
POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

**Wydział Informatyki**

**Katedra Inżynierii Oprogramowania**

Inżynieria Oprogramowania i Baz Danych

**Jarosław Słabik**
Nr albumu 20833

# Elastyczne środowisko programistyczne działające w chmurze

Praca magisterska

dr inż. Mariusz Trzaska

Warszawa, Luty, 2022

## Streszczenie

Niniejsza praca dotyczy istotnego praktycznego problemu wspomagania programistów piszących oprogramowanie na duże maszyny, czyli takie, które spotykamy w halach produkcyjnych, tak by wyeliminować konieczność fizycznej obecności przy tej maszynie. Dotychczasowe doświadczenia w tego rodzaju branżach okazują się niezbyt adekwatne w stosunku do wyzwania, jakie stawia praca zdalna. Kolejnym ważnym i wartym podkreślenia jest problem środowiska pracy programu, gdzie nawet mała różnica pomiędzy środowiskiem programistycznym a produkcyjnym może spowodować awarię maszyny. Inną kwestią poruszaną w tej pracy jest dostrzeżenie wielości programów, w jakie programista musi się zaopatrzyć, aby móc wykonywać swoją pracę w sposób efektywny. Bywa że dla każdego języka programowania dedykowany jest osobny program. W przypadku *full–stack* dewelopera mogą to być nawet 4. osobne, zgoła inne programy. Wraz ze wzrostem zainteresowania rozwiązaniami chmurowymi istnieje konieczność opracowania elastycznego środowiska opierającego się o technologie chmurowe. W pracy przedstawiono propozycje zintegrowanego środowiska programistycznego zwanego IDE (od ang. *Integrated Development Environment*) działającego w chmurze, dostępnego z poziomu przeglądarki internetowej. Efektem ubocznym pracy jest prototyp systemu umożliwiający tworzenie aplikacji w C++ współpracującego z bazą danych PostgreSQL. Prototyp opiera się na trójwarstwowej aplikacji webowej. Całość jest wspierana wtyczkami, gdzie każda wtyczka odpowiada innej technologii. Na potrzeby prototypu są to: wtyczka dla C++, wtyczka dla bazy danych PostgreSQL. Prototyp został przetestowany, tworząc w nim proste aplikacje, współpracujące z relacyjną bazą danych. Na etapie końcowym prototyp został przeanalizowany na tle istniejących rozwiązań oraz zostały skonstruowane wnioski i przemyślenia.



**Słowa kluczowe: IDE, chmura, wtyczki**



## Podziękowania

Autor pracy pragnie wyrazić podziękowanie panu dr. Mariuszowi Trzasce za udzielanie cennych rad i wskazówek, pomocnych przy pisaniu niniejszej pracy.

# Spis treści

# Wstęp

Narzędzia wspierające programistów towarzyszyły im od skonstruowania pierwszego programowalnego komputera. Najpierw był to papierowy zeszyt z zapisaną sekwencją zer i jedynek, a przy niej opis, czym dana sekwencja skutkuje. Przy powstaniu pierwszego języka programowania, jakim był Assembler, powstało pierwsze narzędzie do wspomagania programistów, był to kompilator. Pierwsze zintegrowane środowisko programistyczne do tworzenia programów, było napisane w języku Dartmouth BASIC, był to zwykły notatnik działający w terminalu umożliwiającym wykonywanie poleceń bez wychodzenia z niego. Z czasem przy kolejnych implementacjach IDE dodawano coraz to więcej usprawniających pracę programistów narzędzi. Jednym z pierwszych środowisk programistycznych ze wsparciem wtyczek było Softbench. Został zaprezentowany w 1990 r. [1] przez magazyn Hewlett-Packard Journal. W następnych latach dodawano: kolorowanie składni, obsługę kontroli wersji, debugowanie, analizę semantyczną i wiele innych. Pierwsze IDE bazujące na technologiach internetu pojawiło się w 2009 r. – JavaWIDE, wspierał on wyłącznie technologię Java. Następnym był CODERUN od Microsoft. Tak jak w przypadku wcześniejszego przykładu, był zaprojektowany wyłącznie dla jednej technologii, tym razem dla C#. Obydwa projekty upadły, ale pomysł przetrwał. Aktualnie istnieje kilka IDE bazujących na technologii internetu, są to m. in.: AWS Cloud9 [2], Eclipse Che (dawniej Codenvy) [3], Codeanywhere [4], PaizaCloud [5], Replit [6], SourceLair IDE [7], jednak każdy z nich ma jakieś wady. Najciekawsze ze środowisk zostały szczegółowo opisane w pracy.

Celem pracy było opracowanie koncepcji elastycznego, zintegrowanego środowiska programistycznego, wykorzystującego wtyczki oraz działającego w internetowej chmurze wzorując się na edytorze Vim (Vi iMproved), obsługującym komendy tekstowe. W tym celu zaprojektowany został i zaimplementowany system zawierający klienta jako aplikację jednostronicową (SPA – *Single Page Application*), wizualnie i ideologicznie przypominającą edytor Vim, usługę sieciową (WS – *Web Service*) REST – jako *back-end* oraz organizer wtyczek, który współpracuje z relacyjną bazą danych.

Klient odpowiada za warstwę wizualną, komunikację z użytkownikiem oraz komunikuje się z WS wysyłając żądania. Web Serwis analizuje dostarczone do niego dane i komendy, operuje na strukturze plików i folderów, a następnie wykorzystuje wybraną wtyczkę do realizacji powierzonego mu celu. Używa relacyjnej bazy danych do zapisania ustawień, przynależności wtyczek do projektów i użytkowników oraz przynależności użytkowników do projektów. Prototyp został nazwany „WebVimIDE”.

Wszystkie rysunki i diagramy zamieszczone w tekście zostały stworzone przez autora na użytek niniejszej pracy dyplomowej.

# 1. Wprowadzenie do problematyki

W tym rozdziale został przedstawiony problem którego dotyczy niniejsza praca. Następnie przedstawiono szczegółowo konkurencyjne rozwiązania i przedstawiono subiektywne wrażenia z ich użytkowania.

## 1.1. Problematyka

Programiści podczas swojej pracy wykorzystują wiele narzędzi, m. in. wyspecjalizowane edytory tekstu oraz zintegrowane środowiska deweloperskie.

W dzisiejszych czasach IDE jest to złożony, wypełniony funkcjonalnościami program służący do tworzenia oprogramowania. Zintegrowane środowiska są przystosowane do pracy z jednym lub z góry ustalonymi technologiami, stąd programiści używający kilku różnych technologii, zwłaszcza tych, które ze sobą nie współpracują, są skazani na posiadanie kilku takich, dość ciężkich i powolnych aplikacji. Każda z tych aplikacji może być inna i do każdej trzeba się przyzwyczaić i jej nauczyć. Przełączanie się między IDE potrafi wybić programistę z rytmu pracy.

Edytory tekstu dla programistów są lżejsze, niemniej wymagają od użytkownika konfiguracji pod oczekiwaną technologię. Większość edytorów wspiera wtyczki, które same konfigurują aplikację, zmieniając jej wygląd i funkcjonalności. Dzięki temu pojawia się jedna aplikacja do wielu technologii.

W dobie przenoszenia każdego typu oprogramowania do chmury, również i narzędzia dla deweloperów doczekały się swoich webowych odpowiedników. Od stworzenia pierwszego eksperymentalnego IDE w 2009 r. aż po dzień napisania niniejszej pracy nieustannie są wykonywane próby zaimplementowania webowego środowiska programistycznego, które może przewyższyć desktopowe odpowiedniki. Aktualnie istnieje kilka webowych środowisk deweloperskich, ale tylko niektóre są godne polecenia dla pracy programisty. Większość z nich to tylko nakładki na kompilatory czy też interpretatory.

Należy zaznaczyć że na tę chwilę brakuje na rynku webowego IDE o lekkości edytora tekstu, który można byłoby dowolnie konfigurować za pomocą wtyczek, a zarazem będącego prostym w budowie i który nie dekoncentrowałby wyskakującymi natrętnie okienkami, tak aby można było obsłużyć całe środowisko nie używając komputerowej myszy. Webowe środowiska podzielone są na dwie osobne części: edytor oraz panel zarządzania użytkownikami i środowiskami każda z nich w oddzielnym oknie. Praca przy wielu projektach staje się przez to uciążliwa wymuszając na użytkowniku przeskakiwanie z edytora do panelu, by następnie ponownie do niego powrócić. Warto naświetlić też sporą, potencjalną grupę odbiorców dla których każda minuta pracy jest jak na wagę złota [8]. Wspomniani użytkownicy edytują tekst w tempie myśli, wprowadzając powyżej 200 znaków na minutę, a każde wytrącenie ich z tego rytmu powoduje przerwanie myśli, a jak wiemy myśli bywają ulotne i nieprzywracalne.

Idealnym rozwiązaniem wydaje się emulacja trybu tekstowego w przeglądarce łącząc w sobie prostotę i lekkość wynikającą z małej ilości elementów do wygenerowania. Za wspomnianym trybem przemawia również brak wyskakujących okienek i czysty, schludny wygląd. Największym atutem tego rozwiązania jest możliwość operowania wyłącznie samą klawiaturą.

## 1.2. Konkurencyjne rozwiązania

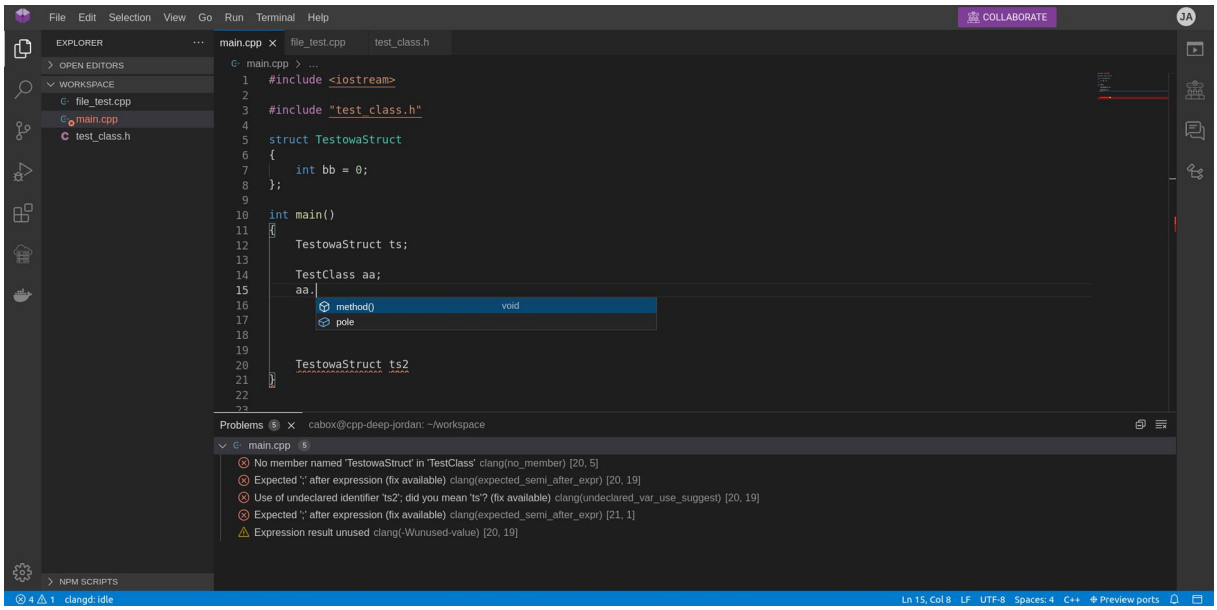
Na rynku dostępnych jest kilka webowych środowisk deweloperskich godnych polecenia, niemniej każde z nich posiada mankamenty, zostały one opisane w osobnych paragrafach. Musimy zaznaczyć, że wszystkie środowiska były analizowane na testowych kontach z darmową subskrypcją, co mogło skutkować gorszą wydajnością, i ostatecznie również gorszymi subiektywnymi wrażeniami autora.

### 1.2.1 Codeanywhere

Interfejs użytkownika został podzielony na dwie części: IDE oraz panel administracyjny zwany dalej *dashboard*. Wygląd oraz mechanizmy *front-end*'u środowiska zostały zapożyczone z Visual Studio Code (VS Code). Składa się on z siedmiu elementów jakimi są:

- lewy pasek aktywności – pozwala przełączać się między widokami i zapewnia dodatkowe wskaźniki/ widoki zaimplementowane we wtyczkach należących do VS Code;
- lewy pasek boczny – zawiera różne zakładki, takie jak drzewo plików, otwarte pliki, czy też ostatnio otwierane pliki;
- pasek stanu – wyświetla informacje o edytowanym pliku, pozycje karetki oraz stanu projektu;
- panele – grupa paneli domyślnie usytuowana jest na dole obszaru edytora, pozwalając na uzyskanie danych wyjściowych, debugowanie, wyświetlanie błędów i ostrzeżeń. Jeden z tych paneli to zintegrowany terminal;
- edytor – główny obszar do edycji plików;
- prawy pasek aktywności – zawiera aktywności niewystępujące w VS Code tj.: przeglądarka internetowa, zarządzanie użytkownikami zaproszonymi do współpracy, czat;
- prawy pasek boczny – wyświetla wybraną aktywność z prawego paska aktywności.

Rysunek 1. przedstawia IDE, jeden z dwóch elementów interfejsu użytkownika zaprezentowanego przez Codeanywhere.

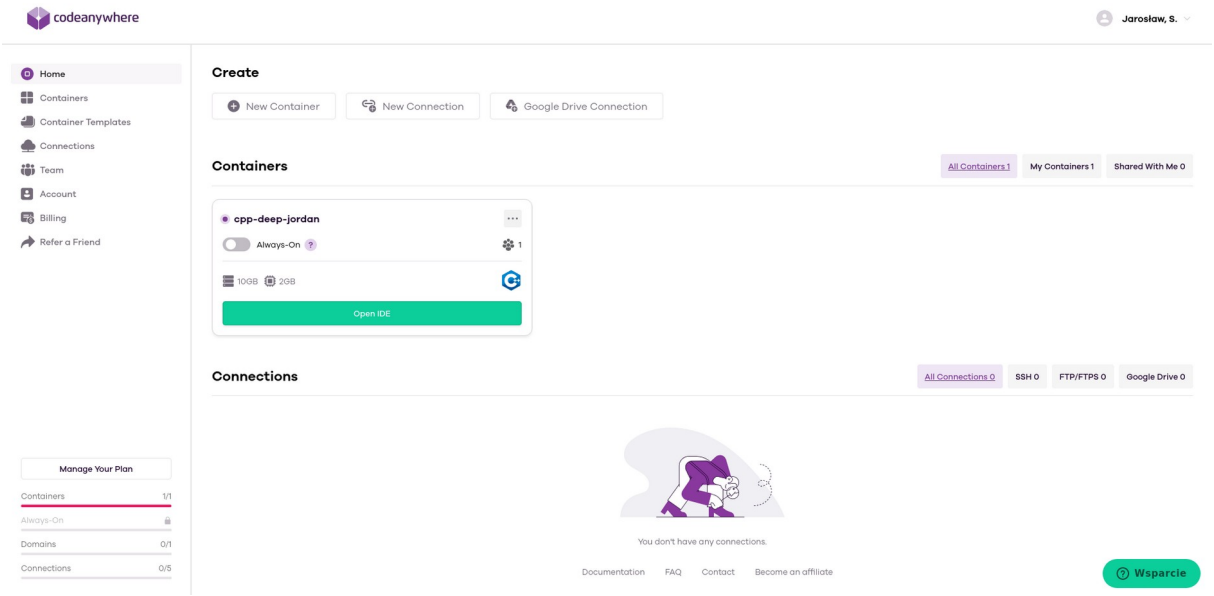



Rysunek 1. Zrzut ekranu IDE należącego do Codeanywhere

Następnym i ostatnim elementem interfejsu jest *dashboard*, który poza informacjami o zalogowanym użytkowniku i aktywnym planie subskrypcji zawiera następujące zakładki:

- kontenery – dodawanie, usuwanie i zarządzanie kontenerami, w których znajdują się projekty;
- szablony kontenerów – dodawanie, modyfikowanie oraz usuwanie szablonów. Szablon jest to w gruncie rzeczy Dockerfile, na podstawie którego jest robiony obraz wykorzystywany w kontenerze. Istnieje kilka predefiniowanych środowisk najpopularniejszych języków;
- połączenia – zarządzanie połączeniami SSH, FTP oraz Google Drive prowadzącymi do lokalizacji plików;
- zespół – zarządzanie zaproszonymi użytkownikami do współpracy;
- konto – zbiór danych o koncie, gdzie możliwa jest zmiana danych;
- płatności – sposób płatności, historia płatności oraz rejestracja karty płatniczej.

Na Rysunku 2. został przedstawiony panel, element interfejsu serwisu Codeanywhere, w którym można zarządzać całym środowiskiem, użytkownikami i płatnościami.




Rysunek 2. Zrzut ekranu panelu należącego do Codeanywhere

IDE wspiera ponad 120 języków programowania ale tylko kilkanaście najpopularniejszych z nich otrzymało pełne wsparcie.

Istnieje obsługa wtyczek znanych z VS Code, rozszerzenia można instalować z *Marketplace* lub ręcznie za pomocą pliku o rozszerzeniu vsix.

Edytor posiada liczne funkcje wspomagania edycji kodu. Wyszczególnia się: auto-uzupełnianie, analizator kodu (*linter*/ *code analysis*), auto-formatowanie, refaktoryzacje symboli oraz nawigacje. Ta ostatnia funkcja zawiera w sobie: wyszukiwanie i skok do wybranej referencji, skok do definicji oraz skok do deklaracji.

Podczas korzystania z systemu przez siedem dni (gdyż tyle trwa testowy darmowy plan) autor pracy zebrał następujące wrażenia:

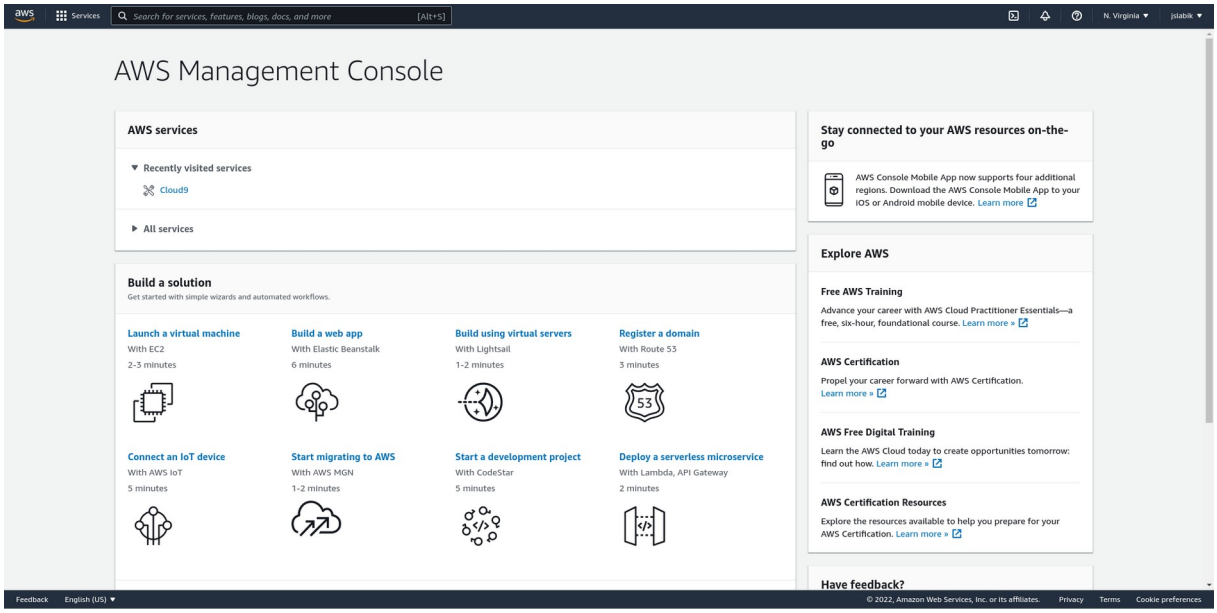
- w porównaniu z płynną edycją widoczne są spowolnienia, zwłaszcza przy tworzeniu nowego pliku (3.50 s.) i przy auto-uzupełnianiu (1 s.);
- uciążliwe przełączanie się pomiędzy projektami;
- tworząc projekt należy już na samym początku wybrać środowisko w znaczeniu kontener, którego w późniejszym czasie nie można zmienić. Stąd projekt jest ściśle powiązany ze środowiskiem. Przykład, jest tworzony projekt „A” w oparciu o środowisko z g++ w wersji 4.8 ale chcąc przetestować projekt w nowszym standardzie należy utworzyć nowy pusty projekt „B”, połączyć go ze środowiskiem zawierającym g++ w wersji 9.3, a następnie skopiować zawartość projektu „A” i wkleić do projektu „B”. Zamiast mechanizmu kopiowania można użyć narzędzia GIT;
- występuje mało przestrzeni roboczej gdy otwarte jest drzewo projektu i czat;
- wydaje się idealne jako narzędzie do przeprowadzania rozmów rekrutacyjnych *on-line*.

### 1.2.2 AWS Cloud9

Środowisko jest integralną częścią usług AWS od Amazon i widać to spoglądając na interfejs użytkownika, pełno tutaj odniesień do różnych usług, samouczków czy całych platform AWS tj. *AWS Lambda*. Całość mieści się w autorskiej platformie *Amazon Elastic Compute Cloud* zwane dalej jako EC2. Interfejs został podzielony na trzy części:

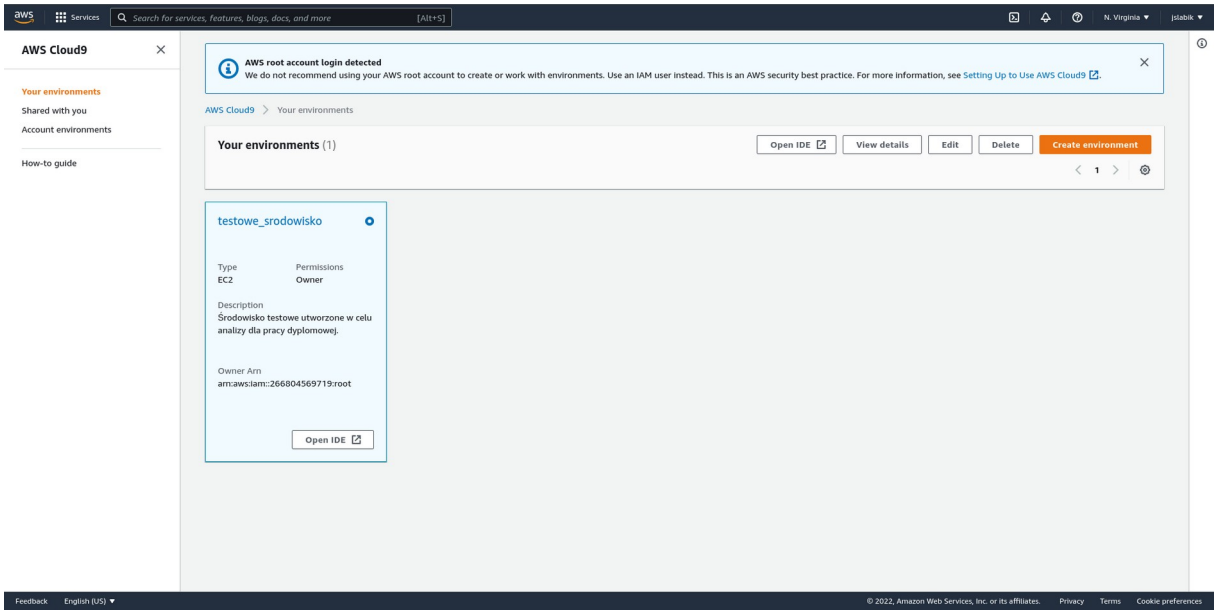
- panel główny zwany *AWS Management Console* – jest to panel, do którego użytkownik kierowany jest po zalogowaniu. Z tego miejsca może przejść do IDE lub innych usług AWS;
- panel środowisk – panel z listą utworzonych środowisk. Środowisko jest to wirtualny serwer dla którego można ustawiać fizyczne parametry, jak ilość pamięci RAM czy ilość rdzeni procesora a nawet system jaki ma się na nim znajdować. Z tego miejsca można przejść do edytora;
- IDE – środowisko deweloperskie, które zostało opisane poniżej.

Rysunek 3. przedstawia główny panel z którego można przejść do AWS Cloud9 lub innych usług należących do Amazon. Można tutaj znaleźć również wszelakie poradniki dotyczące tworzenia aplikacji z wykorzystaniem usług AWS.



Rysunek 3. Zrzut ekranu głównego panelu AWS

Panel z środowiskami został przedstawiony na Rysunku 4. Widać na nim listę środowisk oraz ich opis w niebieskim prostokącie.

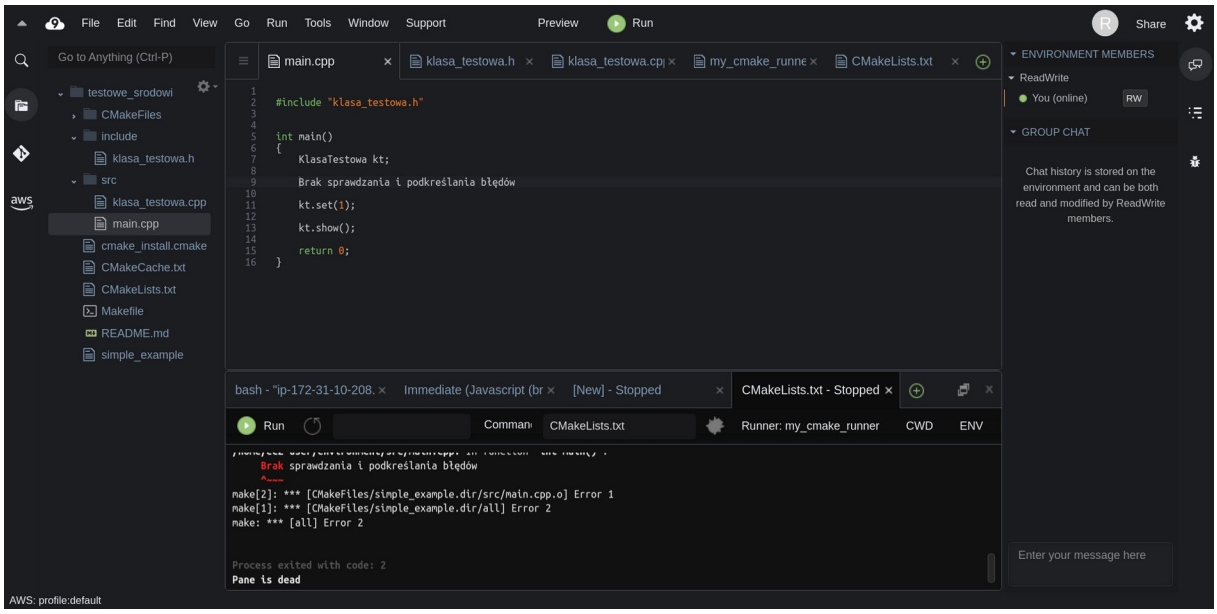



Rysunek 4. Zrzut ekranu panelu z środowiskami od AWS

Interfejs użytkownika samego zintegrowanego środowiska wyglądem przypomina klasyczne desktopowe IDE. Składa się z trzech belek i czterech skalowalnych paneli:

- belka górna – jako jedyna belka posiada możliwość chowania się. Zawiera wiele przydatnych opcji, w tym kompilacje i uruchomienie programu oraz preferencje;
- belka boczna lewa – zawiera następujące aktywności: wyszukiwarka fraz, drzewo projektu, kontrola wersji oraz integracja projektu z innymi usługami AWS;
- lewy panel – zawiera wybraną aktywność z lewej bocznej belki;
- belka boczna prawa – zawiera następujące aktywności: czat, lista tagów oraz debuger;
- prawy panel – zawiera wybraną aktywność z prawej bocznej belki;
- środkowy panel – to główna przestrzeń robocza do edytowania plików;
- dolny panel – umożliwia uruchamianie zakładek z panelami, które pozwalają na uzyskanie danych wyjściowych, debugowanie, wyświetlanie błędów i ostrzeżeń. Jeden z tych paneli to zintegrowany terminal.

Rysunek 5. ukazuje edytor w omawianym IDE podczas pracy, widać na nim brak możliwości sprawdzania i podkreślania błędów oraz wynik procesu kompilacji.




Rysunek 5. Zrzut ekranu IDE od AWS

IDE wspiera tylko technologie wykorzystywane w Amazon. Jest ich 16 a wśród nich są m. in.: C++, Java, Python oraz JS. Większość języków programowania nie otrzymała pełnego wsparcia [9].

Występuje tutaj brak obsługi wtyczek. Środowisko deweloperskie jest ukierunkowane wyłącznie na technologię i użytkowników z obszaru AWS.

Edytor posiada kilka przydatnych funkcji do wspomagania edycji kodu, są to: kolorowanie składni, auto-formatowanie, podpowiadanie składni, refaktoryzacja symboli oraz nawigacja. Ta ostatnia zawiera skok do definicji oraz listę referencji.

Podczas korzystania z systemu i tworząc proste aplikacje, autor pracy zebrał następujące wrażenia:

- edytor działa nadzwyczaj płynnie jak na darmową subskrypcję;
- występuje uciążliwe przełączanie się pomiędzy projektami. Zastosowano tu strategię jeden projekt – jedno środowisko. W celu przełączenia się do innego projektu należy cofnąć się do panelu ze środowiskami;
- tworząc projekt tworzy się również fizyczne środowisko i wydaje się to być właściwym podejściem do tworzenia wydajnych aplikacji na środowiska z małymi zasobami;
- sprawdzanie i podkreślanie błędów, refaktoryzacja oraz nawigacja zostały zaimplementowane wyłącznie dla wybranych języków jak Python czy JavaScript. Dla C++ pojawia się komunikat *„"c_cpp" is not supported yet”*;
- brak podpowiadania składni przy importowaniu bibliotek w Python;
- liczne wsparcia dla integracji z usługami od AWS;
- dobre IDE dla twórców usług sieciowych od Amazona.

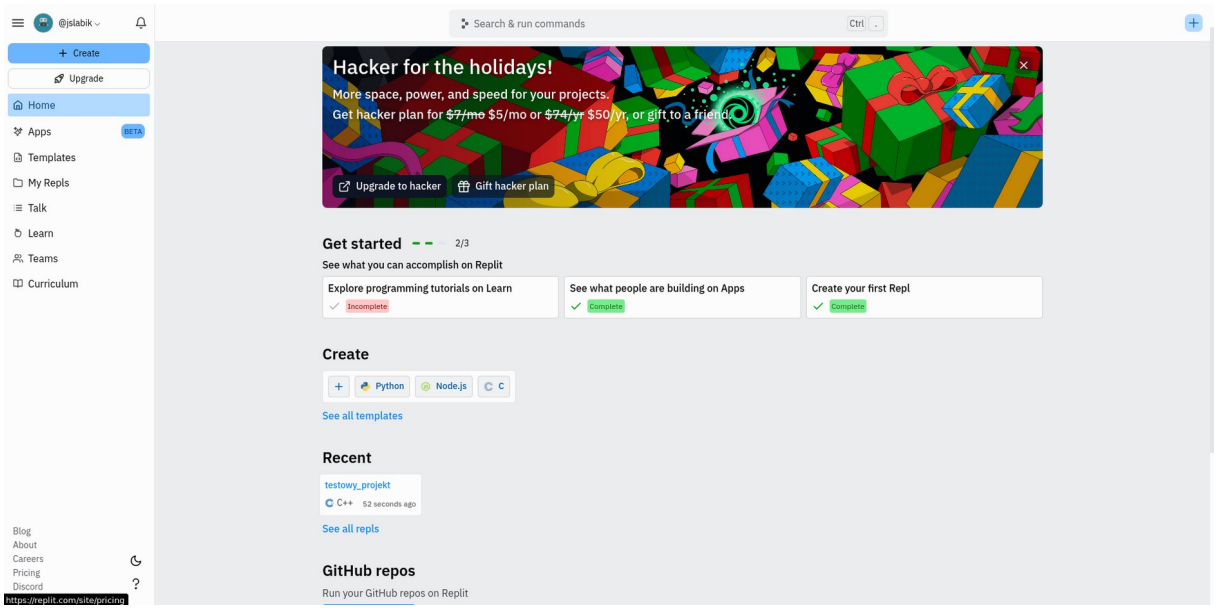
### 1.2.3 Replit

Na pierwszy rzut oka interfejs użytkownika przypomina zwykłą graficzną nakładkę na kompilator. Przy bliższym zapoznaniu okazuje się to być IDE z wieloma kluczowymi mechanizmami. Cały serwis podzielony jest na dwa widoki: panel oraz zintegrowane środowisko programistyczne. Od razu po zalogowaniu system prowadzi do panelu, w którym w głównym obszarze znajdują się widok

z samouczkami, projektami oraz przyciskami do utworzenia nowego projektu. Z lewej strony panelu znajduje się menu z następującymi zakładkami:

- *Home* – widok domowy widoczny od razu po zalogowaniu;
- *Apps* – widok zawierający przykładowe aplikacje stworzone za pomocą tego IDE;
- *Tamplates* – widok z listą predefiniowanych środowisk wraz z sugerowanymi szablonami aplikacji, jako szablony dostępne są też proste gry;
- *My Repls* – widok z listą projektów;
- *Talk* – widok mieści w sobie mikro-serwis społecznościowy umożliwiający mikroblogowanie przypomina Twittera;
- *Learn* – widok z listą poradników;
- *Teams* – zarządzanie członkami zespołu;
- *Curriculum* – lista lekcji pogrupowanych wg technologii, stworzona aby inspirować do nauki programowania.

Panel z widokiem Home został przedstawiony jako zrzut ekranu na Rysunku 6. Warto zauważyć że na załączonym obrazku znajdują się aż trzy przyciski do utworzenia nowego projektu. Kolejną rzeczą wartą uwagi są liczne odnośniki do przykładów i poradników. Twórcy serwisu nałożyli duży nacisk na naukę programowania i łatwe wdrożenie w prezentowane środowisko.




Rysunek 6. Zrzut ekranu panelu Replit

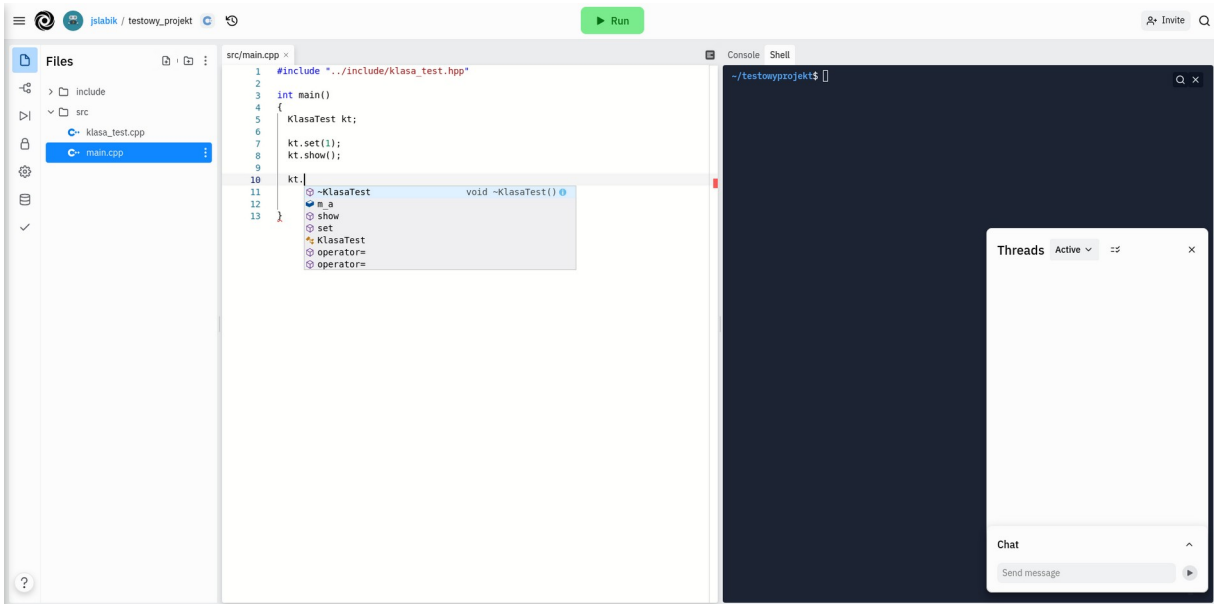
Interfejs użytkownika IDE otrzymał skromną i czystą powłokę graficzną na którą składa się pięć elementów:

- pasek górny – zawiera odsyłacz do panelu z widokiem *Home*, aktualny projekt, duży zielony przycisk do kompilacji i uruchamiania, przycisk do zapraszania nowych członków do projektu oraz przycisk z ikoną lupy do szukania plików;
- panel boczny lewy – mieści w sobie pasek z aktywnościami oraz obszar ze zawartością wybranej aktywności. Aktywności są następujące:
    1. drzewo projektu,

2. kontrola wersji,
3. *debagger*,
4. zmienne środowiskowe systemu,
5. ustawienia,
6. baza danych typu *key-value*,
7. testy jednostkowe;

- panel boczny prawy – zawiera zakładki *Console* oraz *Shell*. Pierwsza realizuje instrukcje wejścia i wyjścia z uruchomionym programem, natomiast druga jest odpowiednikiem terminala;
- panel środkowy – mieści się tutaj edytor kodu;
- panel czatu – okno umieszczone w prawym dolnym rogu umożliwia czat ze zaproszonymi członkami zespołu.

Środowisko zostało zaprezentowane w Rysunku 7. Warto zauważyć obecne podkreślanie błędów oraz auto-uzupełnianie.




Rysunek 7. Zrzut ekranu IDE Replit

IDE wspiera w pełni 52 języki dzięki użyciu LSP. Dla języka C++ został użyty serwer o nazwie cquery, który nie jest już wspierany od trzech lat. To może tworzyć problemy.

Niestety występuje tutaj brak obsługi wtyczek. W omawianym przypadku wielce przydatny byłby generyczny charakter środowiska. Za pomocą *plugin*u można było by zamienić wspomniany wyżej serwer języka cquery na bardziej współczesny, np. clangd.

Edytor posiada sporo przydatnych funkcji m. in. są to: podkreślanie błędów i ostrzeżeń, kolorowanie składni, auto-formatowanie, podpowiadanie składni, refaktoryzacja symboli, aktualne pozycje karetek członków zespołu oraz nawigacja. Ta ostatnia zawiera skok do definicji oraz listę referencji.

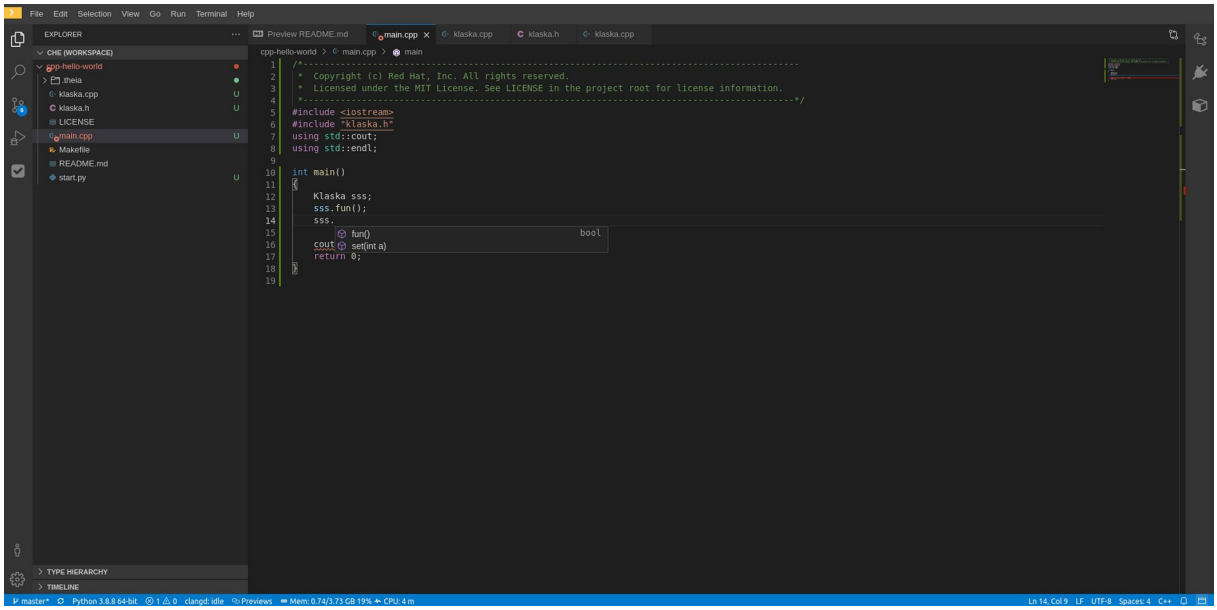
Podczas korzystania z systemu i tworząc proste aplikacje autor pracy zebrał następujące wrażenia:

- brak zauważonych opóźnień podczas korzystania z IDE;

- mechanizmy dostarczane przez LSP zacinają się wówczas wymagany jest restart przeglądarki;
- auto-uzupełnianie sugeruje również prywatne pola klas;
- występuje tutaj łatwe wdrożenie w środowisko dzięki samouczkowi, który jest uruchamiany przy pierwszym zalogowaniu, spora liczba poradników też jest przydatna;
- wykorzystywanie starych, niewspieranych narzędzi;
- każdy projekt otrzymuje dedykowane środowisko uruchomieniowe oraz bazę danych.

### 1.2.4 Eclipse Che

Interfejs użytkownika podobnie jak w przypadku serwisu Codeanywhere został podzielony na dwie części, IDE oraz panel administracyjny. W roli środowiska programistycznego został użyty Eclipse Theia [10], który to z kolei wykorzystuje kody źródłowe VS Code. Różnica z wcześniej przytoczoną pozycją to tylko ikony aktywności. Brakuje tutaj zarządzania użytkownikami, czy czatu. Natomiast pojawia się aktywność z możliwością instalacji własnościowych wtyczek oraz aktywność umożliwiająca zarządzanie kontem. Omawiane środowisko zostało przedstawione na Rysunku 8.

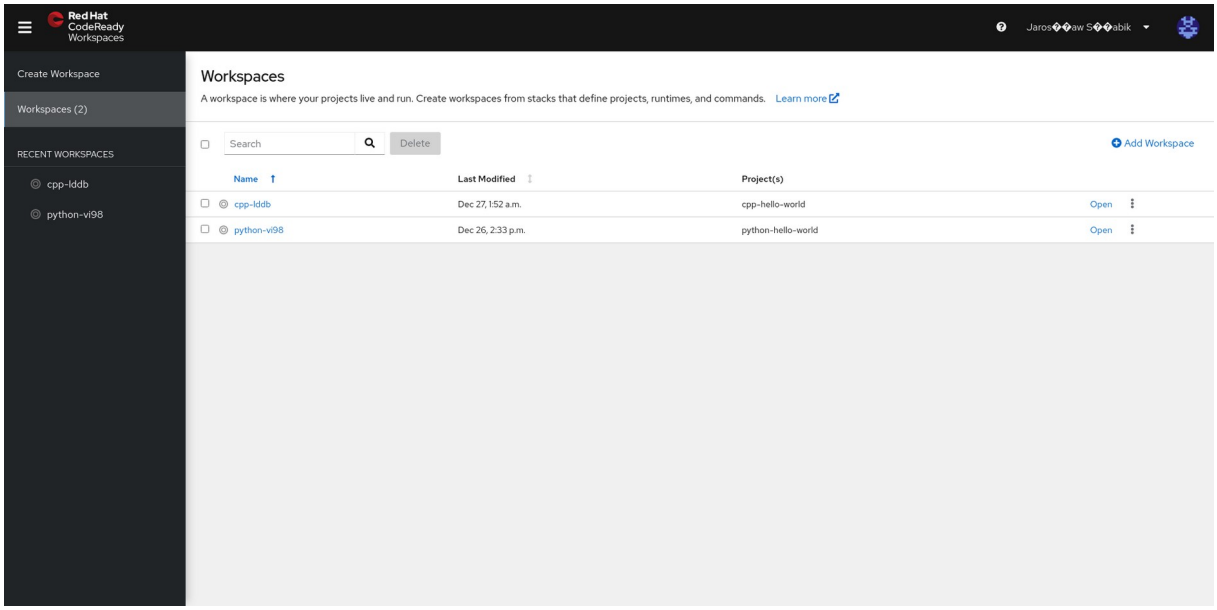



Rysunek 8. Zrzut ekranu IDE należącego do Eclipse Che

Panel administratora różni się od tego z zaprezentowanego w Codeanywhere. Został on podzielony na następujące zakładki:

- *home* – odwołanie do przestrzeni roboczych;
- przestrzenie robocze – tworzenie na podstawie stosu, zarządzanie i uruchamianie przestrzeni roboczych (przestrzeń robocza to kontener), występuje tu możliwość przypięcia projektów do kilku przestrzeni roboczych;
- stosy – tworzenie stosu i modyfikowanie stosu (stos to Dockerfile), brak w testowanej wersji, natomiast istnieją predefiniowane stosy;
- fabryki – modyfikowanie szczegółowych ustawień przestrzeni roboczych tj.: ilość dostępnego RAMu czy ręczne konfigurowanie środowiskiem roboczym, brak w testowanej wersji;
- administracja – zmiana danych użytkownika, usunięcie konta oraz konfiguracja autoryzacji. brak w testowanej wersji.

Omawiany wyżej panel został przedstawiony na Rysunku 9. z uruchomioną zakładką przestrzenie robocze.




Rysunek 9. Zrzut ekranu panelu administratora należącego do środowiska Eclipse Che

IDE wspiera ponad 120 języków programowania ale tylko kilka z nich otrzymało pełne wsparcie. Listę języków z pełnym wsparciem można rozszerzać za pomocą wtyczek.

Istnieje obsługa wtyczek własnościowych oraz tych znanych z VS Code, z czego te pierwsze działają bezproblemowo.

Edytor posiada funkcje wspomagania edycji kodu tj.: auto-uzupełnianie, *refaktoring* oraz nawigację. Ta ostatnia funkcja zawiera w sobie: wyszukiwanie i skok do wybranej referencji, skok do definicji oraz skok do deklaracji.

Podczas korzystania z systemu w wersji sandbox na platformie OpenShift autor pracy zebrał następujące wrażenia:

- środowisko działa płynnie, ale tylko dla pamięci trwałej widoczne są spowolnienia, zwłaszcza przy tworzeniu nowego pliku (3 s.) i przy auto-uzupełnianiu (1.30 s.),
- podczas korzystania z IDE do *Marketplace* zostało załadowanych tylko 35 wtyczek ze 133.,
- występuje tutaj sporo predefiniowanych stosów,
- widoczne jest tu łatwe przełączanie się pomiędzy projektami, gdyż znajdują się w jednym drzewie plików, ale przełączanie się pomiędzy obszarami roboczymi wymaga wejście w panel administratora,
- istnieje tu możliwość wyboru typu pamięci danego obszaru roboczego i możliwości są trzy [11]:
    1. trwała – pliki, folder i ich zawartości zapisywane są do fizycznej pamięci. Restart środowiska nie powoduje usunięcie plików;
    2. ulotna – zmiany przechowywane są w pamięci przeglądarki. Restart IDE powoduje utracenie wszystkich niezakomitowanych zmian;

3. asynchroniczna (funkcja eksperymentalna) – połączenie pamięci trwałej i ulotnej. Zmiany przechowywane są w pamięci ulotnej która to, co pewien czas jest synchronizowana z pamięcią trwałą.

- brak szczegółowych informacji o wtyczkach, dostępna jest tylko nazwa, wersja i opis.

# 2. Analiza dziedziny

Rozdział drugi niniejszej pracy został poświęcony przedstawieniu koncepcji środowiska programistycznego działającego w chmurze. Zostały poruszone tutaj tematy wybranej architektury, sposobie poruszania się po IDE oraz modelu danych. Istotnym zagadnieniem rozdziału jest przedstawienie specyfikacji funkcjonalnej i pozafunkcjonalnej. Na etapie końcowym zostały przedstawione diagramy przepływów danych oraz wady i zalety zaproponowanego rozwiązania.

Modelowanie, które zostało przedstawione w tym rozdziale zostało wykonane metodą strukturalną, ponieważ jest ona lepiej dostosowana do relacyjnego modelu danych, niż podejście obiektowe [12].

## 2.1. Główny zarys koncepcji IDE

Środowisko programistyczne oprócz wcześniej wspomnianych cech, tj. działanie w chmurze, czy obsługa wtyczek wykazuje się również modalnością. Modalność w przypadku edytorów tekstu jest to cecha niezwykle rzadka. Edytory tego typu oferują wiele trybów interakcji zoptymalizowanych pod kątem określonych rodzajów działań m. in.: wstawiania, zaznaczania, tryb komendy oraz tryb normalny [13]. Uzasadnieniem jest to, że każdy tryb jest precyzyjnie dostrojonym narzędziem, pozwalającym użytkownikowi realizować swoje cele w wydajny sposób. Dokonano wszelkich starań, aby IDE było obsługiwane głównie za pomocą samej klawiatury. Wybór ten został dokonany na podstawie obserwacji edytora tekstu, jakim jest Vim. Według Stackowerflow [14] Vim znajduje się na 5. miejscu pod względem liczby użytkowników, jednak uwzględniając wszystkie pochodne Vima plasowałby się na 4. miejscu.

Wspomniany edytor jest konsolowym modalnym, edytorem tekstu, rozwijanym od 1991 r., to tutaj pojawia się przesłanie, aby cały edytor mógł być obsłużony jedynie za pomocą klawiatury oraz posiadał tryby pracy. Wymienione cechy nie miały na celu przyspieszenia mechanicznego wprowadzania kodu, a jego szybką manipulację czy poruszaniu się po nim. Celem było również wyeliminowanie ruchu ręką z klawiatury na myszkę, a następnie z powrotem, aby nie przerywać myśli w rozumieniu jako *flow state* z pozytywnej psychologii. Aby lepiej zrozumieć zainteresowanie wokół tego edytora został przytoczony fragment wypowiedzi Manuel Lopez z serwisu Quora [15]: *I don't need to translate my thought “clear the contents of the parentheses” to “move my hand, move the mouse, keep the left button pressed, move to the other end, release the the left button, press the right button, locate the 'cut' command”. All I need to type is di( and I'm done, and I don't even care how many lines are between those parentheses (50 lines? 150? It's the exact same 3 key presses!).*

IDE nie powinno rozpraszać użytkownika, dlatego zdecydowano się na prosty, niemalże konsolowy wygląd. Zbyt krzykliwy wygląd, czy też notorycznie wyskakujące okienka dekoncentrują odbiorcę, przez co ten jest mniej wydajny. Minimalistyczny wygląd pozwolił na wypełnienie większej przestrzeni roboczej edytorem kodu.

Aplikacja wspiera akcje typu: podpowiadanie, podświetlanie składni, skakanie do definicji, *refaktoring*, czy lista referencji dla wszystkich języków, do których jest zainstalowana wtyczka. Dla większej elastyczności rozszerzenia są przypisane dla konkretnego projektu oraz dla użytkownika. Przełączając się między projektami zmienia się lista wtyczek. Wspomniane wyżej akcje działają nawet wówczas, gdy w projekcie znajdują się pliki od różnych technologii, pod warunkiem że zostały zainstalowane odpowiednie wtyczki.

Narzędzie działa bez żadnych opóźnień na wolnym łączu internetowym oraz na urządzeniach nieprzystosowanych do tworzenia oprogramowania, na przykład tabletach. Zrezygnowano z widoku dla telefonów komórkowych, ponieważ programowanie na telefonie uznano jedynie za ciekawostkę.

W systemie istnieje wbudowane konto admina ze znanym loginem i hasłem tylko dla osoby, która instalowała środowisko. Dopiero poprzez konto admina można stworzyć konta dla deweloperów. Zrezygnowano z typowego modelu SaaS (*Software as a Service*) na potrzeby prototypu, stąd brak możliwości rejestracji nowego użytkownika oraz brak mechanik subskrypcji. Prototyp został przygotowany pod instalację na własnych serwerach. Każdy użytkownik może zakładać nowe projekty i przypisywać wtyczki do nich. Użytkownik może przypisywać do swojego projektu innych użytkowników, aby kooperować. Tylko twórca projektu może usunąć i zmienić nazwę projektu. Tylko twórca projektu może usunąć i zmienić nazwę projektu.

Aby aplikacja wspierała jak najwięcej języków programowania został użyty protokół serwera języka (LSP – *Language Server Protocol*) [16]. Odpowiednio skonfigurowane LSP za pomocą wtyczek daje wsparcie dla nieograniczonej liczby języków.

Wtyczki mają dostęp do wszystkich warstw systemu, dzięki czemu dostarczane rozszerzenie może wpływać na: interfejs użytkownika, logikę biznesową, czy nawet środowisko, na którym znajduje się IDE.

## 2.2. Model koncepcji środowiska

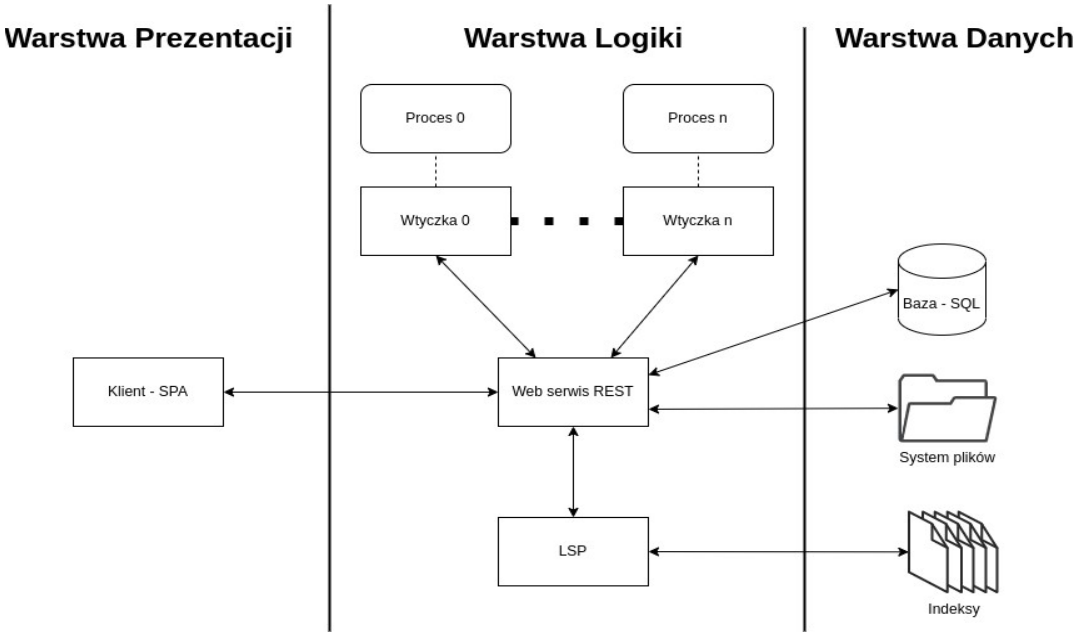
Na potrzeby projektu została wybrana trójwarstwowa architektura systemu z kilku powodów: łatwego serwisu, niskiego kosztu budowy, łatwej migracji, szybkiej konfiguracji, możliwości łatwego dodawania nowych podsystemów, jak np. dodatkowy *Web Serwis*, czy procesy automatyczne. Trzy oddzielne niezależne warstwy umożliwiają podmianę jednej z nich bez ingerencji w pozostałe.

Rysunek 10. przedstawia model koncepcji IDE w chmurze z fizycznie rozmieszczonymi podsystemami, prezentowana tutaj architektura posiada trzy niezależne warstwy: prezentacji, logiki oraz danych.

Warstwa prezentacji zawiera klienta jako webową jednostronicową aplikację działającą w przeglądarce.

Warstwa logiki zawiera *Web Serwis* jako wielowątkową aplikację rozbudowywaną o wtyczki. Każda z zainstalowanych wtyczek może mieć pod sobą uruchomiany proces, gdzie wtyczki nie współpracują ze sobą. Po odpowiednim skonfigurowaniu serwera, uruchamia on serwer wybranego języka.

Warstwa danych mieści w sobie relacyjną bazę danych, system plików oraz indeksy wygenerowane przez serwer języka. Żaden z wymienionych elementów nie współpracuje bezpośrednio ze sobą.



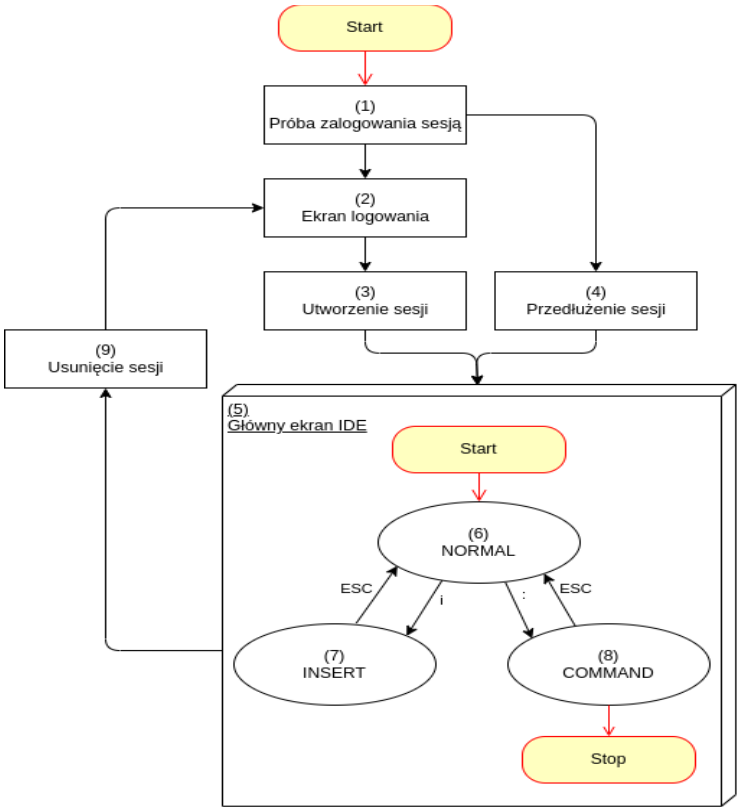

Rysunek 10. Model koncepcji IDE

## 2.3. Poruszanie się po IDE

Na załączonym Rysunku 11. został przedstawiony schemat przemieszczania się w prototypie. Po załadowaniu strony z narzędziem w przypadku braku sesji ukaże się ekran logowania (2). Jeśli sesja istnieje i nie wygasła to pierwszym ekranem, jaki się pojawi jest ekran główny (5). W aplikacji istnieją trzy tryby pracy, a każdy z trybów ma swoje przeznaczenie:

- NORMAL – pozwala na przechodzenie w inne tryby (7 i 8), przemieszczanie się po zawartości pliku oraz udostępnia skróty klawiszowe. Wejście w ten tryb umożliwia klawisz ESC. Podczas wchodzenia w ten tryb następuje synchronizacja zawartości uruchomionego pliku w edytorze z odpowiednikiem po stronie serwera;
- INSERT – umożliwia przemieszczanie się po zawartości pliku i swobodną edycję, tak jak w przypadku większości edytorów tekstu czy IDE. Tryb ten jest uruchamiany przez klawisz „i”;
- COMMAND – tryb do wprowadzania i wybierania ostatnio używanych komend, klawisz „:” odpowiada za włączenie tego trybu. Komenda *logout* powoduje usunięcie sesji i przełączenie na ekran logowania (2).

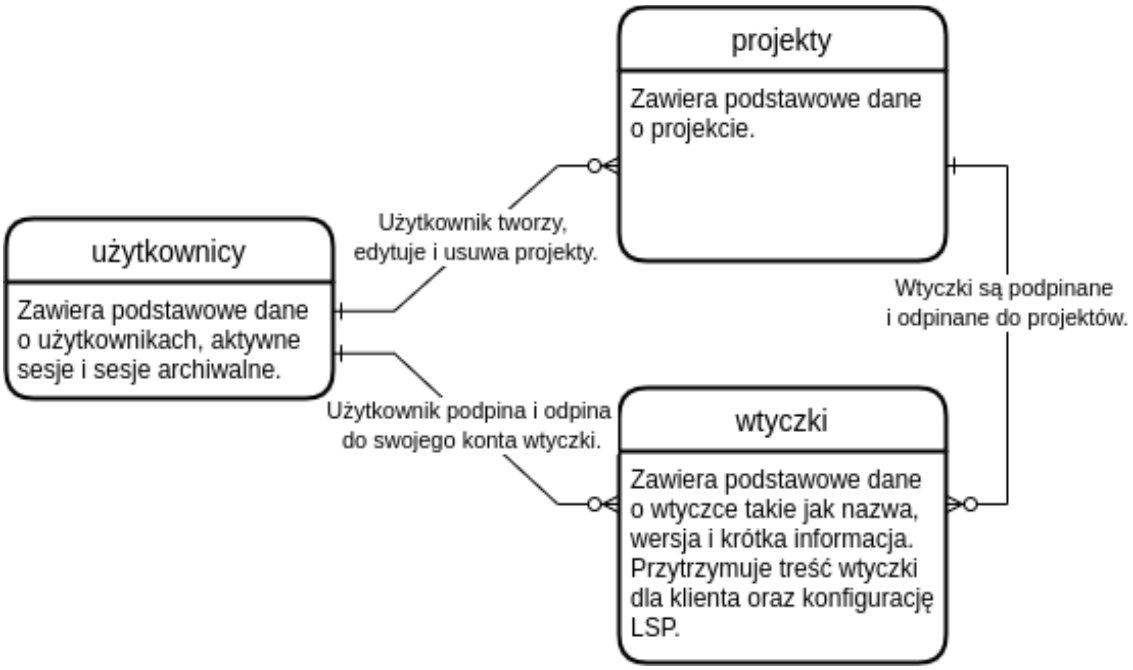
Tryby pracy zostały zaimplementowane na wzór wybranych trybów z Vima. Dzięki tej cesze można wykorzystywać komendy do edycji pliku, dla przykładu wystarczy wprowadzić komendę „di<” gdy karetka jest pomiędzy znakami „<” oraz „>” aby usunąć całą zawartość pomiędzy nimi. Użytkownicy Vima chwalą tryby za mnogość możliwości przy zaskakująco małej liczbie naciśnięć klawiszy [17]. Warto też wspomnieć o komendzie „.”, która wykonuje ostatnio użytą komendę.




Rysunek 11. Diagram poruszania się po IDE

## 2.4. Ideowy model danych – diagram ERD

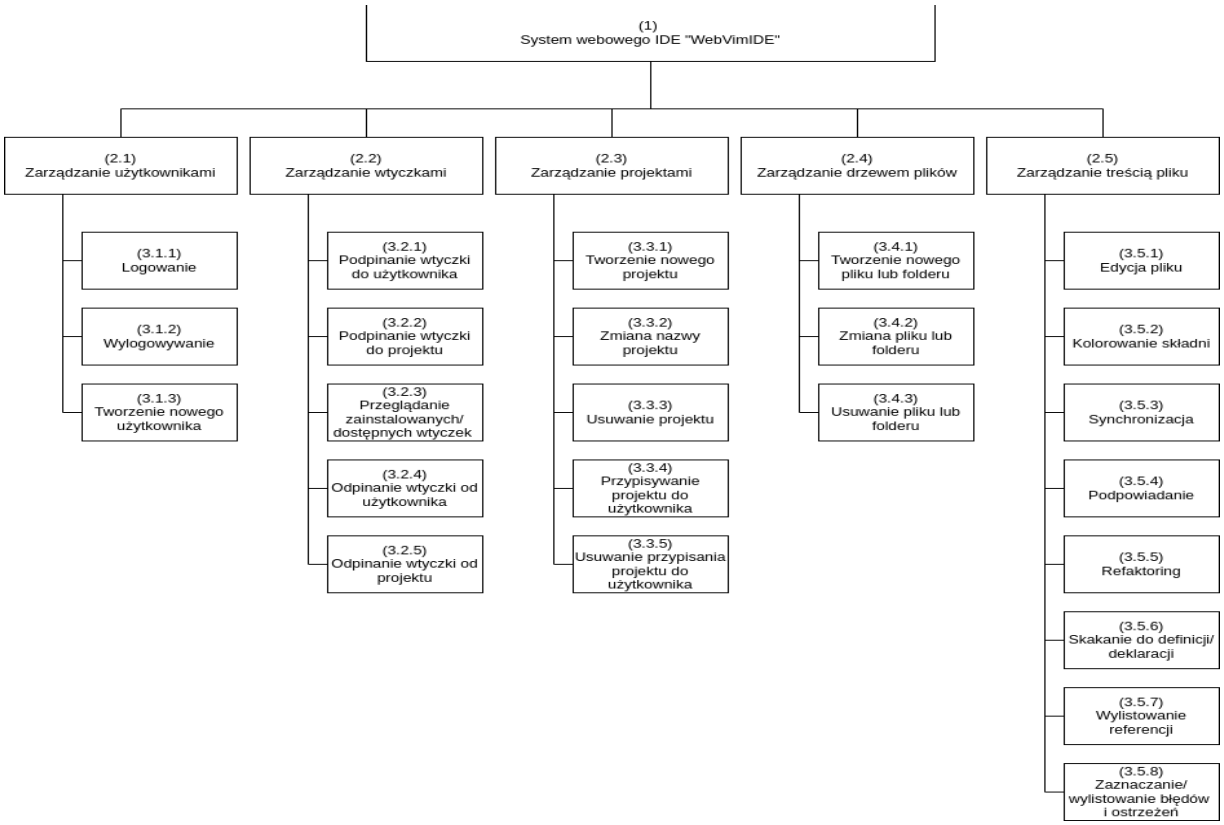
Rysunek 12. przedstawia ideowy diagram związków encji (ERD – *Entity–Relationship Diagram*), który zawiera encje i relacje między nimi zachodzące. Każda relacja ma jednoznaczną nazwę, a opis encji znajduje się pod jej nazwą. Na potrzeby projektu została wybrana notacja kurzych łapek.




Rysunek 12. Ideowy diagram związków encji

## 2.5. Specyfikacja funkcjonalna

Rysunek 13. to diagram hierarchii funkcji (FHD – *Function Hierarchy Diagram*) przedstawiający hierarchię funkcji poprzez zastosowanie dekompozycji głównych funkcjonalności, na potrzeby pracy dokonano dekompozycji na poziomie drugim. Cyfry w nawiasach wskazują na identyfikator, umożliwiający łatwiejszą nawigację w Tabeli 1., zawierającej opisy funkcji.




Rysunek 13. Diagram hierarchii funkcji

Opisy poszczególnych funkcji przedstawionych w Rysunku 13. zostały zawarte w Tabeli 1.

**Tabela 1. Opis funkcji z diagramu hierarchii funkcji**

| **Id** | **Nazwa** | **Opis** |
|---|---|---|
| 1 | System webowego IDE „WebVimIDE”. | IDE pozwala na edycję całych projektów łącznie z drzewem plików oraz samych plików, przy tym udostępnia wiele funkcji wspomagania pisania kodu. |
| 2.1 | Zarządzanie użytkownikami. | System pozwala na tworzenie nowego użytkownika, logowanie i wylogowywanie. |
| 2.2 | Zarządzanie wtyczkami. | Użytkownik może przyłączać i odłączać wtyczki do projektu oraz do swojego konta. |
| 2.3 | Zarządzanie projektami. | Deweloper może tworzyć, usuwać i zmieniać nazwę swoich projektów. Może również przypisywać swoje projekty innym użytkownikom, które może usunąć. |
| 2.4 | Zarządzanie drzewem plików. | Użytkownik może tworzyć, usuwać i zmieniać nazwy folderom i plikom w drzewie projektu. |
| 2.5 | Zarządzanie treścią pliku. | Użytkownik może edytować treść pliku a przy tym wykonywać i wprowadzać komendy do refaktoryzacji, wylistowania referencji, czy do skoczenia do deklaracji. System synchronizuje treść pliku, sprawdza plik pod względem błędów i ostrzeżeń. Dodatkowo jeśli system |

| | | |
|---|---|---|
| | | wykryje odpowiednią akcję uruchamia podpowiadanie. |
| 3.1.1 | Logowanie. | Użytkownik posiadający konto może się zalogować. |
| 3.1.2 | Wylogowywanie. | Zalogowany deweloper może się wylogować. |
| 3.1.3 | Tworzenie nowego użytkownika. | Nowy użytkownik może zostać utworzony dzięki innemu użytkownikowi. |
| 3.2.1 | Podpinanie wtyczki do użytkownika. | Użytkownik może podpiąć wtyczkę do swojego konta, dzięki czemu wtyczka będzie działała w obrębie konta. |
| 3.2.2 | Podpinanie wtyczki do projektu. | Właściciel projektu może podpiąć wtyczkę do projektu, dzięki czemu wtyczka będzie działała tylko w obrębie projektu. |
| 3.2.3 | Przeglądanie zainstalowanych/ dostępnych wtyczek. | Po wydaniu odpowiedniej komendy użytkownik może podejrzeć, jakie aktualnie wtyczki są używane. |
| 3.2.4 | Odpinanie wtyczki od użytkownika. | Użytkownik może odpiąć wtyczkę od swojego konta, tak aby nie była już używana. |
| 3.2.5 | Odpinanie wtyczki od projektu. | Właściciel projektu może odpiąć wtyczkę od projektu, żeby nie była już używana w obrębie projektu. |
| 3.3.1 | Tworzenie nowego projektu. | Użytkownik może utworzyć projekt o wskazanej nazwie. Nazwa musi być unikalna w obrębie systemu. |
| 3.3.2 | Zmiana nazwy projektu. | Właściciel projektu może zmienić nazwę projektu. |
| 3.3.3 | Usuwanie projektu. | Właściciel projektu może usunąć projekt. |
| 3.3.4 | Przypisywanie projektu do użytkownika. | Właściciel projektu może udzielić dostęp innemu użytkownikowi. |
| 3.3.5 | Usuwanie przypisania projektu do użytkownika. | Właściciel projektu może usunąć dostęp innemu użytkownikowi. |
| 3.4.1 | Tworzenie nowego pliku lub folderu. | Użytkownik może w drzewie projektu dodawać pliki i foldery. |
| 3.4.2 | Zmiana nazwy pliku lub folderu. | Użytkownik może w drzewie projektu zmieniać nazwy pliku lub folderu. Zmiany zostaną naniesione w systemie plików. |
| 3.4.3 | Usuwanie pliku lub folderu. | Użytkownik może usuwać pliki i foldery z drzewa projektu. |
| 3.5.1 | Edycja pliku. | Deweloper może w trybie INSERT edytować dowolnie plik. |
| 3.5.2 | Kolorowanie składni. | Edytor pozwala na kolorowanie składni wspieranych języków. |
| 3.5.3 | Synchronizacja. | System synchronizuje otwarty plik z plikiem w systemie plików. |
| 3.5.4 | Podpowiadanie. | Edytor podpowiada składnię oraz inne elementy, jak np.: ścieżki plików, czy pola i metody w obiektach. |
| 3.5.5 | *Refaktoring.* | IDE po wpisaniu odpowiedniej komendy z nową nazwą potrafi zamienić nazwę wskazanego symbolu. |
| 3.5.6 | Skakanie do definicji/ deklaracji. | Środowisko potrafi załadować do edycji plik, w którym |

| | | |
|---|---|---|
| | | znajduje się definicja lub deklaracja wskazanego symbolu. |
| 3.5.7 | Wylistowanie referencji. | Po wpisaniu komendy system wyświetla listę referencji wskazanego symbolu. Każdy element listy można uruchomić do edycji w oknie edytora. |
| 3.5.8 | Zaznaczanie/ wylistowanie błędów i ostrzeżeń. | IDE po synchronizacji pliku sprawdza plik w poszukiwaniu błędów i ostrzeżeń, gdy znajdzie coś niepokojącego, wówczas zaznacza fragment kodu, stawiając ikonkę na marginesie linii z błędem oraz pokazuje listę wszystkich znalezionych problemów z ich opisem. |

## 2.6. Specyfikacja pozafunkcjonalna

Wymagania pozafunkcjonalne przedstawia Tabela 2.

**Tabela 2. Opis wymagań pozafunkcjonalnych**

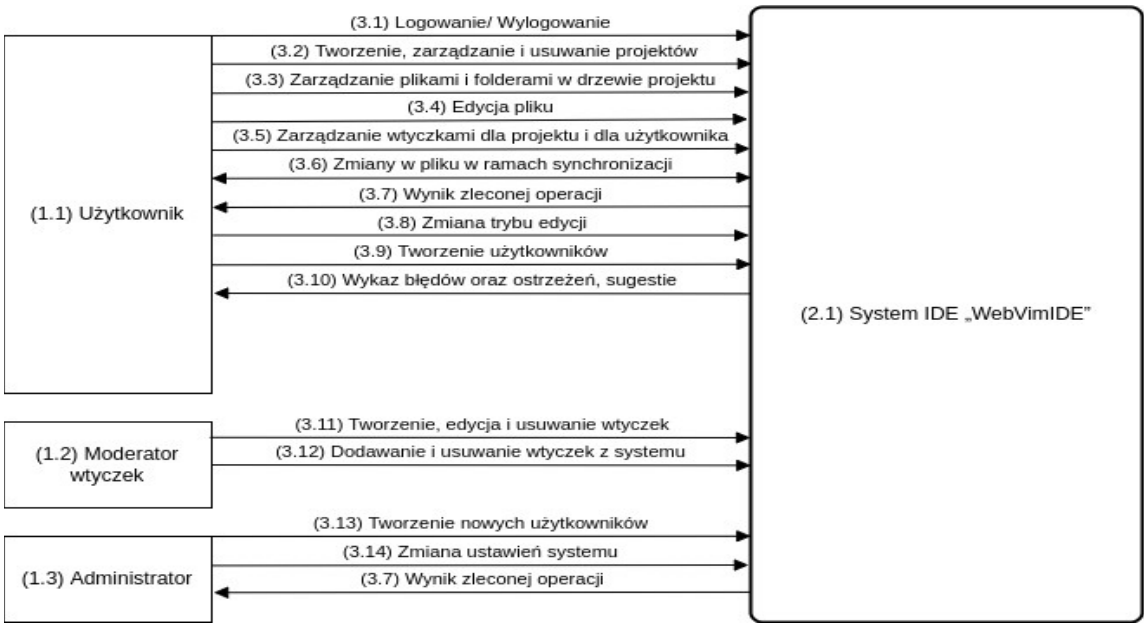
| **Wymaganie pozafunkcjonalne** | **Opis** |
|---|---|
| Objętość systemu. | • System może być postawiony na kilku hostach. |
| Wydajność. | • Serwer powinien obsłużyć 32 połączeń jednocześnie, bez straty na wydajności. |
| Media komunikacyjne. | • Łącze internetowe – min. 10Gbit/s. |
| Bezpieczeństwo. | • Zasoby sieciowe z ograniczonym dostępem osób niepowołanych.<br>• *Backup* bazy danych lokalnie, bez możliwości połączenia przez sieć z zewnątrz. |
| Dostępność. | • System musi funkcjonować w pełnym wymiarze czasu.<br>• Prace konserwacyjne muszą odbywać się w czasie do tego specjalnie wyznaczonym. |
| Skalowalność. | • System musi posiadać możliwość rozbudowy w przypadku zwiększonego ruchu na serwerze.<br>• System musi posiadać możliwość rozbudowy o kolejne podsystemy. |
| *Backup*. | • Do procesu *backup* należy poza bazą danych dodać również system plików z projektami. |

## 2.7. Diagramy DFD

W podrozdziale tym zostały przedstawione diagramy przepływu danych (DFD – *Data Flow Diagram*) przedstawiające w jaki sposób dane przepływają w systemie oraz opisują procesy przetwarzające dane. Omawiane diagramy to kolejno: diagram kontekstowy oraz diagram systemowy. Każdy z nich został przedstawiony i opisany w osobnych paragrafach. Na potrzeby pracy została wybrana notacja Gane–Sarson'a.

### 2.7.1 Diagram kontekstowy

W Rysunku 14. został zaprezentowany diagram kontekstowy DFD który definiuje kontekst i granice działania systemu. W nawiasach okrągłych znajduje się id elementu w celu łatwiejszej identyfikacji.




Rysunek 14. Diagram kontekstowy przepływu danych

Wszystkie elementy diagramu kontekstowego z Rysunku 14. zostały opisane w Tabeli 3. za pomocą następujących kolumn: id, nazwa, typ oraz opis.

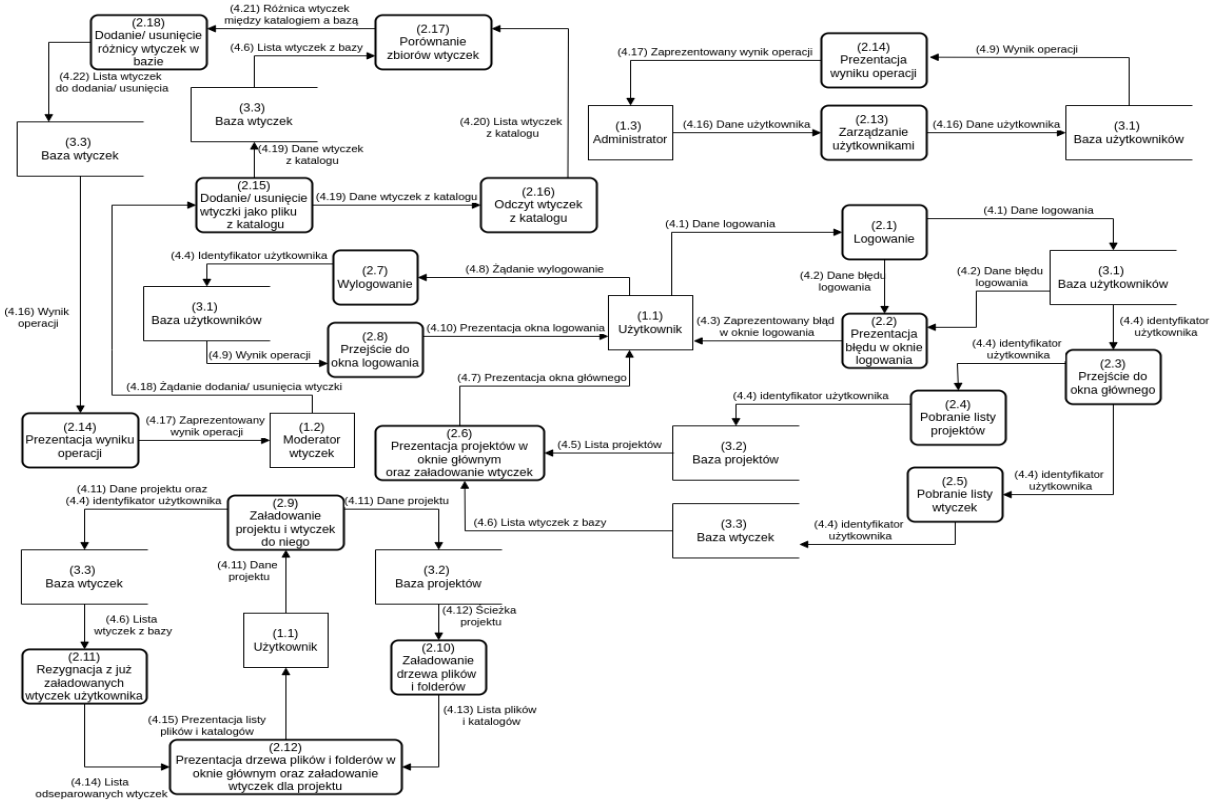
**Tabela 3: Opis kontekstowego diagramu przepływu danych**

| Id | Nazwa | Typ | Opis |
|---|---|---|---|
| 1.1 | Użytkownik | Terminator | Osoba fizyczna korzystająca z funkcjonalności do tworzenia oprogramowania. |
| 1.2 | Moderator wtyczek | Terminator | Osoba fizyczna tworząca wtyczki. |
| 1.3 | Administrator | Terminator | Osoba fizyczna zarządzająca całym systemem. |
| 2.1 | System IDE „WebVimIDE". | Proces | Główny proces systemu zawierający kilka aplikacji, które współpracują ze sobą. |
| 3.1 | Logowanie/ Wylogowanie. | Przepływ danych | Dane użytkownika tj.: login, hasło oraz sesja. |
| 3.2 | Tworzenie, zarządzanie | Przepływ | Dane projektu jak nazwa i identyfikator. |

| | | | |
|---|---|---|---|
| | i usuwanie projektów. | danych | |
| 3.3 | Zarządzanie plikami i folderami w drzewie projektu. | Przepływ danych | Daną jest tutaj ścieżka dostępu do pliku lub folderu. |
| 3.4 | Edycja pliku. | Przepływ danych | Niesie dane jak: ścieżka do edytowanego pliku oraz listę zmian. |
| 3.5 | Zarządzanie wtyczkami dla projektu i dla użytkownika. | Przepływ danych | Danymi są tutaj nazwa wtyczki oraz nazwa projektu lub identyfikator użytkownika. |
| 3.6 | Zmiany w pliku w ramach synchronizacji. | Przepływ danych | Jako dane zawiera listę zmian do wprowadzenia. |
| 3.7 | Wynik zleconej operacji. | Przepływ danych | Wynik reprezentowany jest przez trzy składowe: czy komunikat pozytywny, kod oraz wiadomość. |
| 3.8 | Zmiana trybu edycji. | Przepływ danych | Daną jest tutaj nazwa trybu edycji na podstawie której są blokowane/ odblokowywane funkcje edytora. |
| 3.9 | Tworzenie użytkowników. | Przepływ danych | Nazwa użytkownika, login i hasło są tutaj danymi. |
| 3.10 | Wykaz błędów oraz ostrzeżeń, sugestie. | Przepływ danych | Danymi są tutaj typ, treść oraz pozycja w pliku. |
| 3.11 | Tworzenie, edycja i usuwanie wtyczek. | Przepływ danych | Zmiany w plikach *.so lub *.dll. |
| 3.12 | Dodawanie i usuwanie wtyczek z systemu. | Przepływ danych | Zmiana lokalizacji fizycznego pliku. |
| 3.13 | Tworzenie nowych użytkowników. | Przepływ danych | Nazwa użytkownika, login i hasło są tutaj danymi. |
| 3.14 | Zmiana ustawień systemu. | Przepływ danych | Danymi są tutaj nazwa ustawienia oraz jej wartość. |

### 2.7.2 Diagram systemowy

W Rysunku 15. został zaprezentowany diagram systemowy DFD który ukazujący główne funkcje systemu. W nawiasach okrągłych znajduje się id elementu w celu łatwiejszej identyfikacji.




Rysunek 15. Diagram systemowy przepływu danych

Wszystkie elementy diagramu systemowego z Rysunku 15. zostały opisane w Tabeli 4. za pomocą następujących kolumn: id, nazwa, typ oraz opis.

**Tabela 4: Opis systemowego diagramu przepływu danych**

| Id | Nazwa | Typ | Opis |
|---|---|---|---|
| 1.1 | Użytkownik | Terminator | Osoba fizyczna korzystająca z funkcjonalności do tworzenia oprogramowania. |
| 1.2 | Moderator wtyczek | Terminator | Osoba fizyczna tworząca wtyczki. |
| 1.3 | Administrator | Terminator | Osoba fizyczna zarządzająca całym systemem. |
| 2.1 | Logowanie. | Proces | Sprawdza czy istnieje użytkownik o podanym loginie i czy podane hasło jest przypisane logującego się. |
| 2.2 | Prezentacja błędu w oknie logowania. | Proces | Wyświetla w oknie logowania błąd dotyczący błędu podczas logowania. |
| 2.3 | Przejście do okna głównego. | Proces | Ładuje główne okno i wyświetla go w przeglądarce zamiast okna logowania. |
| 2.4 | Pobranie listy projektów. | Proces | Na podstawie identyfikatora użytkownika pobierane są projekty przypisane do niego. |

| | | | |
|---|---|---|---|
| 2.5 | Pobranie listy wtyczek. | Proces | Na podstawie identyfikatora użytkownika pobierane są wtyczki przypisane do niego. |
| 2.6 | Prezentacja projektów w oknie głównym oraz załadowanie wtyczek. | Proces | Wyświetla w oknie głównym w miejscu do tego przeznaczonym listę projektów do wybrania. Wykonywany jest również wstrzyknięcie wtyczek do klienta. |
| 2.7 | Wylogowanie. | Proces | Proces usuwa dane użytkownika z pamięci następnie, usuwa sesję. |
| 2.8 | Przejście do okna logowania. | Proces | Ładuje okno logowania i wyświetla go w przeglądarce zamiast okna głównego. |
| 2.9 | Załadowanie projektu i wtyczek do niego. | Proces | Zostaje załadowany projekt z bazy następnie, wysyłane jest żądanie do pobrania wtyczek z bazy dla niego. |
| 2.10 | Załadowanie drzewa plików i folderów. | Proces | Odczytanie całego drzewa plików i folderów z systemu pliku na postawie ścieżki. |
| 2.11 | Rezygnacja z już załadowanych wtyczek użytkownika. | Proces | Z listy wtyczek dla użytkownika i dla wybranego projektu eliminowane są te, które są przypisane do użytkownika gdyż on są już załadowane. |
| 2.12 | Prezentacja drzewa plików i folderów w oknie głównym oraz załadowanie wtyczek dla projektu. | Proces | Wyświetla w oknie głównym w miejscu do tego przeznaczonym drzewo plików i folderów. Wykonywany jest również wstrzyknięcie wtyczek do klienta. |
| 2.13 | Zarządzanie użytkownikami. | Proces | Dodanie, usuwanie i modyfikowanie użytkownika. |
| 2.14 | Prezentacja wyniku operacji. | Proces | Wyświetla komunikat o stanie wykonania operacji. |
| 2.15 | Dodanie/ usunięcie wtyczki jako pliku z katalogu. | Proces | Dodaje lub usuwa wtyczkę jako plik biblioteki dynamicznej z katalogu. |
| 2.16 | Odczyt wtyczek z katalogu. | Proces | Ładuje wszystkie plik biblioteki dynamicznej z katalogu a następnie odczytuje ich zawartość. Pliki o niepoprawnej strukturze są pomijane. |
| 2.17 | Porównanie zbiorów wtyczek. | Proces | Porównuje dwa zbiory wtyczek, wczytane z katalogi i załadowane z bazy. Na podstawie nich tworzy dwa zbiory, wtyczki do usunięcia i do dodania. |
| 2.18 | Dodanie/ usunięcie różnicy wtyczek w bazie. | Proces | Dodaje lub usuwa wtyczki z bazy na podstawie otrzymanych zbiorów. |
| 2.19 | Reprezentacja wyniku dodania/ usunięcia wtyczek. | Proces | Wyświetla komunikat zawierający aktualne wtyczki. |

| | | | |
|---|---|---|---|
| 3.1 | Baza użytkowników | Magazyn danych | Baza danych, przechowuje informacje o użytkowniku. |
| 3.2 | Baza projektów | Magazyn danych | Baza danych, zawiera informacje o projektach. |
| 3.3 | Baza wtyczek | Magazyn danych | Baza danych, mieści w sobie informacje o wtyczkach. |
| 4.1 | Dane logowania. | Przepływ danych | Dane użytkownika tj.: login, hasło oraz sesja. |
| 4.2 | Dane błędu logowania. | Przepływ danych | Dana błędu zawiera w sobie: czy komunikat pozytywny, kod oraz wiadomość. |
| 4.3 | Zaprezentowany błąd w oknie logowania. | Przepływ danych | Dana tekstowa reprezentująca kawałek HTML z rysowaniem tekstu o wybranym kolorze. |
| 4.4 | Identyfikator użytkownika. | Przepływ danych | Identyfikator użytkownika jako ciąg liczb i liter. |
| 4.5 | Lista projektów. | Przepływ danych | Lista par składających się z id oraz nazwy reprezentujące projekt. |
| 4.6 | Lista wtyczek z bazy. | Przepływ danych | Lista pochodząca z bazy danych, zawiera elementy opisujące wtyczkę tj.: nazwa, wersja, opis oraz treść wtyczki dla klienta. |
| 4.7 | Prezentacja okna głównego. | Przepływ danych | Dana zawiera HTML opisujący okno główne. |
| 4.8 | Żądanie wylogowanie. | Przepływ danych | Wyemitowany sygnał zlecający wylogowanie. |
| 4.9 | Wynik operacji. | Przepływ danych | Dana reprezentuje czy komunikat jest pozytywny, kod oraz wiadomość. |
| 4.10 | Prezentacja okna logowania. | Przepływ danych | Dana zawiera HTML opisujący okno logowania. |
| 4.11 | Dane projektu. | Przepływ danych | Jest to para składająca się z id oraz nazwy. |
| 4.12 | Ścieżka projektu. | Przepływ danych | Dana tekstowa zawierająca ścieżkę dostępu do określonego pliku. |
| 4.13 | Lista plików i katalogów. | Przepływ danych | Lista ścieżek reprezentująca pliki i katalogi. |
| 4.14 | Lista odseparowanych wtyczek. | Przepływ danych | Wynik porównania wtyczek użytkownika i wtyczek projektu. |
| 4.15 | Prezentacja listy plików i katalogów. | Przepływ danych | HTML przedstawiający drzewo plików i katalogów. |
| 4.16 | Dane użytkownika. | Przepływ danych | Wartości tj.: nazwa użytkownika, login i hasło. |
| 4.17 | Zaprezentowany wynik operacji. | Przepływ danych | Dana tekstowa reprezentująca kawałek HTML z rysowaniem tekstu o wybranym kolorze. |
| 4.18 | Żądanie dodania/ usunięcia wtyczki. | Przepływ danych | Wyemitowany sygnał do sprawdzenia zmian w katalogu z wtyczkami. |
| 4.19 | Dane wtyczek z katalogu. | Przepływ danych | Zawiera elementy opisujące wtyczkę tj.: nazwa, wersja, opis oraz treść wtyczki dla klienta pochodzące z katalogu. |

| | | | |
|---|---|---|---|
| 4.20 | Lista wtyczek z katalogu. | Przepływ danych | Lista pochodząca z katalogu, zawiera elementy opisujące wtyczkę tj.: nazwa, wersja, opis oraz treść wtyczki dla klienta. |
| 4.21 | Różnica wtyczek między katalogiem a bazą. | Przepływ danych | Dana zawiera dwa zbiory opisujące wtyczki: do dodania i do usunięcia. |
| 4.22 | Lista wtyczek do dodania/ usunięcia. | Przepływ danych | Lista wtyczek przygotowana pod operacje dodawania/ usuwania w bazie. |

## 2.8. Wady i zalety zaproponowanego rozwiązania

Po przeanalizowaniu zaproponowanego projektu okazało się że jest w nim więcej zalet niż wad, wskazuje to na to że projekt ma prawo bytu i może przyjąć się na rynku. Zalety i wady zostały zaprezentowane w Tabeli 5.

**Tabela 5. Zalety i wady omawianego rozwiązania**

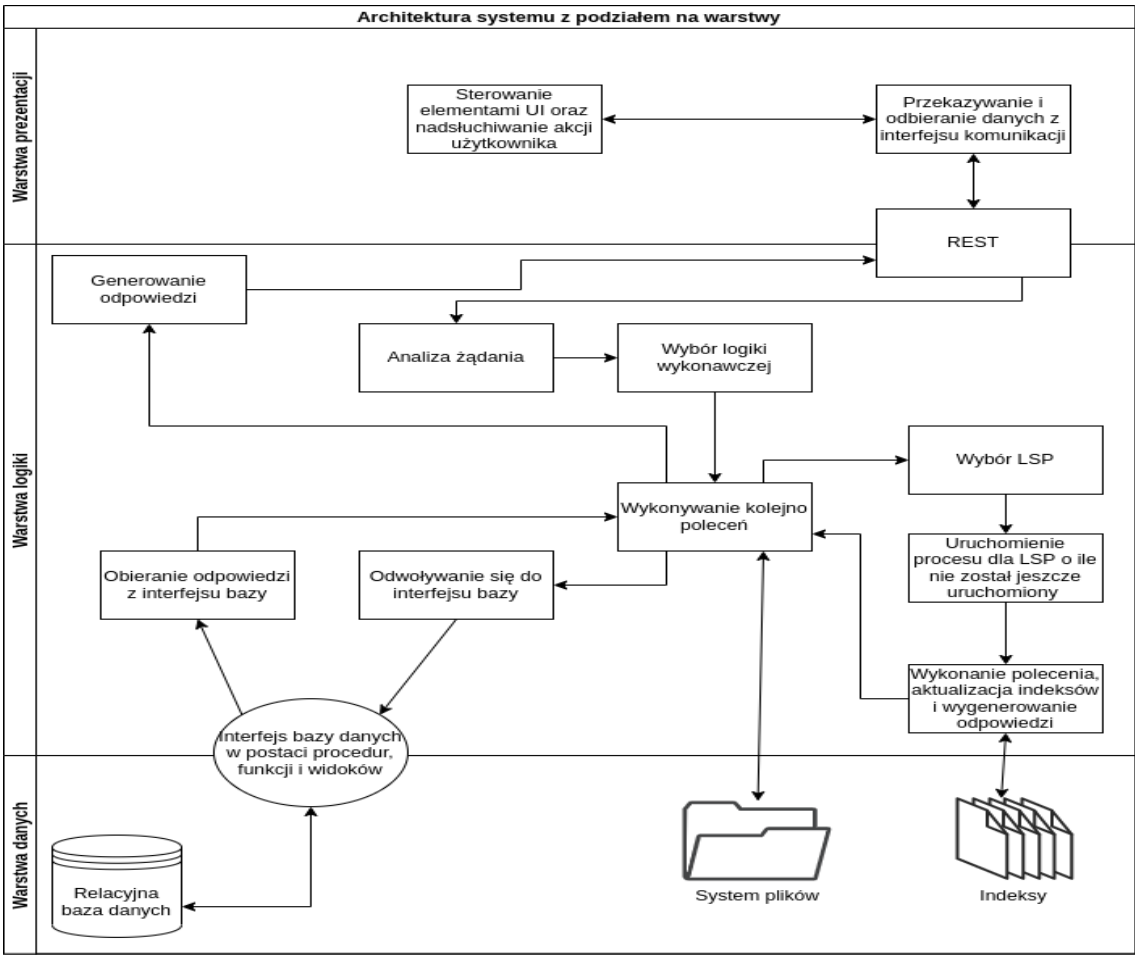
| **Zalety** | **Wady** |
|---|---|
| Prosty, nierozpraszający wygląd. | Przymus do nauki narzędzia. |
| Duża przestrzeń robocza. | IDE może trafić tylko do niszy, jaką są użytkownicy Vima. |
| Szybka synchronizacja plików i co za tym idzie także szybkie podpowiadanie nawet dla dużych plików. | Konkurencja zapewnia więcej funkcji. |
| Możliwa obsługa tylko samą klawiaturą. | Tylko jedno środowisko. |
| Łatwe i intuicyjne skróty klawiszowe oraz komendy. | Załadowanie dużego pliku do edycji trwa dłużej niż w standardowym IDE. |
| Łatwa personalizacja za pomocą wtyczek. | |
| Swobodna rozbudowa narzędzia o kolejne funkcje, dzięki wtyczkom. | |
| Zapewniona podstawowa funkcjonalność znana z innych IDE. | |
| Możliwa praca grupowa. | |
| Praca po nauczeniu się środowiska staje się szybsza. | |
| Wielu programistów, jedno środowisko. | |
| Rozbudowany i rozszerzalny *help.* | |

# 3. Opracowanie projektu

W tym rozdziale zostały przedstawione projekty elementów systemu, rozdział otwiera projekt architektury. W dalszej kolejności został szeroko opisany projekt bazy danych, Gdzie opis projektu interfejsu użytkownika znajduje się na końcu tego rozdziału.

## 3.1. Projekt architektury systemu

Zamieszczony Rysunek 16. prezentuje diagram warstw architektury, przedstawiając poszczególne warstwy systemu oraz komunikacje między nimi zachodzące.




Rysunek 16. Diagram warstw architektury z komunikacją

## 3.2. Projekt bazy danych

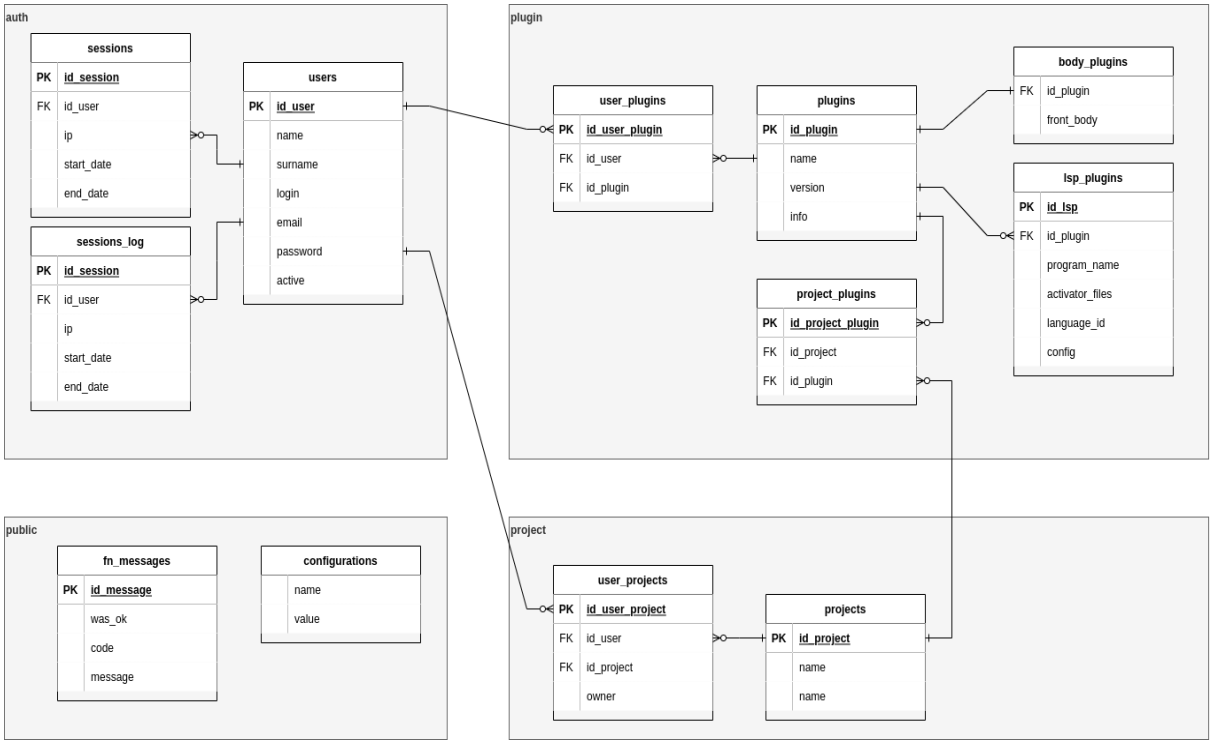
Baza danych jest podstawą warstwy danych oraz jest najbardziej wrażliwą częścią systemu, dlatego też w niniejszej pracy została ona przedstawiona w obszerny sposób.

### 3.2.1 Opis bazy danych

Projekt bazy danych został stworzony w oparciu o relacyjną bazę danych. Jako system zarządzania bazą został wybrany PostgreSQL w wersji 12.3, zaś jako narzędzie do administrowania bazą został wybrany psql.exe, jako aplikacja konsolowa. Wybór takiego, a nie innego systemu dokonano na podstawie własnych doświadczeń z różnymi systemami oraz szerzącą się popularnością Postgresa. Baza danych otrzymała nazwę „db_ide” oraz kodowanie UTF–8, przy projektowaniu bazy zostały wykorzystane schematy.

Schemat (SQL *schema*) jest to mechanizm relacyjnych baz danych pozwalający na tworzenie logicznych ugrupowań obiektów tj.: tabele, funkcje itp. Koncepcja ta została użyta w celu łatwiejszej nawigacji po bazie oraz zachowania porządku w niej.

Rysunek 17. to diagram przedstawiający tabele, połączenia zachodzące między relacjami, pola i ich podstawowe atrybuty oraz schematy. Na potrzeby pracy została wybrana notacja Jamesa Martina, potocznie zwana jako „notacja kurzych łapek”. Jako oznaczenie klucza głównego został przyjęty skrót PK (od ang. *Primary Key*). Klucz obcy otrzymał skrót FK (od ang. *Foreign Key*). Schematy zostały oznaczone jako szare prostokąty okalające tabele.




Rysunek 17. Fizyczny model bazy danych

Baza została podzielona na cztery schematy, gdzie każdy schemat odpowiada innej dziedzinie. Występujące schematy to kolejno:

- `auth` – zawiera tabele opisujące użytkownika tj.: `users`, `sessions` czy `sessions_log`;

- `plugin` – schemat opisuje wtyczki oraz LSP, zawiera tabele: `plugins`, `body_plugins`, `user_plugins`, `project_plugins` oraz `lsp_plugins`;
- `project` – opisuje projekty, zawiera tabele: `projects` oraz `user_projects`;
- `public` – domyślny schemat, w skład którego wchodzą wolne tabele: `fn_messages` oraz `configurations`.

### 3.2.2 Opisy tabel i pól

Tabela 6. ukazuje opisy poszczególnych relacji z uwzględnieniem schematów, które ukazano w diagramie zamieszczonym jako Rysunek 17.

**Tabela 6. Opisy poszczególnych relacji**

| Schemat | Nazwa relacji | Opis |
|---|---|---|
| `auth` | `users` | Przechowuje podstawowe dane użytkownika tj.: imię, nazwisko, login czy hasło. Posiada informację czy użytkownik jest aktywny tzn. czy może się zalogować. |
| | `sessions` | Sesje zalogowanych użytkowników są trzymane w tej tabeli. Na jej podstawie system wie, czy użytkownika można wpuścić od razu do IDE, czy trzeba identyfikować loginem i hasłem. |
| | `sessions_log` | Przechowuje archiwalne sesje. |
| `plugin` | `plugins` | Zawiera informacje o wtyczce tj.: nazwa, wersja czy opis. |
| | `body_plugins` | Do wtyczki dołączona jest wtyczka dla interfejsu użytkownika, która znajduje się w tej tabeli. |
| | `lsp_plugins` | Przechowuje konfiguracje dla LSP. |
| | `user_plugins` | Tabela rozwiązuje połączenie wiele do wielu relacji `users` oraz `plugins`. Dodatkowo przechowuje informację czy użytkownik jest właścicielem projektu. |
| | `project_plugins` | Tabela rozwiązuje połączenie wiele do wielu relacji `projects` oraz `plugins`. |
| `project` | `projects` | Słownik zawierający tylko identyfikator i nazwę projektu. |
| | `user_projects` | Tabela rozwiązuje połączenie wiele do wielu relacji `users` oraz `projects`. |
| `public` | `fn_messages` | Słownik poszerzony o kolumny *code* oraz *was_ok*. Zawiera komunikaty systemowe. |
| | `configurations` | Tabela typu *key-value* przechowuje konfigurację systemu. |

Szczegóły tabel i pól widocznych na diagramie jako Rysunek 18. zostały opisane w poniższych tabelach od 7. do 18. Jako pierwsza relacja szczegółowo opisana została `users` w Tabeli 7.

**Tabela 7. Szczegóły pól zawarte w relacji users**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| `id_user` | BIGINT | PK | NOT NULL UNIQUE |

| | | | |
|---|---|---|---|
| name | VARCHAR(100) | - | NOT NULL |
| surname | VARCHAR(100) | - | NOT NULL |
| login | VARCHAR(50) | - | NOT NULL |
| email | VARCHAR(200) | - | NOT NULL |
| password | VARCHAR(100) | - | NOT NULL |
| active | BOOLEAN | - | NOT NULL |

Relacja sessions zawierająca dane sesji została szczegółowo przedstawiona w Tabeli 8, warto zauważyć że id sesji to unikalny łańcuch znaków o długości 37.

**Tabela 8. Szczegóły pól zawarte w relacji sessions**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_session | VARCHAR(37) | PK | NOT NULL UNIQUE |
| id_user | BIGINT | FK auth.users(id_user) | NOT NULL |
| ip | VARCHAR(40) | - | NOT NULL |
| start_date | TIMESTAMP | - | NOT NULL |
| end_date | TIMESTAMP | - | NOT NULL |

Szczegółowo przedstawiona relacja session_log w Tabeli 9. jest relacją przyrostową, oznacza to że na tej relacji wykonywany jest sam INSERT.

**Tabela 9. Szczegóły pól zawarte w relacji sessions_log**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_session | VARCHAR(37) | - | NOT NULL |
| id_user | BIGINT | - | NOT NULL |
| ip | VARCHAR(40) | - | NOT NULL |
| start_date | TIMESTAMP | - | NOT NULL |
| end_date | TIMESTAMP | - | NOT NULL |

Tabela 10. przedstawia szczegółowy opis pól w relacji plugins, rozszerzony słownik o pola version oraz info, gdzie info może przechowywać stylizowany HTML z opisem wtyczki.

**Tabela 10. Szczegóły pól zawarte w relacji plugins**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_plugin | BIGINT | PK | NOT NULL UNIQUE |
| name | VARCHAR(100) | - | NOT NULL |
| version | VARCHAR(20) | - | NOT NULL |

| | | | |
|---|---|---|---|
| info | TEXT | - | NOT NULL |

Struktura relacji body_plugins została opisana w Tabeli 11. Relacja została stworzona głównie po to, aby oddzielić plik obsługi wtyczki po stronie klienta od reszty informacji o wtyczce w celu optymalizacji bazy.

**Tabela 11. Szczegóły pól zawarte w relacji body_plugins**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_plugin | BIGINT | FK plugin.plugins(id_plugin) | NOT NULL |
| front_body | TEXT | - | NOT NULL |

Tabela 12. przedstawia szczegóły pól relacji lsp_plugins. Program o nazwie zawarty w polu program_name to serwer języka. Cała informacja o LSP wraz z konfiguracją wypełniana jest przez wtyczkę.

**Tabela 12. Szczegóły pól zawarte w relacji lsp_plugins**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_lsp | BIGINT | PK | NOT NULL UNIQUE |
| id_plugin | BIGINT | FK plugin.plugins(id_plugin) | NOT NULL |
| program_name | VARCHAR(100) | - | NOT NULL |
| activator_files | VARCHAR(50) | - | NOT NULL |
| language_id | VARCHAR(10) | - | NOT NULL |
| config | TEXT | - | NOT NULL |

Relacja user_plugins o konstrukcji podanej w Tabeli 13. jest relacją rozwiązującą połączenie wiele do wielu.

**Tabela 13. Szczegóły pól zawarte w relacji user_plugins**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| id_user_plugin | BIGINT | PK | NOT NULL UNIQUE |
| id_user | BIGINT | FK auth.users(id_user) | NOT NULL |
| id_plugin | BIGINT | FK plugin.plugins(id_plugin) | NOT NULL |

Relacja `project_plugins`, tak samo jak wyżej, jest relacją rozwiązującą połączenie wiele do wielu. Strukture relacji opisuje Tabela 14.

**Tabela 14. Szczegóły pól zawarte w relacji project_plugins**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| `id_project_plugin` | BIGINT | PK | NOT NULL UNIQUE |
| `id_project` | BIGINT | FK `project.projects(id_project)` | NOT NULL |
| `id_plugin` | BIGINT | FK `plugin.plugins(id_plugin)` | NOT NULL |

Tabela 15. przedstawia relacje typu słownik o nazwie `projects`, słownik ten zawiera identyfikator oraz nazwę.

**Tabela 15. Szczegóły pól zawarte w relacji projects**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| `id_project` | BIGINT | PK | NOT NULL UNIQUE |
| `name` | VARCHAR(100) | - | NOT NULL |

Relacja przedstawiona w Tabeli 16. nosi nazwę `user_projects`, jest to relacja rozwiązująca połączenie wiele do wielu z dodatkową informacją w polu, czy użytkownik jest właścicielem projektu.

**Tabela 16. Szczegóły pól zawarte w relacji user_projects**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| `id_user_project` | BIGINT | PK | NOT NULL UNIQUE |
| `id_user` | BIGINT | FK `auth.users(id_user)` | NOT NULL |
| `id_project` | BIGINT | FK `project.projects(id_project)` | NOT NULL |
| `owner` | BOOLEAN | - | NOT NULL DEFAULT TRUE |

Pola zawarte w relacji `fn_messages` zostały szczegółowo przedstawione w Tabeli 17. Relacja to rozszerzony słownik o pola `was_ok` oraz `code`.

**Tabela 17. Szczegóły pól zawarte w relacji fn_messages**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| `id_message` | INTEGER | PK | NOT NULL UNIQUE |
| `was_ok` | BOOLEAN | - | NOT NULL |

| | | | |
|---|---|---|---|
| code | INTEGER | - | NOT NULL UNIQUE |
| message | TEXT | - | NOT NULL |

Tabela 18. Przedstawia szczegółowy opis pól relacji o nazwie configurations, która prezentuje się jako relacja *key-value*. Te relacje przechowują unikalny klucz i wartość.

**Tabela 18. Szczegóły pól zawarte w relacji configurations**

| Nazwa | Typ | Klucz | Modyfikator |
|---|---|---|---|
| name | VARCHAR(100) | - | UNIQUE |
| value | TEXT | - | - |

### 3.2.3 Opisy mechanizmów bazy danych

Podczas projektowania systemu zostały wykorzystane mechanizmy relacyjnych baz danych. Spośród mechanizmów tj.: widoki, funkcje i procedury zostały użyte głównie funkcje. Element ten stanowi pewien rodzaj interfejsu między bazą a serwerem tworząc kolejną abstrakcyjną warstwę. Tabela 19. przedstawia szczegółowy opis utworzonych funkcji.

**Tabela 19. Opisy funkcji jako mechanizmy bazy danych**

| Schemat | Nazwa | Opis |
|---|---|---|
| auth | create_session | Generuje klucz sesji i przypisuje go do użytkownika w celu identyfikacji. Zwraca klucz. |
| | login | Sprawdza czy podany użytkownik istnieje i czy dane logowania są poprawne. Wcześniej odszyfrowuje hasło, jeśli dane są niepoprawne to zwraca komunikat o błędzie. W przeciwnym razie wywołuje funkcję create_session i wylicza długość trwania sesji. Zwraca sesję, nazwę użytkownika, czas trwania sesji o id użytkownika. |
| | loginSession | Sprawdza czy sesja istnieje. Jeśli nie istnieje to zwraca komunikat o błędzie. W przeciwnym razie przedłuża sesję i zwraca identyczne dane, jak w przypadku funkcji auth.login. |
| | logout | Sesję wstawia do tabelki *sessions_log* i usuwa ją. |
| | addUser | Tworzy nowego użytkownika na podstawie podanych danych. |
| | getSessionStatus | Sprawdza stan sesji czy istnieje i czy się nie skończyła. |
| plugin | allInstalled | Zwraca listę wszystkich dostępnych wtyczek z informacją, czy dla podanego id użytkownika i id projektu jest zainstalowana wtyczka. |
| | getByUser | Zwraca listę wtyczek przyłączonych do użytkownika o podanym id. |
| | getByProject | Zwraca listę wtyczek przyłączonych do projektu o podanym id. |

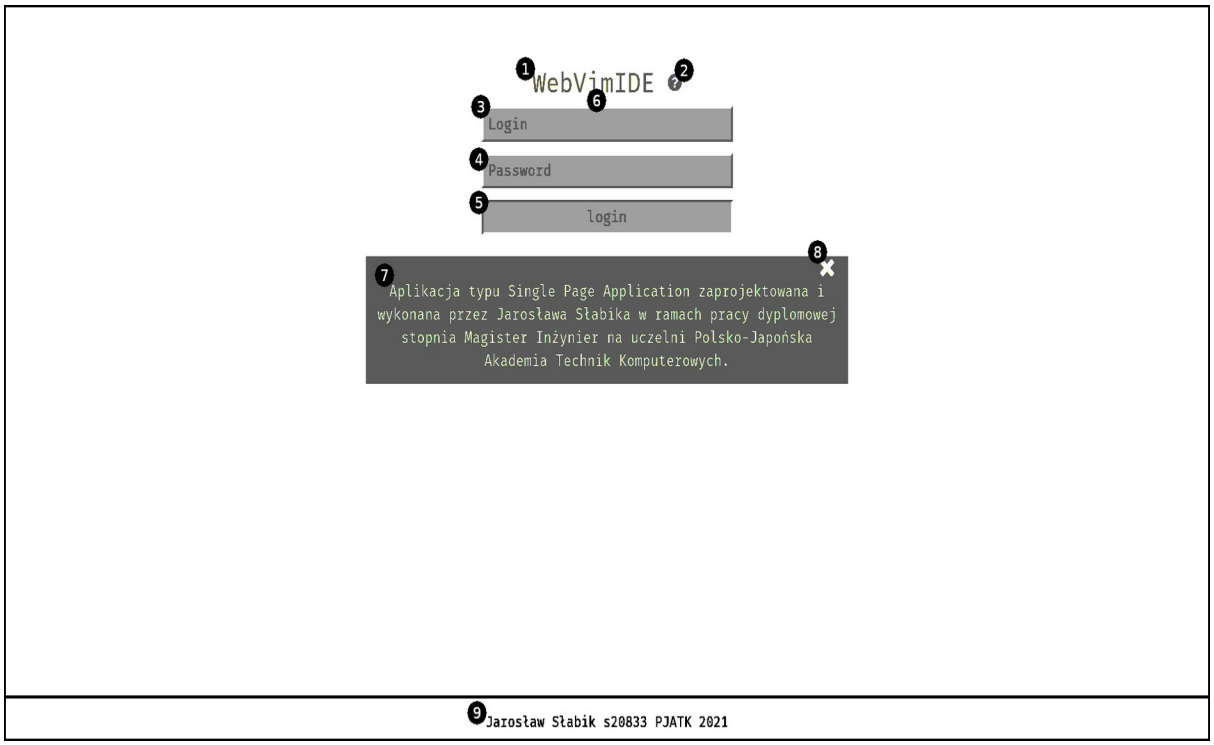
| | | |
|---|---|---|
| | getByUserProject | Zwraca listę wtyczek przyłączonych do użytkownika o podanym id, pomijając wtyczki przyłączone do projektu o wskazanym id. |
| | getById | Zwraca wtyczkę o podanym id. |
| | resolveIdByName | Zwraca id wtyczki na podstawie podanej nazwy. |
| | linkToProject | Przyłącza podaną wtyczkę do projektu o wskazanym id. |
| | detachFromProject | Odłącza podaną wtyczkę od projektu o wskazanym id. |
| | linkToUser | Przyłącza podaną wtyczkę do użytkownika o wskazanym id. |
| | detachFromUser | Odłącza podaną wtyczkę od użytkownika o wskazanym id. |
| | addTmp | Dodaje wtyczkę do rejestracji. |
| | syncTmp | Rejestruje wtyczki. |
| | getLSPByUserAndProject | Zwraca listę LSP na podstawie id użytkownika i id projektu. |
| | registerLSP | Rejestruje LSP. |
| project | addNew | Dodaje nowy projekt o wskazanej nazwie. Użytkownik zostaje oznaczony jako twórca. |
| | getAllByUser | Zwraca listę projektów na podstawie id użytkownika. |
| | resolveIdByName | Zwraca id projektu na podstawie podanej nazwy. |
| | checkPermissionsAndGetName | Sprawdza uprawnienia podanego użytkownika do projektu o wskazanym id. |
| | delete | Usuwa projekt o wskazanym id. |
| | detachUser | Odłącza podany projekt od użytkownika o wskazanym id. |
| | linkUser | Przyłącza podany projekt do użytkownika o wskazanym id. Użytkownik zostaje oznaczony jako gość. |
| | rename | Zmienia nazwę projektu o wskazanym id na nową nazwę, podaną jako parametr. |

## 3.3. Projekt interfejsu użytkownika

W tym podrozdziale zostały przedstawione projekty interfejsów użytkownika zwane inaczej makietami (*UI mockup*). Każda z makiet posiada numery przy kluczowych elementach. Numery te służą łatwiejszej identyfikacji elementu w późniejszych tabelach z opisami.

### 3.3.1 Projekt okna logowania

Rysunek 18. przedstawia projekt okna logowania.




Rysunek 18. Projekt widoku okna logowania

W Tabeli 20. zostały opisane elementy okna logowania z Rysunku 18. W opisie zostały zawarte takie informacje jak: id, typ elementu, alternatywna zawartość oraz opis.

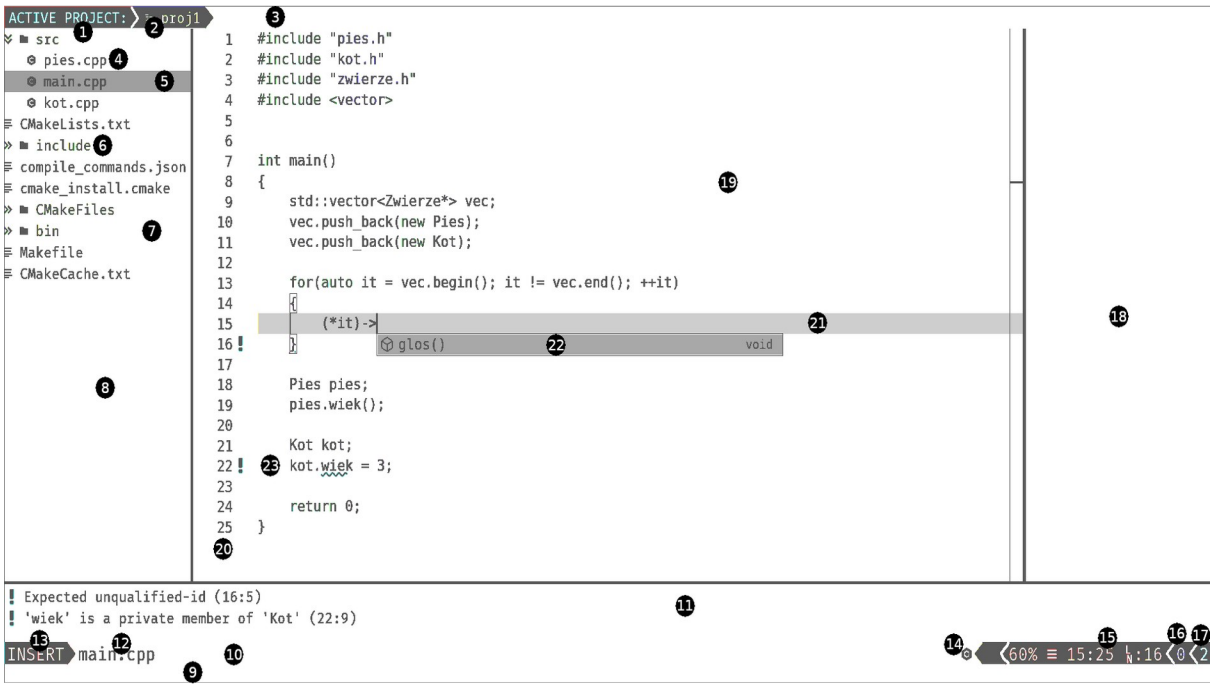
**Tabela 20. Opis elementów dla okna logowania**

| Id | Typ elementu | Alternatywna zawartość | Opis |
|---|---|---|---|
| 1 | Tekst | | Nazwa projektu będąca jednocześnie logotypem. |
| 2 | Przycisk | | Przełącza widoczność okna o id 7. |
| 3 | Pole wprowadzania tekstu | Symbol zastępczy: *Login* | Użytkownik wypełnia te pole swoim loginem. |
| 4 | Pole wprowadzania hasła | Symbol zastępczy: *Password* | Użytkownik wypełnia te pole swoim hasłem. |
| 5 | Przycisk | | Inicjuje proces logowania. |
| 6 | Tekst | Treść błędu np. *Incorrect* | Miejsce na komunikat o błędzie podczas |

| | | | |
|---|---|---|---|
| | | *login or password.* | procesu logowania. |
| 7 | Okno modalne | | Element domyślnie ukryty. Zawiera krotki opis prototypu. |
| 8 | Przycisk | | Zamyka okno modalne o id 7. |

### 3.3.2 Projekt głównego okna

Rysunek 19. przedstawia projekt okna głównego.




Rysunek 19. Projekt widoku okna głównego

W Tabeli 21. zostały opisane elementy okna głównego z Rysunku 19. W opisie zostały zawarte takie informacje jak: id, typ elementu, alternatywna zawartość oraz opis.

**Tabela 21. Opis elementów dla okna głównego**

| Id | Typ elementu | Alternatywna zawartość | Opis |
|---|---|---|---|
| 1 | Text | *„ACTIVE PROJECT:”, „SELECT PROJECT:”, „Lack projects, run command 'new project [name]' to create new project or 'help' to read help”* | Wyświetla aktualny tryb listy projektów. |
| 2 | Text | [ikona] [nazwa_projektu] | Nazwę projektu z poprzedzającą ją ikonką projektu. |
| 3 | Lista wybieralna | | Wyświetla listę dostępnych projektów do wybrania lub już wybrany projekt. Podświetla aktualnie uruchomiony projekt |

| | | | |
|---|---|---|---|
| | | | oraz projekt zaznaczony w celu nawigacji po liście. |
| 4 | Tekst | [ikona] [nazwa_pliku] | Nazwa pliku z poprzedzającą ją ikonką rozszerzenia. |
| 5 | Wiersz | | Podświetlenie aktualnie otwartego pliku. |
| 6 | Tekst |  [nazwa_folderu] | Nazwa folderu z poprzedzającą ją parą ikonek. Pierwsza od lewej symbolizuje jeden z stanów folderu, zamknięty lub otwarty. Druga zaś przedstawia folder. |
| 7 | Drzewo | | Wyświetla pliki i katalogi projektu. |
| 8 | Panel | | Możliwa zmiana rozmiaru lub całkowite schowanie. Zazwyczaj zawiera drzewo plików i katalogów. |
| 9 | Pole wprowadzania tekstu | $ [wpisywana_komenda] | Służy do wprowadzania komend, gdy jest aktywna pojawia się znak zachęty na początku linii. |
| 10 | Lista | | Pasek stanu edytora. |
| 11 | Panel | | Przestrzeń dla dodatkowej zawartości. Najczęściej są to komunikaty i wyniki komend. |
| 12 | Tekst | [nazwa_pliku] | Nazwa aktualnie edytowanego pliku. |
| 13 | Tekst | NORMAL, EDIT, COMMAND | Aktualny tryb edytora. |
| 14 | Tekst | | Ikonka przedstawiająca rozszerzenie edytowanego pliku. |
| 15 | Tekst | [php]% ≡ [nl]:[lp] ln: [nz] | Położenie karetki względem pliku:<br>php – procentowa horyzontalna pozycja;<br>nl – numer linii;<br>lp – ilość linii w pliku;<br>nz – numer znaku. |
| 16 | Liczba | | Ilość ostrzeżeń w edytowanym pliku |
| 17 | Liczba | | Ilość błędów w edytowanym pliku |
| 18 | Panel | | Przestrzeń dla dodatkowej zawartości. |
| 19 | Panel | | Główna przestrzeń robocza. Zawiera edytor pliku. |
| 20 | Lista numerowana | | Margines edytora do wyświetlania numerów linii oraz znaków ostrzeżeń i błędów. |
| 21 | Wiersz | | Podświetlana aktualnie edytowana linia. |
| 22 | Lista wybieralna | | Domyślnie element schowany. Pojawia się w momencie wpisania szczególnego znaku (lista znaków pochodzi od LSP) lub po naciśnięciu kombinacji klawiszy |

| | | | |
|---|---|---|---|
| | | | Ctrl+Spacja. Pozycja składa się z trzech elementów: ikony symbolu/ źródła, sugerowany tekst oraz typ. Zatwierdzenie enterem powoduje wstawienie wybranego tekstu do edytora. |
| 23 | Dekorator wiersza | | Wskazuje linię ikoną oraz fragment kodu za pomocą podkreślenia w miejscu błędu lub ostrzeżenia. |

# 4. Implementacja

W tym rozdziale zostały przedstawione technologie wybrane do implementacji projektu, następnie ukazano skrypty implementujące relacyjną bazę danych, kawałki kodu napisane w języku C++ realizujące logikę biznesową, fragmenty warstwy wizualnej w JavaScript. Na końcu znajdują się zrzuty ekranu przedstawiające gotowe okna IDE.

## 4.1. Wybrane technologie

Do stworzenia prototypu zostały wykorzystane łącznie pięć technologi, które poniżej zostały krótko opisane:

- HTML – (od ang. *HyperText Markup Language*) język znaczników stosowany do tworzenia struktury strony;
- CSS – (od ang. *Cascading Style Sheets*) język służący do opisu formy prezentacji stron;
- JS – język skryptowy JavaScript, służący do dynamicznego tworzenia treści na stronie internetowej;
- C++ – język programowania ze statycznym typowaniem do ogólnego przeznaczenia;
- SQL – strukturalny język zapytań używany do pracy z relacyjnymi bazami danych.

Aby zapewnić projektowi po stronie klienta najlepszą wydajność zrezygnowano z zastosowania frameworku kosztem wydłużonej pracy twórczej [18]. Zrezygnowano również z technologii wymagających konwertowania/ kompilacji tj.: PUB, LESS czy TS.

Identyczną ideologią autor pracy kierował się tworząc usługę sieciową, sam C++ bez użycia jakichkolwiek frameworków spotęgowały liczbę linii kodu ale przyczyniło to się do wzrostu wydajności. Spośród wielu języków programowania został wybrany C++ ze względu na jego szybkość wykonywania działań, którą można zaobserwować w badaniu *A comparison of common programming languages used in bioinformatics* [19]. Badanie te zostało wybrane z dziedziny bioinformatyki ze względu na obszerne rozmiary zbiorów danych oraz nietrywialnych obliczeń. Badanie porównuje wydajność języków programowania tj.: C, C++, Python, C#, Java oraz Perl w implementacjach następujących algorytmów: *Sellers*, budowa drzewa *Neighbor-Joining* oraz BLAST (od ang. *Basic Local Alignment Search Tool*).

*Web Service* komunikuje się z relacyjną bazą danych za sprawą samego języka SQL bez użycia mechanizmu ORM (od ang. *Object-Relational Mapping*), zostało to podyktowane głównie faktem że C++ nie zawiera mechanizmu refleksji. Programy napisane we wspomnianym języku nie znają swojej struktury. C++ posiada prostą odmianą refleksji o nazwie RTTI (od ang. *Run Time Type Information*), czyli dynamiczne informacje o typie. Mechanizm ten jest niewystarczający do budowy ORM. Na rynku istnieją biblioteki dodające ORM do C++, lecz wspomniane mapowanie zostało zaimplementowane na bazie pre-procesów, innymi słowy kod dobudowuje się podczas kompilacji. Użycie surowego SQL generuje potencjalną dziurę w zabezpieczeniu o nazwie *SQL Injection* ale zostały tu wdrożone mechanizmy zabezpieczające przed tą luką.

## 4.2. Implementacja bazy danych

Skrypt do inicjalizacji bazy został podzielony na kilka kluczowych fragmentów, a każdy trafił do innego pliku z rozszerzeniem *.sql. Pliki to kolejno:

- init_database.sql – tworzy bazę danych `db_ide` z kodowaniem UTF-8 oraz dodaje rozszerzenie `uuid-ossp`, które umożliwia generowanie uuid;
- auth_struct.sql – tworzy schemat `auth`, a w nim tabele do opisu użytkownika;
- auth_functions.sql – tworzy funkcje bazodanowe z operacjami na użytkowniku;
- plugin_struct.sql – tworzy schemat `plugin`, a w nim tabele opisujące wtyczki i LSP;
- plugin_functions.sql – tworzy interfejs między strukturą bazy opisującą wtyczki a logiką biznesową;
- project_struct.sql – tworzy schemat `project`, a w nim relacje opisujące projekty;
- project_functions.sql – tworzy zbiór funkcji operujący na tabelach w schemacie `project`;
- public_struct.sql – tworzy niepołączone tabele w schemacie domyślnym jakim jest `public`.

Listing 1. zawiera fragmenty pliku plugin_functions.sql, które przedstawiają definicje funkcji `allInstalled`, `addTmp` oraz `syncTmp` zawarte w schemacie `plugin`.

Listing 1. Fragmenty pliku plugin_functions.sql

```
1   CREATE OR REPLACE FUNCTION plugin.allInstalled(id_user_arg BIGINT,
        id_project_arg BIGINT) RETURNS
2   TABLE
3   (
4       id_plugin_ret BIGINT,
5       name_ret VARCHAR(100),
6       version_ret VARCHAR(20),
7       info_ret TEXT,
8       installed_ret BOOLEAN
9   )
10  AS $$
11  BEGIN
12      RETURN QUERY
13          WITH plugins_for_project_and_user AS
            (
14              SELECT pp.id_plugin FROM plugin.project_plugins AS pp WHERE
                    pp.id_project = id_project_arg
15              UNION
16              SELECT up.id_plugin FROM plugin.user_plugins AS up WHERE
                    up.id_user = id_user_arg
17          )
18          SELECT
19              p.id_plugin, p.name, p.version, p.info, pfpau.id_plugin IS NOT NULL
20          FROM
21              plugin.plugins AS p
22              LEFT JOIN plugins_for_project_and_user AS pfpau ON
23                  pfpau.id_plugin = p.id_plugin
24          ;
25  END;
26  $$ LANGUAGE 'plpgsql';
    […]
254 CREATE OR REPLACE FUNCTION plugin.addTmp(name_arg VARCHAR(100), version_arg
    VARCHAR(20), info_arg TEXT, front_body_arg TEXT) RETURNS VOID
255 AS $$
256 BEGIN
```

```
257     IF to_regclass('temp_plugins') IS NULL THEN
258         CREATE TEMP TABLE temp_plugins
259         (
260             name VARCHAR(100),
261             version VARCHAR(20),
262             info TEXT,
263             front_body TEXT
264         );
265     END IF;
266 
267     INSERT INTO temp_plugins VALUES(name_arg, version_arg, info_arg, translate(
    front_body_arg, E'\n', ''));
268 END;
269 $$ LANGUAGE 'plpgsql';
270 
271 CREATE OR REPLACE FUNCTION plugin.syncTmp() RETURNS VOID
272 AS $$
273 DECLARE
274     rec RECORD;
275     id_plugin_var BIGINT;
276 BEGIN
277     IF to_regclass('temp_plugins') IS NULL THEN
278         CREATE TEMP TABLE temp_plugins
279         (
280             name VARCHAR(100),
281             version VARCHAR(20),
282             info TEXT,
283             front_body TEXT
284         );
285     END IF;
286 
287     -- temp_plugins - plugin.user_plugins -> dodaj
288     FOR rec IN (
289         SELECT name, version, info FROM temp_plugins
290         EXCEPT
291         SELECT name, version, info FROM plugin.plugins)
292     LOOP
293         INSERT INTO plugin.plugins VALUES(DEFAULT, rec.name, rec.version,
    rec.info) RETURNING id_plugin INTO id_plugin_var;
294         INSERT INTO plugin.body_plugins VALUES(id_plugin_var, (SELECT
    front_body FROM temp_plugins WHERE name = rec.name AND version = rec.version));
295     END LOOP;
296 
297     -- plugin.user_plugins - temp_plugins -> odejmij
298     FOR rec IN (
299         SELECT name, version, info FROM plugin.plugins
300         EXCEPT
301         SELECT name, version, info FROM temp_plugins)
302     LOOP
303         id_plugin_var := (SELECT id_plugin FROM plugin.plugins WHERE name =
    rec.name AND version = rec.version);
304         DELETE FROM plugin.user_plugins WHERE id_plugin = id_plugin_var;
305         DELETE FROM plugin.project_plugins WHERE id_plugin = id_plugin_var;
306         DELETE FROM plugin.body_plugins WHERE id_plugin = id_plugin_var;
307         DELETE FROM plugin.lsp_plugins WHERE id_plugin = id_plugin_var;
308         DELETE FROM plugin.plugins WHERE id_plugin = id_plugin_var;
309     END LOOP;
310 
311     DROP TABLE temp_plugins;
```

```
END;
$$ LANGUAGE 'plpgsql';
```

## 4.3. Implementacja logiki biznesowej

Projekt został zaimplementowany w języku C++ o standardzie C++14, jako kompilator został użyty g++ w wersji 9.3.0. Do budowy projektu zostało użyte narzędzie do automatycznego zarządzania procesem kompilacji programów o nazwie CMake w wersji 3.10. Przy implementacji została użyta biblioteka libboost w wersji 1.71.0. Jest to jedna z najbardziej cenionych i fachowo zaprojektowanych bibliotek dla C++ na świecie wg Herb Sutter oraz Andrei Alexandrescu [20].

W celu szybszej kompilacji i zachowania czystego kodu, kod logiki biznesowej został podzielony na kilka części:

- główny trzon aplikacji;
- statyczna biblioteka core_lib – dostarcza mechanizm logowania do pliku i wskazanej bazy, zawiera komponenty do budowy API, konfigurator oraz ogólne narzędzia;
- statyczna biblioteka database_lib – obsługuje połączenia z bazą danych postgreSQL oraz realizuje zapytania;
- statyczna biblioteka json_lib – dostarcza *parser* i *serializer* JSONa;
- statyczna biblioteka lsp_lib – obsługuje połączenie z serwerem języka, realizuje żądania oraz dostarcza protokół LSP opisanymi strukturami;
- statyczna biblioteka plugin_support_lib – zawiera interfejs do wtyczek oraz mechanizm ładowania dynamicznie bibliotek;
- statyczna biblioteka webserver_lib – dostarcza serwer HTTP umożliwiający komunikację po REST.

Listing 2. przedstawia metodę `runProcess`, należącą do klasy `Transport` w przestrzeni nazw `LSP`. Wspomniana metoda uruchamia serwer języka o przekazanej jako parametr nazwie, inicjuje synchroniczną komunikację pomiędzy serwerem języka a web serwisem wykorzystując mechanizm *pipe*.

Listing 2. Fragment klasy `Transport` należący do przestrzeni nazw `LSP` zawarty w pliku lsp_transport.cpp przedstawiający metodę uruchamiającą serwer języka z synchroniczną komunikacją za pomocą mechanizmu pipe

```
[…]
bool LSP::Transport::runProcess(const std::string &serverName)
{
    if(m_serverThread != nullptr || m_inPipe != nullptr || m_outPipe != nullptr
|| m_process != nullptr)
        return false;

    m_serverThread = new std::thread([&]() {
        boost::asio::io_service svc;
        m_inPipe = new boost::process::async_pipe(svc);
        m_outPipe = new boost::process::async_pipe(svc);

        m_process = new boost::process::child(serverName,
boost::process::std_in<*m_inPipe, boost::process::std_out> *m_outPipe, svc,
m_processGroup, boost::process::on_exit([&svc](int code, std::error_code ec)
{std::cout << "Exited " << code << " (" << ec.message() << ")\n";
svc.stop();}));
```

```
19
20          boost::asio::high_resolution_timer send_delay(svc);
21
22          boost::function<void()> read_loop;
23          boost::function<void()> write_loop;
24
25          std::array<char, 20000> recv_buffer;
26          read_loop = [&] {
27              m_outPipe->async_read_some(boost::asio::buffer(recv_buffer), [&]
    (boost::system::error_code ec, size_t transferred) {
28                  if (!ec)
29                  {
30                      m_queueResponse.push(std::string(recv_buffer.data(),
    recv_buffer.size()));
31                      read_loop(); // continue reading
32                  }
33              });
34          };
35
36          auto async_wait = [&write_loop, &send_delay](const
    boost::asio::high_resolution_timer::duration &expiry_time, std::string end){
37              send_delay.expires_from_now(expiry_time);
38              send_delay.async_wait([&](boost::system::error_code ec) {
39                  if(!ec)
40                      write_loop(); // continue writing
41
42                  if(!end.empty())
43                      std::cout << "\r\n";
44              });
45          };
46
47          write_loop = [&] {
48              std::unique_lock<std::mutex> lck{ m_mutex };
49              std::string query;
50              if(!m_queueQuery.empty())
51              {
52                  query = m_queueQuery.front();
53                  m_queueQuery.pop();
54              }
55
56              lck.unlock();
57
58              if(query.empty())
59              {
60                  async_wait(50ms, "");
61              }
62              else
63              {
64                  m_inPipe->async_write_some(boost::asio::buffer(query.c_str(),
    query.size()), [&](boost::system::error_code ec, size_t transferred) {
65                      if(!ec) {
66                          async_wait(50ms, "\r\n");
67                      }
68                  });
69              }
70          };
71
72          read_loop();
73          write_loop();
```

```
74
75          svc.run(); // Await all async operations
76      });
77
78      //Wait to start process
79      while(m_process == nullptr || !m_process->running()){}
80      return true;
81  }
    […]
```

## 4.4. Implementacja warstwy wizualnej

Implementacja ta została dokonana za pomocą technologii internetu czyli HTML 5, JS w wersji ES6 oraz CSS 3, przy implementacji zostały użyte następujące komponenty: jQuery w wersji 3.4.1, Monaco Editor w wersji 0.31.0 oraz hotkeys-js w wersji 3.8.7.

### 4.4.1 Implementacja logiki warstwy wizualnej

Poniżej zostały przedstawione listingi fragmentów wybranych plików JS. Pierwszym z przedstawionych listingów jest Listing 3, ukazuje on fragmenty klasy `PowerLineSupport` przedstawiający metodę `runCommand` uruchomiającą komendy oraz wywołanie metody `addCommandLine` z definicją komendy wbudowanej, jaką jest podłączenie wtyczki do użytkownika.

Listing 3. Fragmenty klasy `PowerLineSupport` zawarte w pliku power_line_support.js przedstawiające (od linii 328) metodę uruchomiającą komendy oraz (od linii 529) definicję wbudowanej komendy do podłączenia wtyczki do użytkownika

```
    […]
328 runCommand(command)
329 {
330     let gr = command.match(/help/);
331     if(gr !== null)
332     {
333         let mapHelp = new Map();
334         for (const com of this.#m_listOfCommands)
335         {
336             let before = mapHelp.get(com.membership);
337             if (before === undefined)
338                 mapHelp.set(com.membership, "  - " + com.help +
    "<BR>");
339             else
340                 mapHelp.set(com.membership, before + "  - " +
    com.help + "<BR>");
341         }
342
343         let result = "";
344         for (const key of Array.from(mapHelp.keys()))
345         {
346             result += "- " + key + " commands:<BR>";
347             result += mapHelp.get(key) + "<BR>";
348         }
349
350         resizableContent.setBottomContentAsString(result);
351         resizableContent.showBottomContent(200);
352
353         return;
354     }
```

```
355
356     for (const com of this.#m_listOfCommands)
357     {
358         let gr = command.match(com.regexp);
359         if (gr !== null)
360         {
361             com.callback(gr);
362             return;
363         }
364     }
365
366     let result = "<P class='error_color'>Command not found. Please check help
    typing 'help' command.</P>";
367     resizableContent.setBottomContentAsString(result);
368     resizableContent.showBottomContent(60);
369 }
    […]
529 this.addCommandLine("built-in", /link user plugin (.*)/, "", undefined, "'link
    user plugin [plugin_name]' - Add plugin [plugin_name] to logged user",
    function(group = []){
530     let jsonIn = {
531         plugin: group[1]
532     };
533
534     rest.POST("plugin/linktouser/", jsonIn, function(json){
535         if(json.was_ok)
536         {
537             pluginSupport.createPlugin(json.message.plugin.name,
    Base64.decode(json.message.plugin.body));
538
539             resizableContent.setBottomContentAsString("<SPAN
    class='good_color'>" + json.message.message + "</SPAN>");
540             resizableContent.showBottomContent(60);
541         }
542         else
543         {
544             console.error(json.message);
545             resizableContent.setBottomContentAsString("something went wrong");
546             resizableContent.showBottomContent(60);
547         }
548     });
549 });
    […]
```

Listing 4. przedstawia fragment klasy `BestCodeEditor` odpowiedzialny za mechanizm podpowiadania, który potrafi także synchronizować plik, jeśli istnieją zmiany.

Listing 4. Fragment klasy `BestCodeEditor` zawarty w pliku best_code_editor.js przedstawiający mechanizm podpowiadania

```
    […]
75  if(this.m_completionItemProviderLSP !== undefined)
76      this.m_completionItemProviderLSP.dispose();
77  this.m_completionItemProviderLSP =
    monaco.languages.registerCompletionItemProvider(languageId, {
78      triggerCharacters: completionTriggerCharacters,
79
80      provideCompletionItems: function (model, position, token) {
81          var lastInsertedCharacter = model.getValueInRange({
```

```
            startLineNumber: position.lineNumber,
            startColumn: position.column - 2,
            endLineNumber: position.lineNumber,
            endColumn: position.column
        });

        if(lastInsertedCharacter === "//" || lastInsertedCharacter === " /" || lastInsertedCharacter === ";/" || lastInsertedCharacter === ")/")
            return;

        let request = "";
        let jsonIn = {};
        if(self.#m_listOfChanges.length !== 0)
        {
            request = "project/" + projectSupport.getCurrentProjectId() + "/lsp/completewithsync/";
            jsonIn = {
                path: self.getPath(),
                changes: self.#m_listOfChanges,
                pos: {
                    line: position.lineNumber,
                    row: position.column
                }
            };
        }
        else
        {
            request = "project/" + projectSupport.getCurrentProjectId() + "/lsp/complete/";
            jsonIn = {
                path: self.getPath(),
                pos: {
                    line: position.lineNumber,
                    row: position.column
                }
            };
        }

        let result = [];
        rest.POST(request, jsonIn, function(json){
            if(json.was_ok && json.message.length !== 0)
            {
                json.message.forEach(function(value, key){
                    result.push({
                        label: value.insertText,
                        kind: value.kind,
                        detail: value.detail,
                        insertText: value.insertText
                    });
                });
            }
            else
            {
                console.error(json.message);
            }
        }, function(){}, false);

        self.#m_listOfChanges = [];

```

```
138         return {
139             suggestions: result
140         };
141     }
142 });
    […]
```

### 4.4.2 Implementacja interfejsu użytkownika

W tym podrozdziale zostały przedstawione w formie zrzutów ekranu zaimplementowane okna interfejsu użytkownika. Na potrzeby wydruku oryginalny schemat kolorów „Dracula” został rozjaśniony.

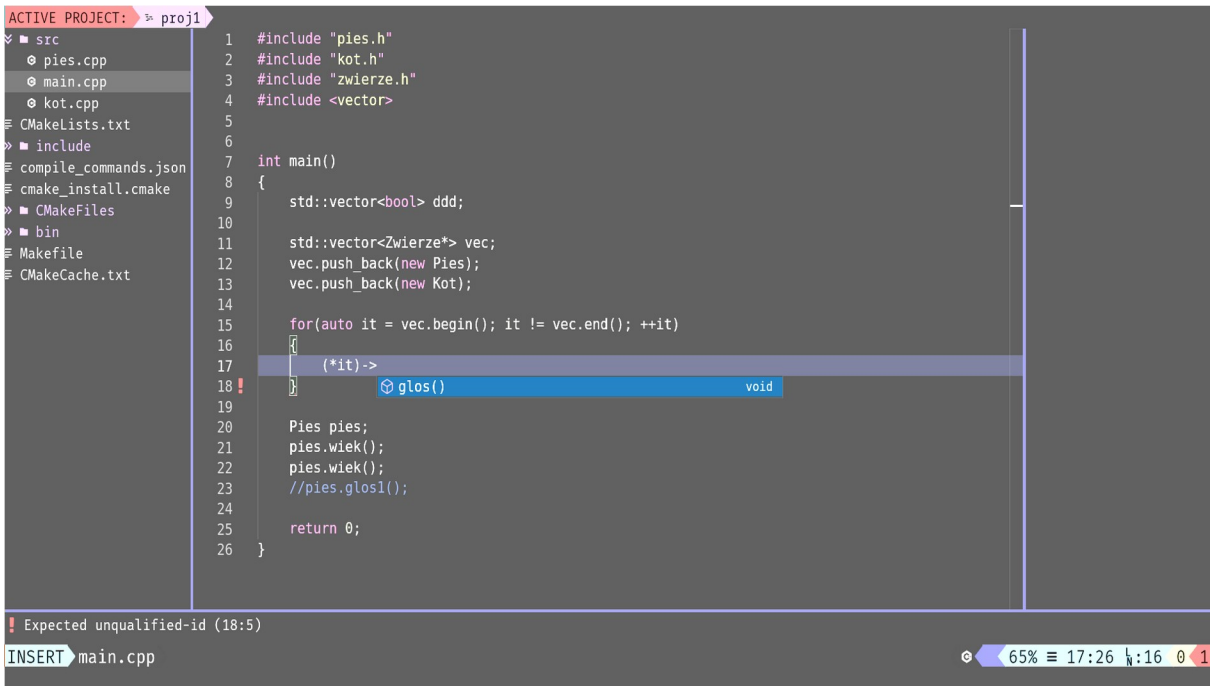
Jako pierwsze zostało zaprezentowane okno logowania w Rysunku 20, okno to pozostaje w spoczynku tzn. brak tutaj żądania od użytkownika do wykonania.


Rysunek 20. Okno logowania

Użytkownicy od razu po zalogowaniu (niemający żadnego przypisanego projektu) zastaną widok przedstawiony w Rysunku 21.

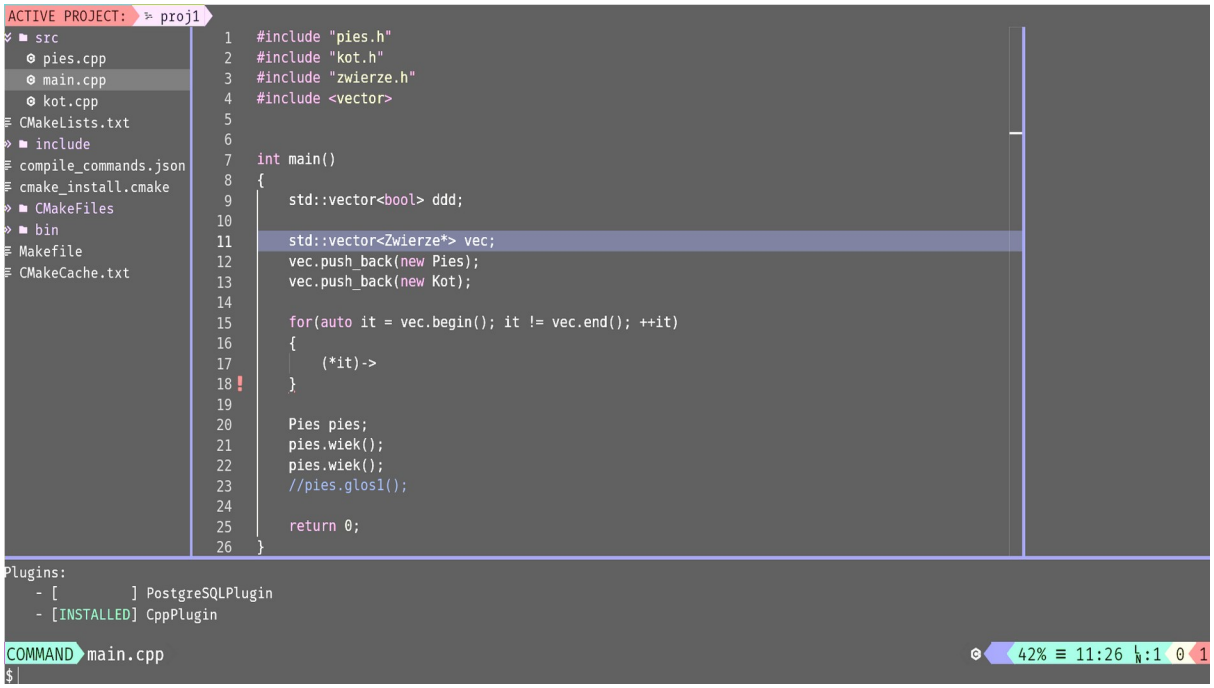

Rysunek 21. Okno główne – wariant brak projektów

W Rysunku 22. został przedstawiony IDE podczas edycji pliku main.cpp w projekcie proj1. Widać na nim efekt podpowiadania po wpisaniu znaku ‘>’ oraz zaznaczenie linii i fragmentu błędu.




Rysunek 22. Okno główne – wariant uruchomiony projekt proj1 oraz edytowany plik main.cpp z widocznymi błędami i uruchomionym auto-uzupełnianiem

Rysunek 23. przedstawia listę dostępnych wtyczek z zaznaczeniem tych, które są zainstalowane. Efekt ten jest uzyskany po uruchomieniu komendy „plugins”.




Rysunek 23. Okno główne – wariant wyniku uruchomionej komendy „plugins”

# 5. Testy wybranych elementów

Na początku tego rozdziału zostały przedstawione przypadki testowe. W następnej części przedstawiono zrzuty ekranów z wybranych przeprowadzonych przypadków testowych. W części końcowej zostały wymienione testy automatyczne.

## 5.1. Przypadki testowe

Przypadki testowe (TC od ang. *Test Case*) zostały zaprezentowane w formie tabeli, ze względu na mnogość przypadków w pracy znalazły się tylko wybrane. Tabela 22. składa się z siedmiu następujących kolumn:

- id – unikalny numer;
- tytuł – krótki i zwięzły opis testu;
- warunki – dane i warunki początkowe potrzebne do wykonania testu;
- kroki – opis krok po kroku co zostało wykonywane;
- rezultat – rezultat wykonania kroków testowych;
- wykonanie – informacja czy przypadek był wykonywany ręcznie (M), czy jest pokryty kodem (A);
- ukazany – informacja czy przypadek testowy został przedstawiony w postaci zrzutów ekranów (T), czy nie (N).

**Tabela 22: Przypadki testowe**

| Id | Tytuł | Warunki | Kroki | Rezultat | Wyko. | Uka. |
|---|---|---|---|---|---|---|
| 1 | Logowanie – poprawne dane. | Użytkownik wylogowany, brak sesji, aktywny ekran logowania. | Uzupełnienie pól *Login* oraz *Password* poprawnymi danymi, kliknięcie w przycisk *login*. | Pojawienie się okna głównego z załadowanymi wtyczkami i projektami o ile takowe są. | M | N |
| 2 | Logowanie – błędne dane. | Użytkownik wylogowany, brak sesji, aktywny ekran logowania. | Uzupełnienie pól *Login* oraz *Password* błędnymi danymi, kliknięcie w przycisk *login.* | Pojawienie się komunikatu na ekranie logowania o błędnych danych. | M | N |
| 3 | Utworzenie nowego projektu. | Użytkownik zalogowany, brak projektów. | Wejście w tryb komendy. Wpisanie komendy „add project proj1” i zatwierdzenie klawiszem Enter. | Generacja nowego projektu w pasku projektów. Uruchomienie nowo utworzonego projektu. Utworzenie katalogu na serwerze. | M | T |
| 4 | Utworzenie nowego projektu o nazwie | Użytkownik zalogowany, istniejący projekt o nazwie | Wejście w tryb komendy. Wpisanie komendy „add project proj1” i zatwierdzenie | Pojawienie się komunikatu o istniejącym projekcie. | M | N |

| | | | | | | |
|---|---|---|---|---|---|---|
| | istniejącego już projektu. | „proj1”. | klawiszem Enter. | | | |
| 5 | Utworzenie nowego pliku. | Użytkownik zalogowany, istniejący i aktywny projekt o nazwie „proj1”. | Wejście w tryb komendy. Wpisanie komendy „add node main.cpp” i zatwierdzenie klawiszem Enter. | W drzewie plików oraz w folderze projektu „proj1” pojawił się plik main.cpp. | M | N |
| 6 | Synchroni-zacja edytowane-go pliku z plikiem na serwerze. | Użytkownik zalogowany, istniejący i aktywny projekt o nazwie „proj1”, gotowy do edycji pusty plik main.cpp. Brak wtyczek. | Wejście w tryb wstawiania. Napisanie funkcji main w języku C++. Wyjście z trybu wstawiania. | Edytor koloruje składnię języka C++, dostępne jest tylko podpowiadanie słownikowe. Pusty plik main.cpp w folderze proj1 otrzymał zawartość edytora. | M | N |
| 7 | Załadowanie wtyczki do języka C++. | Użytkownik zalogowany, brak wtyczek. | Wejście w tryb komendy. Wpisanie „link user plugin CppPlugin” i zatwierdzenie klawiszem Enter. | Komunikat o poprawnym przyłączeniu wtyczki do użytkownika. | M | T |
| 8 | Edycja pliku z wtyczką. | Użytkownik zalogowany, istniejący i aktywny projekt o nazwie „proj1”, gotowy do edycji pusty plik main.cpp. Załadowana wtyczka „CppPlugin”. | Wejście w tryb wstawiania. Napisanie funkcji main w języku C++, używając przy tym prostej struktury. Wyjście z trybu wstawiania. | Edytor koloruje składnię języka C++, podpowiadanie składni oraz zawartości struktury po odwołaniu się do zawartości zmiennej. Pusty plik main.cpp w folderze proj1 otrzymał zawartość edytora. | M | N |
| 9 | Refaktoryza cja | Użytkownik zalogowany, istniejący i aktywny projekt o nazwie „proj1”, gotowy do edycji plik main.cpp. Załadowana wtyczka „CppPlugin”. | W edytorze zaznaczyć symbol kursorem lub karetką. Wejście w tryb komendy i wpisać „rename nowySymbol”. | Wszystkie symbole adekwatne do zaznaczonego zmieniły swoją nazwę na „nowySymbol”. Plik na serwerze został synchronizowany. | M | T |
| 10 | Zaznaczanie błędów. | Użytkownik zalogowany, istniejący i aktywny | Wejście w tryb wstawiania. Napisanie funkcji main w języku C++ | Wszystkie linie i fragmenty kodu z błędami zostały zaznaczone. Zostały | M | N |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | projekt o nazwie „proj1”, gotowy do edycji pusty plik main.cpp. Załadowana wtyczka „CppPlugin”. | tworząc przy tym błędy w składni. Wyjście z trybu wstawiania. | ukazane również treści błędów. | | |
| 11 | Metody HTTP. | | Wykrywanie metody i realizacja zadań podpiętych do tej metody. | Sukces. | A | N |
| 12 | Serializacja tekstu w formacie JSON na instancje klasy i na odwrót. | | Zamiana łańcucha znaków w formacie JSON na instancje klasy. Zamiana instancję klasy na łańcucha znaków w formacie JSON. | Sukces. | A | T |

## 5.2. Testy funkcjonalne

W podrozdziale tym zostały ukazane za pomocą zrzutów ekranów wybrane testy funkcjonalne, każdy test znajduje się w osobnym paragrafie. Zrzuty ekranów prezentują stan przed, w trakcie przeprowadzania testu oraz stan po wykonaniu testu.

### 5.2.1 Utworzenie nowego projektu

Rysunek 24. to zrzut ekranu przedstawiający IDE przed przeprowadzeniem testu funkcjonalnego nr 3. z Tabeli 22. Widać na nim brak projektów oraz już wpisaną komendę przy uruchomionym trybie COMMAND.

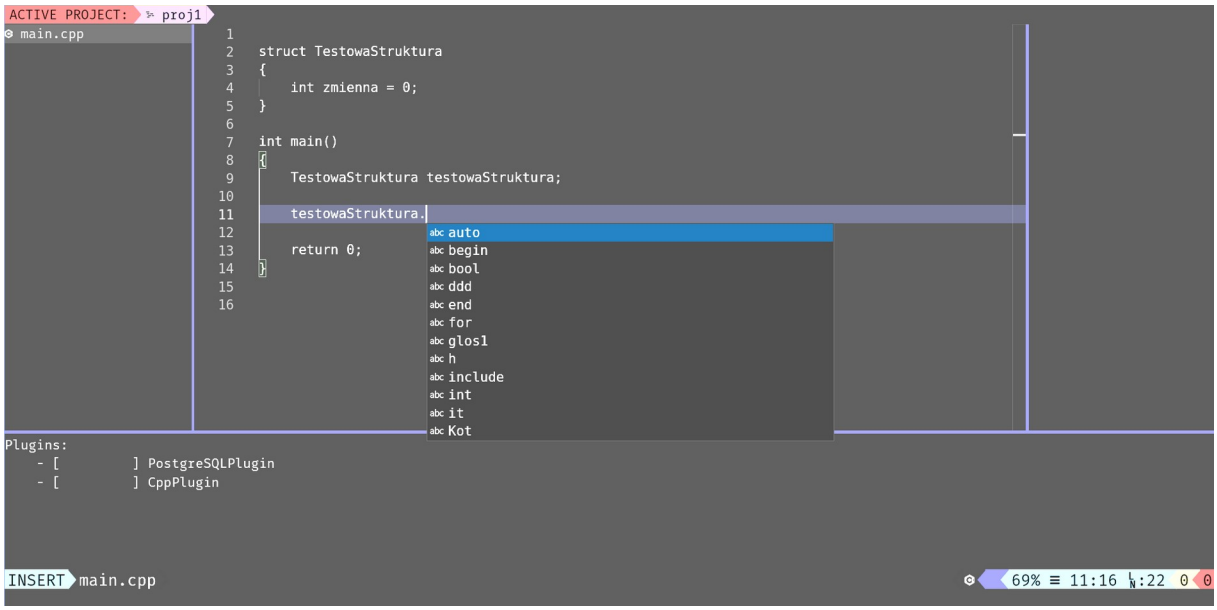

Rysunek 24. Zrzut ekranu przed przeprowadzeniu testu nr 3.

Rysunek 25. to zrzut ekranu przedstawiający IDE po przeprowadzeniem testu funkcjonalnego nr 3. z Tabeli 22. Widać na nim aktywny projekt „proj1”.


Rysunek 25. Zrzut ekranu po przeprowadzeniu testu nr 3.

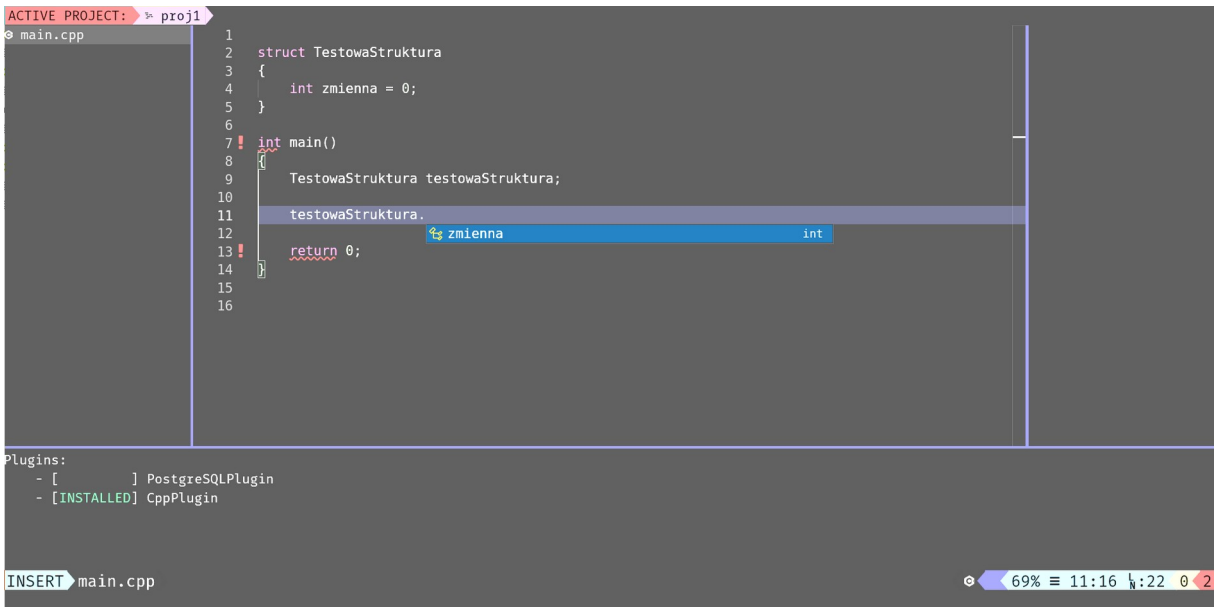
### 5.2.2 Załadowanie wtyczki do języka C++

Stan przed testem załadowania wtyczki nr 7 z Tabeli 22. został przedstawiony w Rysunku 26. jako rzut ekranu. Widać na nim auto-uzupełnianie na podstawie słownika oraz listę wtyczek.




Rysunek 26. Zrzut ekranu przed przeprowadzeniem testu nr 7.

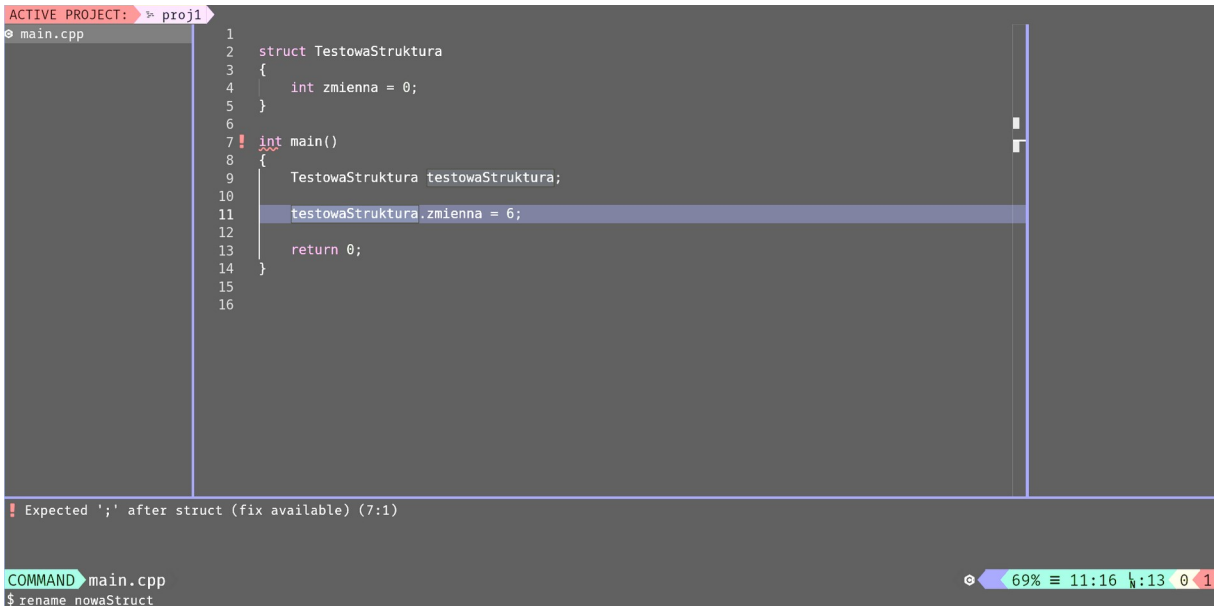
Stan po przeprowadzeniu testu załadowania wtyczki nr 7 z Tabeli 22. został przedstawiony w Rysunku 27. Zrzut ekranu prezentuje pozytywny komunikat oraz auto-uzupełnianie zrealizowany przez LSP.




Rysunek 27. Zrzut ekranu po przeprowadzeniem testu nr 7.

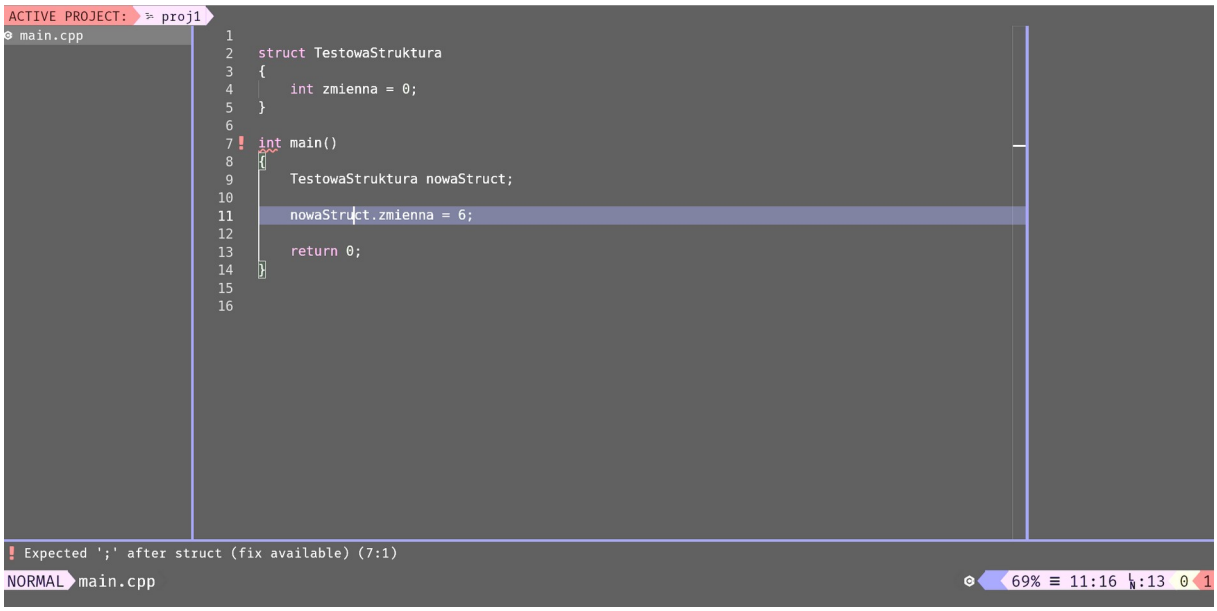
### 5.2.3 Refaktoryzacja

Stan przed testem refaktoryzacji nr 9 z Tabeli 22. został przedstawiony w Rysunku 28. jako rzut ekranu. Widać na nim otworzony do edycji plik main.cpp, zaznaczony symbol „testowaStruktura” oraz wpisana już komenda do zmiany nazwy.




Rysunek 28. Zrzut ekranu przed przeprowadzeniem testu nr 9.

Po przeprowadzeniu testu nr 9. z Tabeli 22. w pliku main.cpp symbol „testowaStruktura” został zamieniony na „nowaStruct”. Rezultat widać na zrzucie ekranu przedstawionym w Rysunku 29.

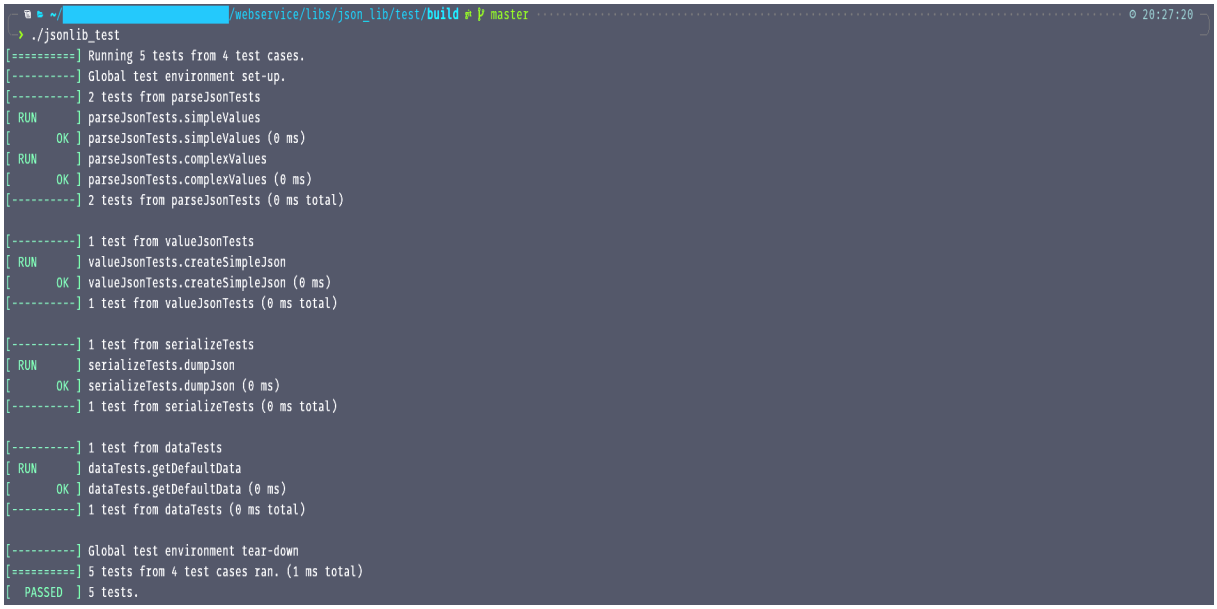



Rysunek 29. Zrzut ekranu po przeprowadzeniu testu nr 9.

## 5.3. Testy jednostkowe

W podrozdziale tym zostały ukazane za pomocą zrzutów ekranów wybrane testy jednostkowe. Zrzuty ekranów prezentują stan po przeprowadzeniu testu.

Rysunek 30. zawiera zrzut ekranu przedstawiający rezultat po przeprowadzeniu testów jednostkowych nr 12. z Tabeli 22. serializacji JSON.




Rysunek 30. Zrzut ekranu po przeprowadzeniu testów jednostkowych

# 6. Analiza systemu na tle istniejących rozwiązań

W rozdziale tym zostały przedstawione kryteria analizy porównawczej, następnie prototyp został porównany do przytoczonych w pracy konkurencyjnych rozwiązań.

## 6.1. Kryteria analizy porównawczej

Wykonana analiza porównawcza została objęta następującymi kryteriami:

- interfejs użytkownika oraz jego możliwości,
- wspierane technologie,
- wtyczki i rozszerzenia,
- funkcje wspomagania,
- subiektywne wrażenia z użytkowania.

## 6.2. Porównanie

W podrozdziale tym zostało wykonane porównanie prototypu z konkurencyjnymi rozwiązaniami przytoczonymi w podrozdziale 1.2. Każdy element w osobnym paragrafie.

### 6.2.1 Interfejs użytkownika oraz jego możliwości

W odróżnieniu od wszystkich przytoczonych konkurencyjnych rozwiązaniach, prototyp nie posiada panelu administracyjnego. Takie rozwiązanie pozwala na „nieodrywaniu" się od edytora kodu. Wszystkie IDE podzielone są na panele, które mają możliwość zmiany rozmiaru, a także całkowitego schowania.

Codeanywhere oraz Eclipse Che posiadają wygląd i funkcjonalność identyczną do tych znanych z VS Code, ponieważ wspomniany edytor kodu jest najpopularniejszy to przytoczone środowiska posiadają największą grupę potencjalnych odbiorców. AWS Cloud9 oraz Replit wyglądem przypominają klasyczne desktopowe IDE. Szczególnie ten pierwszy. Prototyp został wzorowany na modalnym edytorze Vim. Edytor ten wymaga od użytkownika poświęcenia sporo czasu, aby stał się narzędziem produktywnym. Mówi się że krzywa nauki Vima jest stroma a mimo to znajduje się w top 5. programów deweloperskich. Podobnie jest z prototypem, trzeba się go nauczyć, co może stanowić argument o jego odrzuceniu.

Prototyp jako jedyny nie posiada możliwości: debugowania, emulacji terminalu, kontroli wersji oraz konteneryzacji. Autor przekonany jest że trzy ostatnie braki można nadrobić wtyczkami. Debugowanie jest specyficzną funkcją, która wymaga sporej ingerencje w kod.

### 6.2.2 Wspierane technologie

IDE takie jak Codeanywhere, Eclipse Che, Replit oraz prototyp korzystają z edytora Monaco Editor [20], który posiada wbudowaną funkcję kolorowania składni dla niemalże wszystkich języków programowania (aż 80). Warto zwrócić uwagę na to, że wspomniany edytor pochodzi od VS Code.

Wszystkie środowiska deweloperskie oprócz AWS Cloud9 posiadają wsparcie LSP, dzięki czemu tylko brak implementacji serwera języka ogranicza to IDE przed wsparciem wszystkich języków programowania.

Warto nadmienić że Replit jako jedyny udostępnia predefiniowane środowiska z gotową konfiguracją języka programowania wraz z *framework*'iem np. Python + Django.

### 6.2.3 Wtyczki i rozszerzenia

Obsługa wtyczek została zaimplementowana w trzech IDE: Codeanywhere, Eclipse Che oraz w prototypie. Ostatni z nich obsługuje tylko wtyczki własnościowe. Nie potrafi przyswajać ich z gotowych repozytoriów tj. VS Code *Marketplace* jak to ma miejsce w przypadku tych dwóch pierwszych.

### 6.2.4 Funkcje wspomagania

Funkcje edytora we wszystkich zintegrowanych środowiskach są podobne z racji tego, że większość z nich korzysta z LSP. Protokół ten wspiera takie funkcje jak: formatowanie kodu, *refaktoring* symboli, wyłapywanie błędów i ostrzeżeń, podpowiadanie, czy nawigacja po projekcie. W skład nawigacji wchodzą funkcje: skok do definicji, skok do deklaracji oraz wyszukiwanie referencji i skok do wybranej. AWS Cloud9 pomimo braku LSP nie odstępuje konkurencji. Na pochwałę zasługuje Codeanywhere za unikalną funkcjonalność jaką jest analizator wyłapujący fragmenty kodu które można napisać lepiej.

### 6.2.5 Subiektywne wrażenia z użytkowania

Każde z omawianych IDE posiada własną unikalną grupę odbiorców. Dla AWS Cloud9 są to deweloperzy korzystający z AWS, a dla Codeanywhere i Eclipse Che są to użytkownicy VS Code. Dla najmłodszych i uczących się dopiero programowania osób najlepszy wydaje się być Replit. Natomiast prototyp może wzbudzić spore zainteresowanie wśród użytkowników Vim, do których zalicza się autor pracy.

Codeanywhere oraz Replit jako jedyne mają rozbudowane funkcje do współpracy w takim stopniu, że widać pozycje karetki innych członków projektu. Oba z nich mogą pomyślnie być wykorzystywane do nauki zdalnej lub przeprowadzania rekrutacji na odległość na stanowisko deweloperskie.

Autor pracy jako kilkuletni użytkownik Vima widzi duży potencjał w tym projekcie. Prototyp posiada unikalny wygląd oraz spore możliwości rozbudowy o dodatkowe funkcje. Łączy w sobie wybrane cechy omawianych IDE, a zarazem posiada możliwości edytorów modalnych.

# 7. Podsumowanie i wnioski

Celem niniejszej pracy magisterskiej było opracowanie koncepcji elastycznego, modalnego, zintegrowanego środowiska programistycznego wykorzystującego wtyczki oraz działającego w internetowej chmurze. Tak skonstruowany cel zdaniem autora został osiągnięty. Prototyp został zaprojektowany, zaimplementowany i przetestowany w oparciu o przytoczoną koncepcję. Działający prototyp został skonfrontowany z kilkoma konkurencyjnymi rozwiązaniami. Na podstawie tej konfrontacji zostały wyciągnięte następujące wnioski.

Prototyp dzięki swojemu interfejsowi użytkownika trafi do niemałej grupy odbiorców, korzystających na co dzień z modalnego edytora tekstu Vim. Dodatkowo, użytkownikom dla których nauka modalnych edytorów okazała się zbyt trudna, prototyp ułatwi im wdrożenie w tę dziedzinę.

Webowe IDE wspierające mechanizmy współpracy mogą urozmaicić zdalną naukę programowania lub ułatwić przeprowadzanie rozmów rekrutacyjnych na odległość.

Nie wliczając funkcji debugowania przedstawiony IDE nie odbiega znacząco od rozwiązań konkurencji, a niektóre jego elementy działają lepiej. Błyskawiczna synchronizacja zmian, a w tym również edycja i refaktoryzacja jest jednym z tych elementów. Zostało to uzyskane za sprawą wykorzystania dziennika zmian oraz wybranych technologii.

Zaprezentowany projekt można rozwijać poprzez dodanie funkcji: debugowania, emulacji terminalu oraz obsługę kontroli wersji. Dodatkowo konteneryzacja też wygląda na funkcję wartą implementacji. Pomocny byłby również mechanizm ładowania wtyczek z sieci, jak to ma miejsce w VS Code *Marketplace.*

Autor pracy żywi nadzieję, że zaprezentowana praca dyplomowa będzie inspiracją do próby zmierzenia się z modalnymi edytorami tekstu.

# Prace cytowane


1 Martin R. Caga. The HP softBench Environment: An Architecture for a New Generation of Software Tools, pages 36-47. Volume 41 Issue 3. June 1990.

2 AWS Cloud9. https://aws.amazon.com/cloud9/ – Dostęp z dnia 20.12.2021.

3 Eclipse Che (dawniej Codenvy). https://www.eclipse.org/che/ – Dostęp z dnia 20.12.2021.

4 Codeanywhere. https://codeanywhere.com/ – Dostęp z dnia 20.12.2021.

5 PaizaCloud. https://paiza.cloud/en/ – Dostęp z dnia 20.12.2021.

6 Replit. https://replit.com/ – Dostęp z dnia 20.12.2021.

7 SourceLair IDE. https://www.sourcelair.com/#products – Dostęp z dnia 20.12.2021.

8 https://www.freecodecamp.org/news/vim-for-people-who-use-visual-studio-code/ – Rozdział *Keep your hands at 10 and 2*, dostęp z dnia 20.12.2021.

9 https://docs.aws.amazon.com/cloud9/latest/user-guide/language-support.html – Dostęp z dnia 20.12.2021.

10 Eclipse Theia. https://theia-ide.org/ – Dostęp z dnia 20.12.2021.

11 https://www.eclipse.org/che/docs/next/installation-guide/configuring-storage-types/ – Dostęp z dnia 20.12.2021.

12 Dariusz Olczyk. Zeszyty Naukowe Warszawskiej Wyższej Szkoły Informatyki, strony 87-98. Tom 4. Warszawska Wyższa Szkoła Informatyki. 2010. http://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt4/Modelowanie_Strukturalne_-_Definicje_Notacja_Techniki_I_Narzedzia.pdf – Dostęp z dnia 20.12.2021.

13 https://wincent.com/wiki/Modal_editor – Dostęp z dnia 20.12.2021.

14 https://insights.stackoverflow.com/survey/2021#most-popular-technologies-new-collab-tools – Dostęp z dnia 20.12.2021.

15 https://www.quora.com/What-is-the-point-to-learn-and-use-Vim-by-just-coding-faster-than-1-to-2-seconds-than-other-editors – Dostęp z dnia 20.12.2021.

16 https://microsoft.github.io/language-server-protocol/specifications/specification-3-16/ – Dostęp z dnia 20.12.2021.

17 http://www.viemu.com/a-why-vi-vim.html – Dostęp z dnia 20.12.2021.

18 https://krausest.github.io/js-framework-benchmark/2021/table_chrome_96.0.4664.45.html – Dostęp z dnia 20.12.2021.

19 Fourment, M., Gillings, M.R. A comparison of common programming languages used in bioinformatics. BMC Bioinformatics. Volume 9 Issue 82. February 2008. https://doi.org/10.1186/1471-2105-9-82 – dostęp z dnia 20.12.2021.

20 Herb Sutter, Andrei Alexandrescu. C++ Coding Standards. 101 Rules, Guidelines, and Best Practices, page 147. ISBN 0-321-11358-6.

21 https://microsoft.github.io/monaco-editor/ – Dostęp z dnia 20.12.2021.