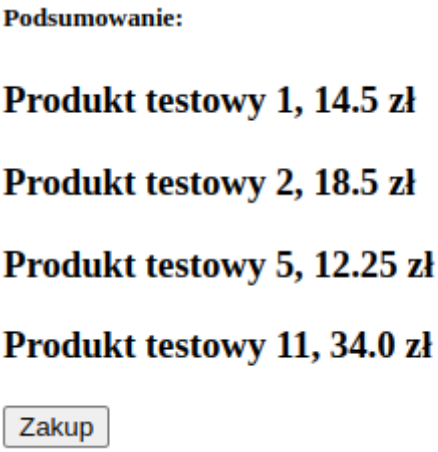
Listing 33 Fragment przykładu kodu strony *landing-page*. Źródło: opracowanie własne

```
<html lang="pl">
    <head>
        <meta charset="utf-8">

        <meta property="og:title" content="Produkt testowy 1">
        <meta property="og:product:price" content="14.5">
    </head>

    <body>
        ...

        <script>
            window.apiKey = "..."
        </script>
        <script
th:src="@{/static/js/affiliate_script.entry.js}"></script>
    </body>
</html>
```



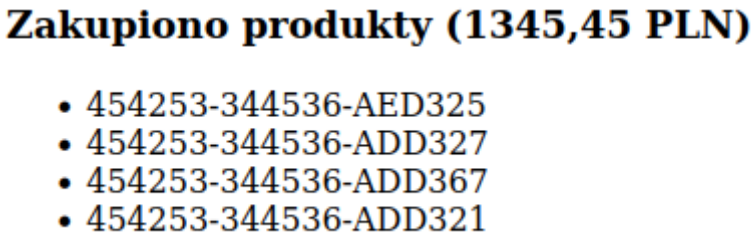

Rysunek 52 Widok fragmentu przykładowej strony landing-page. Źródło: opracowanie własne

Listing 34 Fragment przykładu kodu strony purchase-page. Źródło: opracowanie własne

```
<html lang="pl">
<head>
    <meta charset="utf-8">
</head>

<body>
    <script type="application/javascript">
        window.eventLayer = [{
            type: "purchase",
            commission: 123.45,
            productIds:     ["454253-344536-AED325",     "454253-344536-
ADD327", "454253-344536-ADD367", "454253-344536-ADD321"]
        }]
    </script>

    <script>
        window.apiKey = "..."
    </script>
    <script th:src="@{/static/js/affiliate_script.entry.js}"></script>
</body>
</html>
```

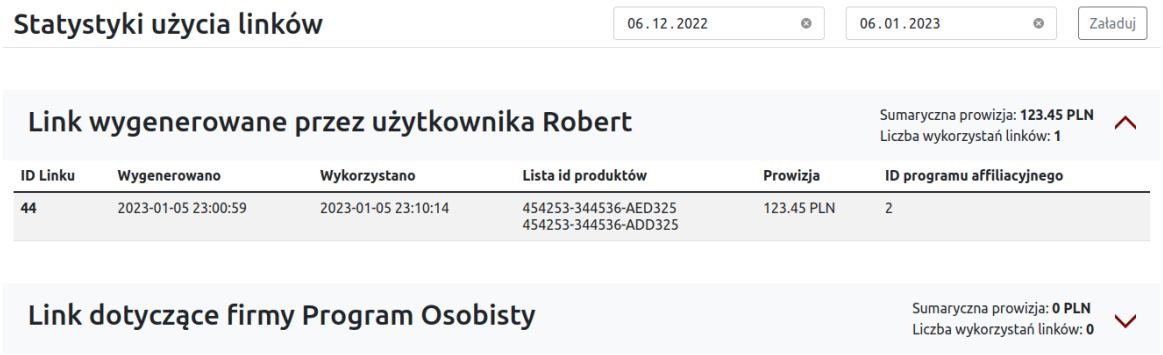



Rysunek 53 Wycinek przykładowej strony *purchase-page* z podsumowaniem. Źródło: opracowanie własne

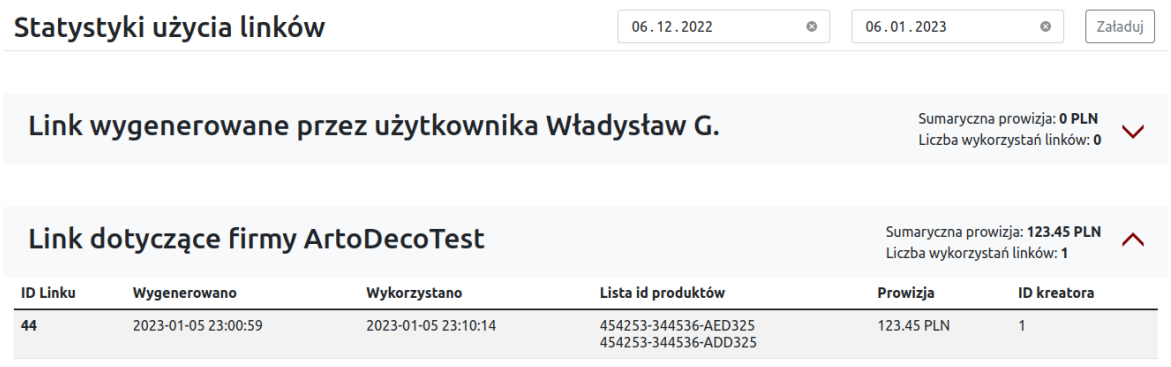
Rezultatem przejścia całego procesu przez obu użytkowników, jest rejestracja transakcji wraz z udzieloną prowizją. W celu zapewnienia kontroli nad procesem (przez wszystkich aktorów), dodano dodatkowy ekran „statystyki użytych linków". Podzielono je na 2 rodzaje:

- Linki wygenerowane przez projektanta (rys. nr 54);
- Linki skojarzone z danym programem afiliacyjnym (rys. nr 55).

Statystyki te pozwalają na przeprowadzenie rozliczeń, czy też walidacji, czy wszystkie transakcje oraz ich prowizje zostały poprawnie zarejestrowane.



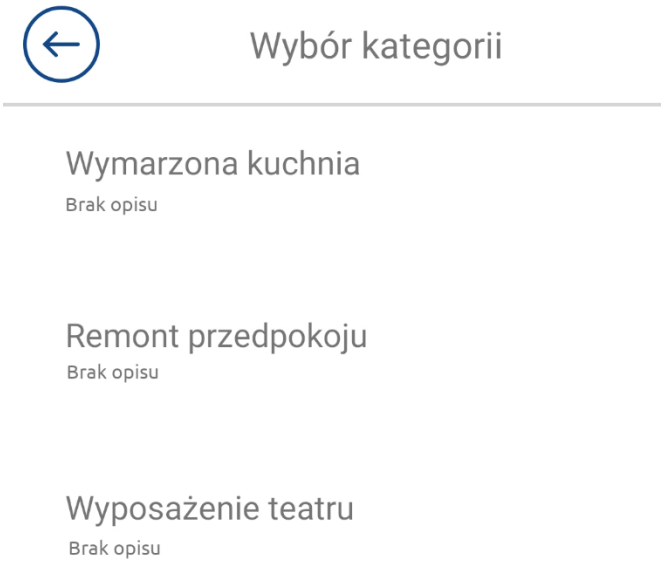

Rysunek 54 Statystyki użycia linków w ujęciu per projektant




Rysunek 55 Statystyki użycia linków w ujęciu per sprzedawca

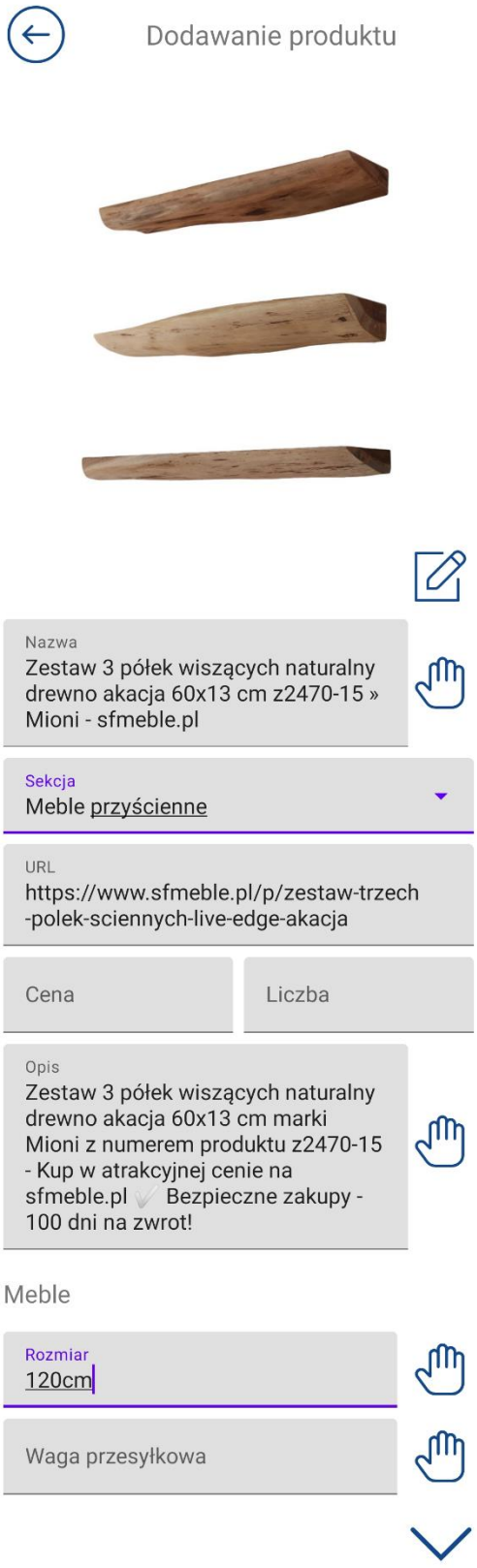
## 7.6. Aplikacja mobilna

W celu korzystania z tej aplikacji należy wprowadzić poświadczenia do formularzu logowania. Projektant, gdy znajdzie interesujący go wybór, będąc na jego stronie, wybiera systemowy „udostępnij”, następnie wskazuje na tą aplikację. Analogicznie jak we wtyczce – użytkownik musi wybrać projekt (zrzut ekranu z rys. nr 56). Po wybraniu projektu następuje próba *webscrapingu* i pobrania danych ze strony. Formularz wprowadzania nowego produktu został przedstawiony na rys. nr. 57.




Rysunek 56 Wybór projektu w aplikacji mobilnej. Źródło: opracowanie własne

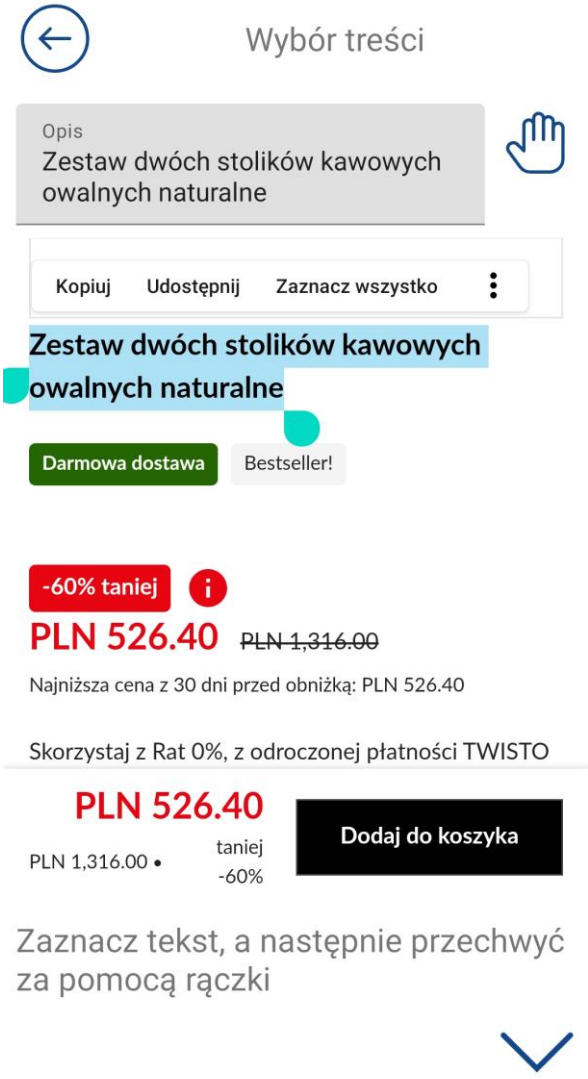
Każde pole tekstowe zawiera obok ikonkę w postaci łapki, która po kliknięciu uruchamia komponent WebView, służący do zaznaczania tekstu i uzupełniania nim wartości danego pola.




Rysunek 57 Formularz oddawania produktu z uzupełnionymi danymi. Źródło: opracowanie własne

Przykład wykorzystania zaznaczenia tekstu jest zwizualizowany za pomocą zrzutów ekranu poniżej (rys. nr 58, rys. nr 59):


Rysunek 58 Zawartość pola opis przed aktualizacją. Źródło: własne




Rysunek 59 Zawartość pola opis po aktualizacji. Źródło: własne

Po zakończenie pracy z formularzem produkt zostaje dodany, z wykorzystaniem przycisku z ikoną „checked”.

# 8. Analiza rezultatów

Ten rozdział skupia się na analizie stopnia wypełnienia wymagań przez prototyp, identyfikacji wad oraz zalet, wskazanie potencjalnych ścieżek rozwoju aplikacji i pomysłów na przyszłość.

## 8.1. Wady i zalety

Uzyskany prototyp nie jest wolny od niedoskonałości, ale zawiera cechy, które pozwalają wyróżnić rozwiązanie na tle konkurencji.

Do grona wad możemy zaliczyć:

- Brak wsparcia systemu iOS;
- Brak wsparcia przeglądarki Chrome;
- Moduł serwerowy rozwiązania został przygotowany tylko do pracy z urządzeniami desktop, brak wsparcia responsywności w niektórych przypadkach;
- Dane wypełniane w formularzu wtyczki nie są cachowane, co pozwoliłoby na pracę z systemem w sytuacji utraty połączenia z siecią, co występuje relatywnie często (błąd sieci ERR_NETWORK_CHANGED);
- Rozwiązanie prototypowe mogłoby współpracować z kontenerem Mercure, co pozwoliłoby na natychmiastowe odświeżanie zmian w projekcie jeśli inny projektant je wprowadzi;
- System nie jest zintegrowany z dostawcami tożsamości jak Google.

Zaletami rozwiązania są m.in.:

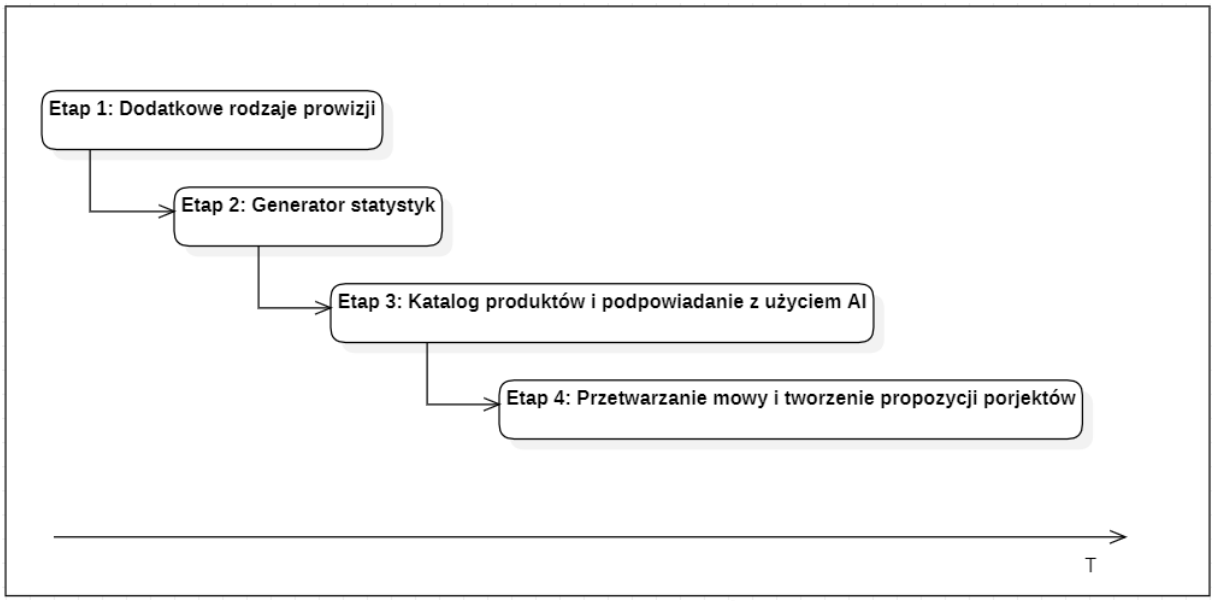
- Wprowadzenie zestawów dodatkowych atrybutów do systemów kolekcjonowania treści;
- Usprawnienia komponentów odpowiedzialnych za webscraping stron produktów;
- Wprowadzenie procesów afiliacyjnych do systemu;
- Prostota obsługi, która w zamierzeniu ma przyśpieszyć pracę z projektem;
- System zaprojektowany o możliwość łatwego dodawania nowych rodzai atrybutów;
- Podział aplikacji na serwisy co ułatwia pracę programistom przy potrzebie implementacji nowej funkcjonalności;
- Z punktu widzenia technologii zastosowano aktualne standardy programowania oraz biblioteki, które będą posiadać wsparcie przez długi czas;
- Aplikacja choć dedykowana projektantom wnętrz, po drobnych modyfikacjach może sprawdzać się w innych dziedzinach gdzie udział biorą usługodawca – pośrednik – usługobiorca jak np. sporządzanie projektu organizacji imprez masowych.

## 8.2. Kierunki rozwoju aplikacji

Po analizie bieżącej funkcjonalności, a także możliwych kierunków rozwoju zaproponowano następujące rozwiązania:

- Podczas implementacji rozwiązania, jedna z konkurencyjnych firm opracowała podpowiadanie proponowanych produktów, które można dodać do listy (Formly). Analiza funkcjonalności wykazała, że szukanie podobnych produktów odbywa się za pomocą prostych technik wyszukiwania podobnych fraz za pomocą kwerend SQL. Wychodząc naprzeciw temu, możliwe jest zaimplementowanie rozwiązania opartego o algorytmu SI z wykorzystaniem dodatkowych cech produktów jako model wielowymiarowy. Przykładem algorytmu, który mógłby to zrealizować może być drzewo klasyfikacji *Decision Tree* połączone z API biblioteki *Keras*, która implementuje rozwiązania nauczania maszynowego;
- Jednym z możliwych kierunków rozwoju jest także wykorzystanie technik NLP (*Natural Language Processing*), za pomocą których klient może dyktować wymagania co do projektów, a projektant otrzyma propozycje produktów najbardziej pasujących do opisu, ze zbioru tych co zostały dodane do bazy danych. Rozwiązanie to wymaga dużej ilości zapisanych danych;
- Rozszerzenie procesów afiliacyjnych o alternatywne metody rozliczania – np. mikroprowizja za ilość wyświetleń produktów na projektach użytkownika;
- Dodanie uniwersalnego generatora statystyk – funkcjonalność wprowadziłaby konstruktor statystyk z danych dostępnych w projektach, które użytkownicy mogą budować sami, za pośrednictwem komponentów wizualnych GUI, w celu uzyskania interesujących go danych np. koszt za produkty posiadające pewną kategorię z danym wzorem.

Podsumowujący harmonogram przedstawiony został na rys nr 60.




Rysunek 60 Harmonogram rozwoju aplikacji. Źródło: opracowanie własne

## 8.3. Podsumowanie

W ramach projektu opracowano i przeanalizowano problematykę tematu systemów wspomagających pracę architektów wnętrz. Zbadano także rozwiązania dostępne na rynku, określono ich zalety, wskazano wady i przedstawiono potencjalne sposoby na ich udoskonalenie. W celu realizacji założeń, na których opiera się prototyp, sporządzono zbiór wymaganej wiedzy teoretycznej, dotyczącej współpracy afiliacyjnej. Następnie przeprowadzono analizę potrzeb określonych przez potencjalnych użytkowników aplikacji. Na jej podstawie określono koncept architektury systemu, z wykorzystaniem diagramów UML. Wykonano także końcową ewaluację projektu i wyciągnięto wnioski co do możliwych kierunków rozwoju.

Bezpośrednim efektem tej pracy jest wytworzenie pełnej koncepcji rozwiązania problemu, którego rezultatem była implementacja prototypu z wykorzystaniem dobrych praktyk programowania, a także aktualnych technologii i standardów. Przedstawione tu rozwiązanie charakteryzuje się współpracą ze zintegrowaną, wewnętrzną siecią afiliacyjną, co jest wyróżnieniem względem konkurencyjnych oraz istniejących rozwiązań. Ponadto produkty, które są dodawane do systemu poprzez wtyczkę na przeglądarkę Chrome/aplikację Android, posiadają możliwość przypisywania zestawów atrybutów rozszerzających zakres gromadzonych danych. Zestawy atrybutów mogą być dowolnie modyfikowane przez projektanta. Użytkownik ma do dyspozycji intuicyjne oprogramowanie z prostym i przejrzystym interfejsem.

Architektura prototypu została opracowana z myślą o wprowadzaniu kolejnych funkcjonalności. Możliwe kierunki rozwoju przedstawiono w punkcie 8.2. Postawione przed koncepcją prototypu cele zostały w pełni zrealizowane.

# Bibliografia


[1]. Zasób internetowy cs.lmu.edu.pl, https://cs.lmu.edu/~ray/notes/webapps/, dostęp: 29 październik 2022

[2]. Strona aplikacji, https://kislist.com/#about, dostęp: 30 październik 2022

[3]. Strona aplikacji, https://listonic.com/pl/funkcje/, dostęp 1 listopada 2022

[4]. Strona aplikacji, https://formly.pl/pl/product, dostęp 1 listopada 2022

[5]. Strona aplikacji, https://webepartners.pl/reklamodawca/, dostęp 1 listopada 2022

[6]. Strona aplikacji, https://webepartners.pl/wydawca/, dostęp 1 listopada 2022

[7]. Zasób internetowy, https://webepartners.pl/wp-content/uploads/2022/03/personalizuj_widget_1.jpg, dostęp 30 październik 2022

[8]. Zasób internetowy, https://pracabezszefa.pl/wp-content/uploads/2018/12/zarobki-z-webepartners.png, dostęp: 1 listopada 2022

[9]. Zasób intenetowy, https://apilevels.com/, dostęp 28 listopada 2022

[10]. Strona bibliotek, https://jsoup.org/apidocs/, dostęp 13 grudnia 2022

[11]. Zasób internetowy, https://aff44.com/blog/historia-afiliacji-jak-to-sie-zaczelo, dostęp 4 listopada 2022

[12]. ISBN: 978-83-650-6899-6, John Kalench, MLM Marketing Wielopoziomowy (przekład polski Being The Best You Can Be in MLM), 2018

[13]. Zasób internetowy, https://support.google.com/google-ads/answer/116495?hl=pl&ref_topic=24937, dostęp 11 listopada 2022

[14]. Zasób internetowy, https://www.techtarget.com/whatis/definition/cost-per-sale-CPS, dostęp 11 listopda 2022

[15]. Zasób internetowy, https://www.bigcommerce.com/ecommerce-answers/what-are-cpl-cost-per-lead-campaigns/, dostęp 11 listopada 2022

[16]. Zasób internetowy, https://www.klipfolio.com/metrics/advertising/cost-per-thousand-cpm/, dostęp 11 listopada 2022

[17]. Zasób internetowy, https://sprawnymarketing.pl/blog/marketing-afiliacyjny-programy-partnerskie-podstawy/, dostęp 12 listopada 2022

[18]. Zasób internetowy, https://webepartners.pl/blog/afiliacja-co-to-jest/, dostęp 12 listopada 2022

[19]. Zasób internetowy, https://business.trustedshops.pl/blog/cost-per-click-definicja/, dostęp 13 listopada

[20]. Strona standardu Schema.org, https://schema.org/, dostęp 14 listopada

[21]. Dokumentacja standard Schema.org, https://schema.org/docs/full.html, dostęp 14 listopada

[22]. Narzędzie, https://search.google.com/test/rich-results, dostęp 14 listopada

[23]. Zasób MSDN, https://developer.mozilla.org/en-US/docs/Glossary/MVC, dostęp 15 listopada 2022

[24]. Zdjęcie, https://developer.mozilla.org/en-US/docs/Glossary/MVC/model-view-controller-light-blue.png, dostęp 15 listopada 2022

[25]. Zasób internetowy, https://www.rfc-editor.org/rfc/rfc7231#section-4, dostęp 16 listopada 2022

[26]. Dokumentacja kotlin, https://kotlinlang.org/docs/getting-started.html#create-your-powerful-application-with-kotlin, dostęp 16 listopada 2022

[27]. Dokumentacja Kotlin, https://kotlinlang.org/docs/basic-syntax.html#type-checks-and-automatic-casts, dostęp 16 listopada 2022

[28]. Dokumentacja Spring, https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html, dostęp 17 listopada 2022

[29]. Dokumentacja Spring, https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html, dostęp 17 listopada 2022

[30]. https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html, dostęp 17 listopada 2022

[31]. Dokumentacja Mozilla, https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics, dostęp 17 listopada 2022

[32]. Documentacja Mozilla, https://developer.mozilla.org/en-US/docs/Web/HTML/Element, dostęp 17 listopada 2022

[33]. Dokumentacja Mozilla, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference, dostęp 17 listopada 2022

[34]. Dokumentacja Vue, https://vuejs.org/guide/introduction.html#single-file-components, dostęp 18 listopada 2022

[35]. Dokumentacja Postgres, https://www.postgresql.org/about/, dostęp 18 listopada 2022

[36]. Strona JetBrains, https://www.jetbrains.com/idea/features/, dostęp 19 listopada 2022

[37]. Dokumentacja developers.android, https://developer.android.com/studio/intro, dostęp 20 listopada 2022

[38]. Dokumentacja developers.android, https://developer.android.com/topic/architecture, dostęp 20 listopada 2022

[39]. Dokumentacja developer.chrome, https://developer.chrome.com/docs/extensions/mv2/overview/, dostęp 20 listopada 2022

[40]. Zasób internetowy, https://www.cookiebot.com/en/cookie-law/, dostęp 15 stycznia 2023

# Spis rysunków

Rysunek 1 Zrzut ekranu zarządzania listą produktów w KIS List. Źródło: opracowanie własne. ..... 9
Rysunek 2 Interfejs użytkownika wtyczki do dodawania/edycji produktów. Źródło: opracowanie własne ..... 9
Rysunek 3 Ekran wyboru oraz definiowania listy zakupowej. Źródło: opracowanie własne.. 10
Rysunek 4 Ekran zarządzania elementami listy zakupowej. Źródło: opracowanie własne ..... 11
Rysunek 5 Ekran edycji danych produktu w aplikacji Formly. Źródło: opracowanie własne. 12
Rysunek 6 Fragment aplikacji Formly podsumowujący projekt. Źródło: opracowanie własne ..... 12
Rysunek 7 Zrzut okna przedstawiający listę ostatnio dodanych materiałów. Źródło: opracowanie własne ..... 13
Rysunek 8 Zrzut ekranu przedstawiający zakładkę "materiały". Źródło: opracowanie własne ..... 13
Rysunek 9 Obraz przedstawia fragment ekranu kreatora rotatorów. Źródło: opracowanie własne ..... 15
Rysunek 10 Zdjęcie przedstawia przykład kreatora *widgetów*. Żródło: [6] ..... 15
Rysunek 11 Zrzut ekranu przedstawiający podsumowanie transakcji. Źródło: [7] ..... 16
Rysunek 12 Diagram przedstawiający relacje pomiędzy poszczególnymi warstwami architektury MVC. Żródło: [24] ..... 27
Rysunek 13 Diagram sekwencji przedstawiający sposób interakcji pomiędzy użytkownikiem a komponentami MVC. Źródło: opracowanie własne ..... 27
Rysunek 14 Interfejs programu DBeaver. Źródło: opracowanie własne. ..... 39
Rysunek 15 Ekran IDE InteliJ IDEA. Źródło: opracowanie własne ..... 40
Rysunek 16 Ekran Android Studio. Źródło: opracowanie własne ..... 41
Rysunek 17 Diagram przypadków użycia aktora "Sprzedawca". Źródło: opracowanie własne ..... 46
Rysunek 18 Diagram przypadków użycia aktora "Projektant". Źródło: opracowanie własne 47
Rysunek 19 Diagram przypadków użycia aktora "Projektant". Źródło: opracowanie własne 48
Rysunek 20 Diagram przypadków użycia aktora "Administrator techniczny". Źródło: opracowanie własne ..... 49
Rysunek 21 Diagram sekwencji procesu dodawania produktu z wykorzystaniem wtyczki .... 56
Rysunek 22 Diagram sekwencji procesu zakupu produktu z wykorzystaniem linku afiliacyjnego ..... 57
Rysunek 23 Grafika przedstawiająca diagram komponentów systemu. Źródło: opracowanie własne ..... 58
Rysunek 24 Diagram pakietów warstwy widoku. Źródło: opracowanie własne ..... 59
Rysunek 25 Diagram pakietów aplikacji na platformę Android. Źródło: opracowanie własne ..... 60
Rysunek 26 Diagram pakietów wtyczki Chrome. Źródło: opracowanie własne ..... 61
Rysunek 27 Diagram pakietów aplikacji po stronie serwera. Źródło: opracowanie własne.... 62
Rysunek 28 Diagram związków encji systemu. Źródło: opracowanie własne ..... 63
Rysunek 29 Formularz logowania do aplikacji. Źródło: opracowanie własne ..... 97
Rysunek 30 Zrzut ekranu przedstawiający listę grup. Źródło: opracowanie własne ..... 97
Rysunek 31 Zrzut ekranu przedstawiający formularz edycji grupy. Źródło: opracowanie własne ..... 98

Rysunek 32 Zrzut ekranu przedstawiający formularz edycji konta użytkownika. Źródło: opracowanie własne ..... 98
Rysunek 33 Zrzut ekranu przedstawiający administratorską listę użytkowników w systemie 99
Rysunek 34 Zrzut ekranu przedstawiający formularz edycji programu partnerskiego. Źródło: własne ..... 99
Rysunek 35 Zrzut ekranu listy zestawów atrybutów. Źródło: opracowanie własne ..... 100
Rysunek 36 Zrzut ekranu edytora zestawu atrybutów. Źródło: opracowanie własne ..... 100
Rysunek 37 Fragment ekranu przedstawiający 2 rodzaje konstruktora atrybutów nieujęte powyżej. Źródło: opracowanie własne ..... 101
Rysunek 38 Zrzut ekranu przedstawiający listę projektów. Źródło: własne ..... 101
Rysunek 39 Pusty widok projektu, bez produktów. Źródło: własne ..... 102
Rysunek 40 Sekcja edycji ustawień projektu. Źródło: własne ..... 102
Rysunek 41 Podgląd uwag do projektu. Źródło: opracowanie własne ..... 103
Rysunek 42 Formularz dodawania projektu/wizualizacji. Źródło: własne ..... 104
Rysunek 43 Moment po uruchomieniu wtyczki. Źródło: własne ..... 104
Rysunek 44 Formularz dodawania produktu wewnątrz wtyczki. Źródło: opracowanie własne ..... 105
Rysunek 45 Sekcja atrybutów dodatkowych w formularzu. Źródło: własne ..... 106
Rysunek 46 Uzupełnianie wartości poprzez zaznaczenie. Źródło: opracowanie własne ..... 106
Rysunek 47 Strona przykładowego produktu. Źródło: opracowanie własne ..... 107
Rysunek 48 Sekcja uwag do projektu we wtyczce. Źródło: opracowanie własne ..... 108
Rysunek 49 Widok projektu wraz z nowododanymi produktami. Źródło: opracowanie własne ..... 109
Rysunek 50 Zrzut ekranu przedstawiający globalny spis produktów. Źródło: opracowanie własne ..... 110
Rysunek 51 Przykład wygenerowanego linku afiliacyjnego do strony z produktem. Źródło: opracowanie własne ..... 111
Rysunek 52 Widok fragmentu przykładowej strony landing-page. Źródło: opracowanie własne ..... 112
Rysunek 53 Wycinek przykładowej strony *purchase-page* z podsumowaniem. Źródło: opracowanie własne ..... 113
Rysunek 54 Statystyki użycia linków w ujęciu per projektant ..... 113
Rysunek 55 Statystyki użycia linków w ujęciu per sprzedawca ..... 113
Rysunek 56 Wybór projektu w aplikacji mobilnej. Źródło: opracowanie własne ..... 114
Rysunek 57 Formularz oddawania produktu z uzupełnionymi danymi. Źródło: opracowanie własne ..... 115
Rysunek 58 Zawartość pola opis przed aktualizacją. Źródło: własne ..... 116
Rysunek 59 Zawartość pola opis po aktualizacji. Źródło: własne ..... 116
Rysunek 60 Harmonogram rozwoju aplikacji. Źródło: opracowanie własne ..... 118

# Spis tabel

Tabela 1 Specyfikacja wymagań niefunkcjonalnych. Źródło: opracowanie własne ..... 45
Tabela 2 Przypadek użycia "Logowanie". Źródło: opracowanie własne ..... 50
Tabela 3 Przypadek użycia "Zmiana danych". Źródło: opracowanie własne ..... 50
Tabela 4 Przypadek użycia "Dodanie nowej grupy". Źródło: opracowanie własne ..... 51
Tabela 5 Przypadek użycia "Dodanie produktu przez wtyczkę". Źródło: opracowanie własne ..... 51
Tabela 6 Przypadek użycia "Dodanie projektu". Źródło: opracowanie własne ..... 52
Tabela 7 Przypadek użycia "Dodanie komentarza do projektu". Źródło: opracowanie własne ..... 52
Tabela 8 Przypadek użycia "Założenie programu partnerskiego". Źródło: opracowanie własne ..... 53
Tabela 9 Przypadek użycia "Generowanie linku afiliacyjnego". Źródło: opracowanie własne ..... 53
Tabela 10 Przypadek użycia "Pobieranie zestawienia statystyk". Źródło: opracowanie własne ..... 54
Tabela 11 Przypadek użycia "Tworzenie nowego zestawu atrybutów". Źródło: opracowanie własne ..... 54
Tabela 12 Przypadek użycia "Pobieranie wartości atrybutów przez wtyczkę/aplikację Android". Źródło: opracowanie własne ..... 55
Tabela 13 Struktura tabeli *user_account*. Źródło: opracowanie własne ..... 65
Tabela 14 Struktura tabeli *user_group*. Źródło: opracowanie własne ..... 66
Tabela 15 Struktura tabeli *user_group_member*. Źródło: opracowanie własne ..... 67
Tabela 16 Struktura tabeli *user_session*. Źródło: opracowanie własne ..... 67
Tabela 17 Struktura tabeli *user_key*. Źródło: opracowanie własne ..... 67
Tabela 18 Struktura tabeli *file*. Źródło: opracowanie własne ..... 68
Tabela 19 Struktura tabeli *category*. Źródło: opracowanie własne ..... 68
Tabela 20 Struktura tabeli *category_post*. Źródło: opracowanie własne ..... 69
Tabela 21 Struktura tabeli *section*. Źródło: opracowanie własne ..... 70
Tabela 22 Struktura tabeli *product*. Źródło: opracowanie własne ..... 70
Tabela 23 Struktura tabeli *attributes_set*. Źródło: opracowanie własne ..... 71
Tabela 24 Struktura tabeli *affiliate_program*. Źródło: opracowanie własne ..... 72
Tabela 25 Struktura tabeli *affiliate_link*. Źródło: opracowanie własne ..... 72
Tabela 26 Struktura tabeli *affiliate_link_usage*. Źródło: opracowanie własne ..... 73
Tabela 27 Opis metody POST api/internal/account. Źródło: opracowanie własne ..... 74
Tabela 28 Opis metody POST /api/internal/group/{id}. Źródło: opracowanie własne ..... 75
Tabela 29 Opis metody POST /api/internal/product. Źródło: opracowanie własne ..... 76
Tabela 30 Opis metody POST /api/internal/post. Źródło: opracowanie własne ..... 77
Tabela 31 Opis metody POST /api/external/affiliate-link. Źródło: opracowanie własne ..... 78
Tabela 32 Opis metody POST /api/external/affiliate-link/action/use. Źródło: opracowanie własne ..... 78
Tabela 33 Opis metody GET /api/internal/affiliate-statistics/{from}/{to}. Źródło: opracowanie własne ..... 79

# Spis listingów

Listing 1 Fragment strony HTML zawierającej znaczniki <*meta*> spełaniające standard Schema.org. Źródło: opracowanie własne ..... 24
Listing 2 Fragment strony HTML zawierającej skrypt json-ld spełniający standard Schema.org. Źródło: opracowanie własne ..... 24
Listing 3 Przykład sparametryzowanego żądania GET. Źródło: opracowanie własne..... 28
Listing 4 Przykład żądania POST widoczny w konsoli przeglądarki internetowej. Źródło: opracowanie własne ..... 28
Listing 5 Przykład żądania PUT widoczny w konsoli przeglądarki internetowej. Źródło: opracowanie własne ..... 29
Listing 6 Przykład żądania HEAD. Źródło: opracowanie własne ..... 29
Listing 7 Przykład żądania DELETE. Źródło: opracowanie własne ..... 29
Listing 8 Przykład klasy pełniącej funkcji serwisu utworzonej w języku Kotlin ..... 30
Listing 9 Przykład klasy *frameworka* oznaczony jako *@Service*. Źródło: opracowanie własne ..... 31
Listing 10 Przykład klasy oznaczony jako *@Controller*. Przedstawiono 2 punkty końcowe typu GET. Źródło: opracowanie własne ..... 32
Listing 11 Przykład klasy encji w *frameworku* Spring. Źródło: opracowanie własne..... 33
Listing 12 Fragment formularza (<*form*>) pliku HTML przedstawiający podstawowe elementy służace do wprowadzania danych (np. <*input*>, <*select*>). Zastosowano Bootstrap 5 jako bibliotekę styli *css*. Źródło: opracowanie własne..... 34
Listing 13 Sposób podłączenia zewnętrznego skryptu JS. Źródło: opracowanie własne ..... 35
Listing 14 Przykład funkcji napisanej w języku Javascript. Źródło: opracowanie własne..... 36
Listing 15 Definicja komponentu Vue. Źródło: opracowanie własne ..... 37
Listing 16 Przykład instrukcji SELECT w PostgreSQL. Źródło: opracowanie własne..... 38
Listing 17 Fragment zawartości pliku *docker-compose.yaml* opisujący usługę bazodanową. Źródło: opracowanie własne. ..... 64
Listing 18 Struktura odpowiedzi każdego żądania API w aplikacji. Źródło: opracowanie własne..... 74
Listing 19 Zawartość afiliacyjnego skryptu js – klasy zdarzeń. Źródło: opracowanie własne 80
Listing 20 Zawartość afiliacyjnego skryptu js – zarys ogólny. Źródło: opracowanie własne . 81
Listing 21 Zawartość struktury eventu zamieszczonego na stronie *purchase-page*. Źródło: opracowanie własne ..... 83
Listing 22 Fragment kodu odpowiadający za komponent edytora atrybutów. Źródło: opracowanie własne ..... 84
Listing 23 Implementacja komponentu *SelectItem*. Źródło: opracowanie własne..... 85
Listing 24 Fragment serwisu odpowiadającego za generowanie linków afiliacyjnych. Źródło: opracowanie własne ..... 88
Listing 25 Przykład obiektu DTO odpowiadające za dostarczenie danych żądania przesłanych do serwera. Źródło: opracowanie własne..... 89
Listing 26 Fragment serwisu *ProductAddCmd*. Źródło: opracowanie własne..... 89
Listing 27 Fragment funkcji *scrapData*. Źródło: opracowanie własne ..... 90
Listing 28 Fragment funkcji *parseSchema*. Źródło: opracowanie własne ..... 92
Listing 29 Przykład danych schema.org w formacie json-ld umieszczonej na przetwarzanej stronie internetowej. Źródło: opracowanie własne ..... 92
Listing 30 Zawartość komponentu wywołanego przez *AttributesItemLoader - SelectItem*. Źródło: opracowanie własne ..... 93

Listing 31 Fragment metody *scrapAttributes*. Źródło: opracowanie własne ..... 94
Listing 32 Fragment aktywności *ContentSelectorActivity*. Źródło: opracowanie własne ..... 95
Listing 33 Fragment przykładu kodu strony *landing-page*. Źródło: opracowanie własne .... 111
Listing 34 Fragment przykładu kodu strony purchase-page. Źródło: opracowanie własne... 112