НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.02

«До захисту допущено»
В.о. завідувача кафедри
_____ Едуард ЖАРІКОВ
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Метод та засоби автоматизованого створення первинного мультиплатформенного mobile-проєкту»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІП-02мп
Швець Едуард Ярославович _____

Керівник:
доц., к.т.н., викладач кафедри ІПІ
Муха Ірина Павлівна _____

Рецензент:
доц., к.т.н., викладач кафедри ІСТ
Ткач Михайло Мартинович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Швецю Едуарду Ярославовичу

1. Тема дисертації «Метод та засоби автоматизованого створення первинного мультиплатформенного mobile-проєкту», науковий керівник дисертації доц., к.т.н., викладач кафедри ІПІ Муха Ірина Павлівна, затверджені наказом по університету від «27» жовтня 2021 р. № 3587-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об’єкт дослідження – Процес створення мультиплатформенного mobile-проекту.

4. Предмет дослідження – Методи та засоби автоматизованого створення шаблону мультиплатформенного mobile-проекту

5. Перелік завдань, які потрібно розробити:

- дослідити процес створення мультиплатформенного мобільного проєкту та наявні засоби його автоматизації;
- розробити метод автоматизованого створення первинного мультиплатформенного мобільного проєкту.
- розробити програмне забезпечення для реалізації запропонованого методу.
- провести дослідження ефективності запропонованих рішень.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 4 плакати

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Вивчення рекомендованої літератури	01.09.2021	
2	Аналіз існуючих методів розв’язання задачі	13.09.2021	
3	Постановка та формалізація задачі	19.09.2021	
4	Аналіз вимог до програмного забезпечення	23.10.2021	
5	Моделювання програмного забезпечення	01.10.2021	
6	Розробка програмного забезпечення	21.10.2021	
7	Виконання експериментальних досліджень	08.11.2021	
8	Оформлення пояснювальної записки	16.11.2021	
9	Подання дисертації на попередній захист	22.11.2021	
10	Подання дисертації на захист	6.12.2021	

Студент Едуард ШВЕЦЬ

Науковий керівник Ірина МУХА

РЕФЕРАТ

Магістерська дисертація: 97 с., 24 рис., 27 табл., 3 додатки, 36 джерел

Актуальність теми. Мультиплатформенний підхід до розробки мобільних застосувань дозволяє створювати додатки для різних платформ шляхом одноразового написання блоку універсального коду, а потім спільного використання цього коду на різних платформах. Тому мультиплатформенна розробка мобільних застосунків є актуальною темою.

Мета досліджень. Пришвидшити процес розробки мультиплатформенного мобільного застосунку.

Для реалізації поставленої мети були сформовані **наступні завдання**:

– дослідити процес створення мультиплатформенного мобільного проєкту та наявні засоби його автоматизації;

– розробити метод автоматизованого створення фрагментів мультиплатформенного мобільного проєкту;

– розробити програмне забезпечення для реалізації запропонованого методу;

– провести дослідження ефективності запропонованих рішень.

Об’єкт досліджень. Процес створення мультиплатформенного mobile-проекту.

Предмет досліджень. Метод та засоби автоматизованого створення первинного мультиплатформенного mobile-проекту.

Наукова новизна отриманих результатів полягає у:

– розробці проблемно-орієнтованої мови для опису сутностей БД та налаштувань DAO, CRUD, DI, які є вхідними даними для автоматизованого створення первинного мультиплатформенного мобільного проєкту;

– запропонованому методі автоматизованого створення первинного мультиплатформенного мобільного проєкту, який передбачає транспіляцію вхідних описів сутностей БД та налаштувань DAO, CRUD, DI у SQL та Kotlin компоненти загального коду мультиплатформенного мобільного проєкту.

Практичне значення отриманих результатів у застосуванні програмного забезпечення для автоматизації процесу створення первинного мультиплатформенного мобільного проєкту, який представляє собою програмну реалізацію компонентів сервісів роботи з БД.

Апробація. Результати роботи доповідались на «Першій Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021)».

Публікації. Наукові положення опубліковані в тезах наукової конференції «Перша Всеукраїнська науково-практична конференця молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021)».

Ключові слова: ПЕРВИННИЙ МУЛЬТИПЛАТФОРМЕННИЙ ПРОЄКТ, KOTLIN MULTIPLATFORM MOBILE, МУЛЬТИПЛАТФОРМЕННА МОБІЛЬНА РОЗРОБКА, ПРЕДМЕТНО-ОРІЄНТОВАНА МОВА, АВТОМАТИЗАЦІЯ, ГРАМАТИКА, ПЛАГІН

ABSTRACT

Master's dissertation: 91 pp., 24 figs., 27 tables, 3 appendices, 36 sources

Actuality of theme. The multi-platform approach to mobile application development allows you to create applications for different platforms by writing a single block of universal code, and then sharing this code on different platforms. Therefore, multi-platform development of mobile applications is a topical issue.

The purpose of research. Accelerate the process of developing a multi-platform mobile application.

To achieve this goal, the following **tasks were formed:**

– to study the process of creating a multi-platform mobile project and the available means of its automation;

– to develop a method of automated creation of fragments of a multiplatform mobile project;

– to develop software for the implementation of the proposed method;

– to study the effectiveness of the proposed solutions.

Object of research. The process of creating a multi-platform mobile project.

Subject of research. Method and means of automated creation of primary multi-platform mobile-project.

The scientific novelty of the results is:

– development of problem-oriented language to describe the essence of the database and settings DAO, CRUD, DI, which are the input data for the automated creation of the primary multi-platform mobile project;

– the proposed method of automated creation of the primary multiplatform mobile project, which involves translating input descriptions of database entities and settings DAO, CRUD, DI in SQL and Kotlin components of the common code of the multiplatform mobile project.

The practical significance of the results obtained in the use of software to automate the process of creating a primary multi-platform mobile project, which is a software implementation of the components of database services.

Approbation. The results of the work were presented at the "First All-Ukrainian scientific-practical conference of young scientists and students" Software Engineering and Advanced Information Technologies (SoftTech-2021) ".

Publications. Scientific provisions were published in the abstracts of the scientific conference "The First All-Ukrainian Scientific and Practical Conference of Young Scientists and Students" Software Engineering and Advanced Information Technologies (SoftTech-2021) ".

Keywords: PRIMARY MULTIPLATFORM PROJECT, KOTLIN MULTIPLATFORM MOBILE, MULTIPLIPFORM MOBILE DEVELOPMENT, SUBJECT-ORIENTED LANGUAGE, LANGUAGE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПК – персональний комп’ютер;

ОС – операційна система;

ПЗ – програмне забезпечення;

IDE (Integrated development environment) – комплекс програмних засобів, що використовується програмістами для розробки програмного забезпечення.

БД – база даних;

SQL – декларативна мова програмування, що використовується для створення, модифікації та управління даними в реляційній БД;

SDK (Sofware development kit) – набір засобів розробки, що дозволяє фахівцям з програмного забезпечення створювати програми;

DAO (Data Access Object) – абстрактний інтерфейс доступу до БД;

CRUD – акронім, що означає чотири базові функції, які використовують під час роботи з БД: створення (create), читання (read), модифікація (update), видалення (delete).

ВСТУП

Останнім часом все більшої популярності у світі мобільної розробки набуває мультиплатформенна розробка мобільних додатків під ОС Android та iOS. Існує ряд технологій, які дозволяють це зробити, зокрема, Xamarin, React Native, Flutter, Kotlin Multiplatform Mobile (KMM). Найбільш переспективною серед них є KMM, яка має помітні переваги у порівнянні із іншими альтернативними технологіями.

Оскільки версії одного і того ж додатку для Android та iOS часто мають багато спільного, наприклад, бізнес-логіку додатку (зокрема, код, який відповідає за автентифікацію, керування даними, аналітику тощо), то для різних платформ цілком природним є використання загального коду, який реалізує відповідний функціонал.

KMM SDK якраз і дозволяє створити блок універсального коду, а потім спільно використовувати цей код при роботі багатоплатформних мобільних програм. Платформний же код використовується тільки там, де це необхідно — для реалізації інтерфейсу або під час роботи з платформо-залежними API.

Використання KMM хоча і прискорює процес створення мультиплатформенних мобільних додатків за рахунок спільного використання загального коду, однак написання цього коду теж потребує значних витрат часу.

При цьому функціональність загального коду передбачає реалізацію певних етапів, характерних для будь-якого мультиплатформенного мобільного додатку, зокрема, створення бази даних (БД) для збереження інформації на мобільному пристрої, підключення архітектурного шаблону тощо. Автоматизувавши процес створення таких уніфікованих фрагментів коду, можна значно прискорити як процес створення загального коду, так і процес створення мультиплатформенного мобільного додатку загалом.

Тема мультиплатформенної розробки є актуальною, тому що мультиплатформенний підхід до розробки мобільних застосувань дозволяє

створювати додатки для різних платформ з однією кодовою базою, що заощаджує час.

Отже, метою даної магістерської роботи є пришвидшення процесу розробки мультиплатформенного мобільного застосування за рахунок автоматизації створення деяких компонентів його загального коду.

Об’єктом досліджень є процес створення мультиплатформенного мобільного проекту.

Для досягнення мети магістерської роботи були сформовані наступні завдання:

– дослідити процес створення мультиплатформенного мобільного проєкту та наявні засоби його автоматизації;

– розробити метод автоматизованого створення фрагментів мультиплатформенного мобільного проєкту;

– розробити програмне забезпечення для реалізації запропонованого методу;

– провести дослідження ефективності запропонованих рішень.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасний стан розвитку мобільної розробки

Розробка мобільних застосунків - це процес створення ПЗ, які запускаються на мобільному пристрої. Розробка мобільного застосунку включає в себе створення пакетів ПЗ, реалізацію серверних служб, сервіси роботи з БД, тестування застосунку на цільових пристроях тощо [1].

1.1.1 Основні мобільні платформи

На сучасному ринку смартфонів домінують дві платформи. Одна з них - платформа iOS від Apple Inc. Платформа iOS - це ОС, яка керує популярною лінійкою смартфонів Apple iPhone. Друга платформа - це Android від Google. ОС Android використовується не лише пристроями Google, а й багатьма іншими виробниками для створення власних смартфонів та інших смарт-пристроїв [2].

Хоча між цими двома платформами є деяка схожість під час створення мобільних застосунків, розробка під iOS та розробка під Android передбачає використання різних комплектів розробки ПЗ (SDK) і різного ланцюга інструментів розробки [3]. У той час як Apple використовує iOS виключно для власних пристроїв, Google робить Android доступним для інших компаній за умови, що вони відповідають певним вимогам, таким як включення певних програм Google на пристрої, які вони постачають [4].

Отже, розробники можуть створювати програми для сотень мільйонів пристроїв, орієнтуючись на обидві платформи.

1.1.2 Основні підходи до розробки мобільних застосувань

Існує два підходи до розробки мобільних застосувань:

– нативний;

– мультиплатформенний.

Нативні мобільні застосунки – це програми, написані тією ж мовою, що і ПЗ, для якого вони створені [5]. Тобто, мобільний застосунок і платформа написані на одному коді. Наприклад, для розробки нативних мобільних застосунків під iOS, використовують мови програмування Swift та Objective-C, а за допомогою таких мов програмування як Kotlin та Java, розробник створює застосунки під Android. Оскільки програма написана у зазначеному коді (Swift/ Objective-C для iOS або Kotlin/Java для Android), вона підходить лише для конкретного пристрою, тобто, пристрою, який працює на платформі Android або iOS відповідно. Попередньо встановлені програми, такі як повідомлення, календар або фото, є нативними. Дуже важливим недоліком нативного мобільного застосунку є те, що він є несумісним з декількаами мобільними платформами.

Мультиплатформенна мобільна розробка, в свою чергу, – це підхід, який дозволяє створювати мобільні застосунки, сумісні з основними мобільними платформами: Android та iOS [6]. Мультиплатформенна мобільна розробка є альтернативою створення окремих мобільних застосунків під кожну мобільну ОС.

Отже, мультиплатформенна розробка має перевагу перед нативною розробкою, тому що головною задачею мультиплатфомренної мобільної розробки є як раз створення єдиного мобільного застосунку, який одночасно буде працювати на декількох платформах (Android та iOS).

1.2 Мультиплатформенні технології

Розробка мультиплатформенних мобільних застосунків постійно розвивається завдяки появі нових технологій. Мультплатформенні технології дозволяють написати код один раз, а потім перевести цей код у нативний для декількох ОС, що дозволяє створювати мобільний застосунок на різних платформах з мінімальними зусиллями. Мультиплатформенні технології надають можливості повторного використання коду, що значною мірою покращує продуктивність написання коду.

На сьогоднішній день існує ряд передових технологій, які дозволяють розробнику створювати мультиплатформенні мобільні застосунки. Це такі технології як:

– KMM;
– Flutter;
– React Native.

Аналіз цих мультиплатформенних технологій проведемо за наступними критеріями:

– реалізація користувацького інтерфейсу;
– реалізація бізнес-логіки;
– реалізація багатошарової архітектури;
– сумісність мов програмування;
– інтеграція з існуючими Android- та iOS-проєктами.

Реалізація користувацького інтерфейсу. Усі мультиплатформенні технології, що розглядаються, реалізують користувацький інтерфейс по-різному.

Flutter використовує спеціальний спосіб реалізації користувацького інтерфейсу, який використовує інструмент із власного SDK [7] - віджети. Віджети - це будівельні блоки мобільного застосування на Flutter, які потрібно використовувати у своєму мобільному застосуванні. Усе мобільне застосування складається з віджетів. Технологія Flutter поставляється з розширеною підтримкою анімації, малювання та жестів, що дозволяє будь-якому створеному віджету вести себе так, як хоче розробник. Тобто, мультиплатформенний застосунок Flutter буде мати спільний вигляд користувацького інтерфейсу, як для Android, так і для iOS.

React Native, в свою чергу, для реалізації користувальницького інтерфейсу, використовує спеціальні компоненти з бібліотеки React.js [8], які на платформі iOS та Android будуть візуалізовані як нативні елементи відповідної платформи. React.js має багато основних компонентів, від елементів керування формою до

індикаторів активності, але якщо мобільний застосунок вимагає наявності нестандартних компонент, наприклад, поле вводу тексту з різнокольоровим обрамленням, то такі елементи на обох платформах (iOS та Android), будуть виглядати однаково.

KMM реалізує користувальницький інтерфейс шляхом реалізації нативних елементів iOS та Android окремо на відповідній платформі. Тобто, елементи користувальницького інтерфейсу будуть мати стандартний для кожної платформи вигляд.

Реалізація бізнес-логіки. Реалізація бізнес-логіки в усіх технологіях виноситься до загальної частини, різниця полягає лише в її реалізації на відповідних мовах програмування. В технології KMM реалізація бізнес-логіки виконується на мові програмування Kotlin, в Flutter - на мові Dart, а в React Native - на JavaScript.

На конференції «Google I/O 2019» компанія Google заявила, що відтепер, пріорітетною мовою програмування для розробки Android-додатків є Kotlin, тому що ця мова програмування має багато переваг [4]. Оскільки Kotlin схожий на мову для розробки iOS-додатків – Swift, то написання мультиплатформенного коду для Android-розробників та iOS-розробників буде легше на Kotlin, ніж вивчати мову Dart для Flutter, яка в свою чергу, позиціонує себе як заміну JavaScript, знання якого потребує React Native.

Реалізація багатошарової архітектури. Технології Flutter та React Native реалізують багатошарову архітектуру за рахунок ручного відділення користувальницького інтерфейсу від бізнес-логіки.

KMM, в свою чергу, за замовчуванням реалізує відділення користувацького інтерфейсу та бізнес-логіки, тобто робити це вручну немає потреби.

Сумісність мов програмування. З точки зору сумісності, технології Flutter та React Native вимагають використання лише мов програмування Dart та JavaScript відповідно.

Технологія KMM, в свою чергу, допускає використання модулів, написаних на інших мовах програмування, таких як Java та JavaScript.

Інтеграція з існуючими Android- та iOS-проєктами. Технології Flutter та React Native вимагають розробку продукта з їх власною інфраструктурою, тобто, використовуючи відповідні мови програмування, Dart та JavaScript.

Технологія KMM дає можливість інтеграції з існуючими Android- та iOS-проєктами, тому що відповідні нативні проєкти та мультиплатформенні проєкти KMM мають однаковий спосіб реалізації елементів користувацького інтерфесу.

Таким чином, можна зробити висновок, що технологія KMM має ряд переваг перед технологіями Flutter та React Native:

– краще рішення щодо реалізації користувацького інтерфейсу, який на відміну від Flutter та React Native є нативним для відповідної платформи, а отже є дуже швидким та зрозумілим;

– краще рішення щодо реалізації бізнес-логіки за рахунок використання мови програмування Kotlin;

– KMM за замовчуванням реалізує відділення користувацького інтерфейсу від бізнес-логіки;

– KMM на відміну від Flutter та React Native легко інтегрувати з існуючими Android- та iOS-проєктами.

1.3 Мультиплатформенний проєкт

Мультиплатформенний застосунок – це мобільне ПЗ, яке призначено для роботи одразу на декількох мобільних ОС (Android або iOS). Мультиплатформнений застосунок реалізується за рахунок створення мультиплатформенного мобільного проєкту [9].

Мультиплатформенний мобільний проєкт технології KMM має наступну структуру [10]:

– загальний (shared) модуль;

– модуль iOS;

– модуль Android.

В загальному модулі знаходиться спільна бізнес-логіка iOS- та Android-застосунків, тобто загальний модуль і є мультиплатформенним модулем KMM.

Android-модуль являє собою типовий проєкт Android, написаний на мові програмування Kotlin.

Модуль iOS зберігається як проєкт iOS, написаний на мові програмування Swift.

Наявний підхід до розробки мультиплатформенного мобільного застосунку з використанням технології KMM передбачає винесення типових для декількох платформ дій, як правило, бізнес-логіки, у єдиний універсальний блок коду – загальний код, який в подальшому спільно використовуватиметься в одних частинах програми, залишаючи інші частини (наприклад, UI-шар) повністю нативними. Загальний код реалізується на мові Kotlin і міститься у загальному модулі (shared-модулі) KMM-проєкту.

Спільна бізнес-логіка мультиплатформенного мобільного застосунку може включати сервіси для роботи з БД, сервіси для роботи з мережею, моделі даних. Також до неї можуть входити архітектурні компоненти програми, які безпосередньо не включають UI, але з ним взаємодіють (ViewModel, Presenter тощо).

Більшість мобільних застосунків локально зберігають дані на клієнтських пристроях в БД SQLite - спеціалізованій, оптимізованій під мобільні платформи програмній бібліотеці, яка реалізує систему управління реляційною БД [11]. Тому реалізацію сервісів роботи з БД можна вважати типовою функціональністю при розробці мультиплатформенних мобільних застосунків.

Створення відповідної БД потребує виконання рутинних дій по реалізації сутностей предметної області, стандартних операцій обробки таблиць БД - CRUD-операцій (Create Read, Update, Delete) [12], інтерфейсу взаємодії застосунку з БД -

DAO-класів (Data Access Object), класу управління БД. Сутності БД і CRUD-операції реалізуються на мові SQL, DAO-класи і клас управління БД - на мові Kotlin.

Окрім того, в рамках shared-модуля можна реалізувати механізм впровадження залежностей (Dependency Injection, DI). Це дасть змогу створювати і обробляти залежності (об'єкти інших класів, які можна використовувати як сервіс) поза залежними класами і, як наслідок, спростить модифікацію класів.

З розвитком KMM стали з'являтися мультиплатформенні фреймворки, зокрема, SQLDelight – фреймворк для роботи з SQLite, Koin [13] – DI-фреймворк для впровадження залежностей. Однак їх функціональність не дозволяє в повній мірі автоматизувати ні процес створення shared-модуля, ні окремих його складових.

Зокрема, фреймворк SQLDelight підтримує лише генерацію класів даних (data-класів) для сутностей БД на мові програмування Kotlin і автоматично реалізує взаємодію мобільного застосунку з БД [14].

Левова ж частина роботи по створенню загального коду (shared-модуля) все ще виконується в ручному режимі.

KMM SDK якраз і дозволяє оптимізувати розробку мультиплатформенного мобільного застосунку шляхом одноразового написання блоку універсального коду, а потім спільного використання цього коду на різних платформах. Платформний же код використовується тільки там, де це необхідно — для реалізації інтерфейсу або під час роботи з платформозалежними API.

Нижче приведена схема мультиплатформенного мобільного проєкту (рис. 1.1).

Рисунок 1.1 – Схема мультиплафторменного мобільного проєкту

Оскільки, типовим функціоналом при розробці мобільних застосунків є реалізація сервісів роботи з БД, актуальною задачею є автоматизація саме цих процесів. При цьому інші сервіси передбачається реалізовувати в ручному режимі.

З урахуванням зазначеного, програмні компоненти загального коду, які є реалізацією сервісів роботи з БД, назвемо базовим кодом, а частину shared-модуля, що містить базовий код - первинним мультиплатформенним мобільним проєктом (рис. 1.2).

Фактично, процес створення shared-модуля мультиплатформенного мобільного застосунку пропонується здійснювати у два етапи: генерація базового коду (первинного проєкту) та подальша реалізація структурних компонент вручну.

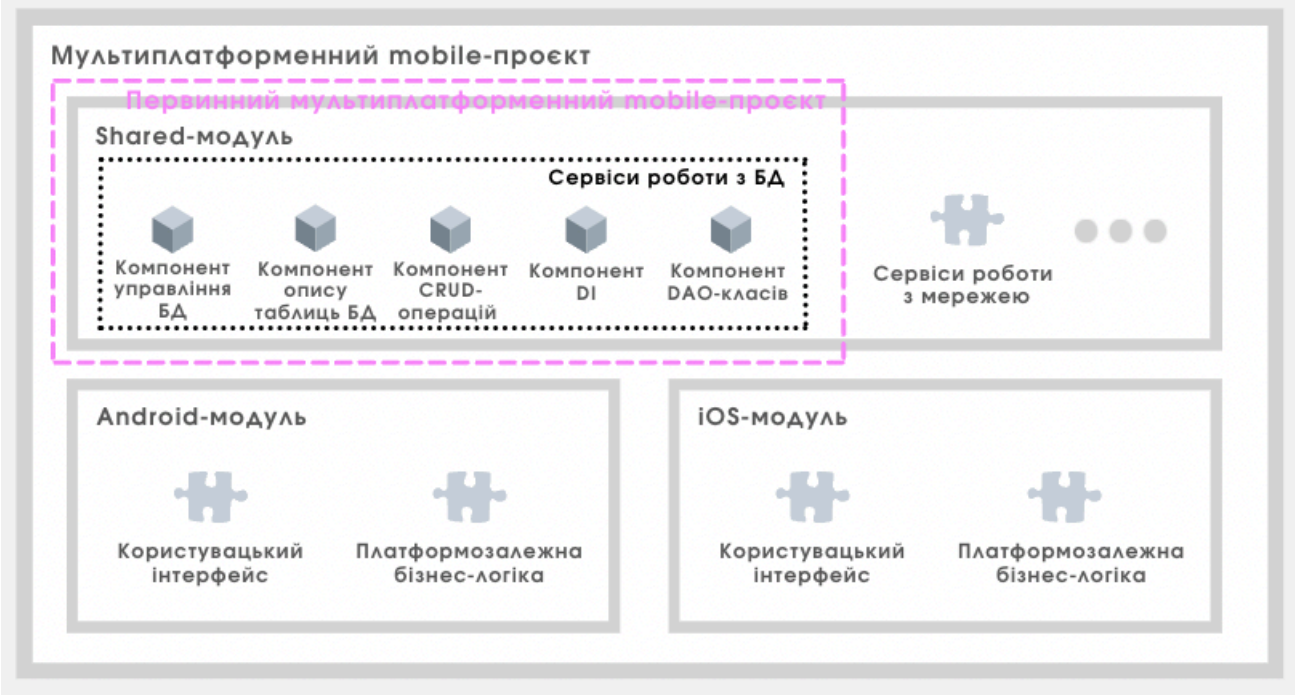


Рисунок 1.2 – Схема первинного мультиплафторменного мобільного проєкту

1.4 Постановка задач

Проведений огляд предметної області показав, що типовим функціоналом при розробці мультиплатформенного мобільного застосунку є реалізація сервісів роботи з БД, яка служить для зберігання даних на клієнтських пристроях. Реалізація даних сервісів потребує виконання значної кількості рутинних дій в ручному режимі. Тому автоматизація створення програмних компонент для реалізації відповідних сервісів в рамках так званого первинного мультиплатформенного мобільного проєкту, під яким розуміється частина shared-модуля, що містить код відповідних компонент (базовий код), є достатньо актуальною задачею.

Для вирішення даної проблеми були сформовані наступні задачі досліджень:

– розробити метод автоматизованого створення первинного мультиплатформенного мобільного проєкту;

– розробити програмне забезпечення для реалізації запропонованого методу;

– провести дослідження ефективності запропонованих рішень.

1.5 Висновки по розділу

Проведений огляд предметної області показав, що технологія KMM для створення мультиплатформенного мобільного застосунку має значні переваги у порівнянні із іншими сучасними технологіями, зокрема, Flutter та React Native.

Оскільки, типовим функціоналом при розробці мультиплатформенного мобільного застосунку є реалізація сервісів роботи з БД, що потребує виконання значної кількості рутинних дій в ручному режимі, то актуальною є задача автоматизації створення програмних компонент для реалізації відповідних сервісів в рамках так званого первинного мультиплатформенного мобільного проєкту, під яким розуміється частина shared-модуля, що містить код відповідних компонент (базовий код).

Процес створення shared-модуля мультиплатформенного мобільного проєкту пропонується здійснювати у два етапи: генерація первинного мультиплатформенного проєкту та подальша реалізація структурних компонент вручну.

2 АВТОМАТИЗАЦІЯ СТВОРЕННЯ ПЕРВИННОГО МУЛЬТИПЛАТФОРМЕННОГО МОБІЛЬНОГО ПРОЄКТУ

2.1 Наявний підхід до створення первинного мультиплатформенного мобільного проєкту

Як було зазначено раніше, для реалізації первинного мультиплатформенного мобільного проєкту потрібно вручну реалізувати усі його компоненти, а саме:

- компонент управління БД;
- компонент опису таблиць БД;
- компонент CRUD-операцій (для кожної сутності БД);
- компонент DAO-класів (для кожної сутності БД);
- компонент DI-класів.

У якості прикладу, будемо використовувати тестову схему БД (рис 2.1) для зберігання інформації про книги, видавництво та жанри літератури.

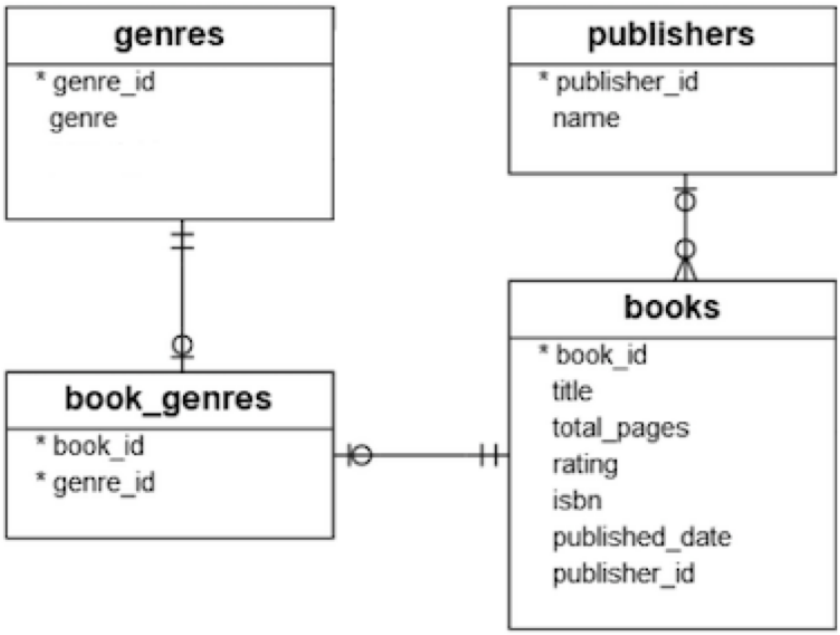


Рисунок 2.1 – Тестова схема БД

Наведемо опис кожного зазначеного вище компоненту shared-модуля первинного мультиплатформенного мобільного проєкту.

Компонент управління БД. Більшість мобільних додатків зберігають дані локально на пристрої. Дані, зазвичай, зберігаються в БД SQLite. Для реалізації компоненту управління БД в технології KMM використовується бібліотека SQLDelight.

Тобто, даний компонент буде містити клас управління БД мультиплатформенного мобільного застосування (рис 2.2).

```
class ApplicationDatabase(sqlDriver: SqlDriver) {

private val database: AppDatabase = AppDatabase(sqlDriver)

    val bookQueries: BookQueries = database.bookQueries
    val bookGenresQueries: BookGenresQueries = database.bookGenresQueries
    val publisherQueries: PublisherQueries = database.publisherQueries
    val genreQueries: GenreQueries = database.genreQueries
}
```

Рисунок 2.2 – Клас управління БД

Компонент опису таблиць БД. Даний компонент буде містити файли опису таблиць БД на SQL-мові.

Компонент опису таблиць БД зберігається у виглядів файлів .sq [15]. Таблиці БД описані на SQL-мові (рис 2.3).

```
CREATE TABLE BookGenres (
    id                  INTEGER PRIMARY KEY,
    book_id             INTEGER,
    genre_id            INTEGER,

    CONSTRAINT bookgenres_book_fk
    FOREIGN KEY (book_id) REFERENCES Book(id),

    CONSTRAINT bookgenres_genre_fk
    FOREIGN KEY (genre_id) REFERENCES Genre(id)
);
```

Рисунок 2.3 – Приклад сриптів створення таблиці BookGenres

Компонент CRUD-операцій. Компонент CRUD-операцій повинен забезпечувати усі сутності БД чотирма основними функціями, а саме, повинна бути реалізована можливість створювати, читати, оновлювати та видаляти дані для кожної таблиці БД.

Компонент CRUD-операцій зберігається у виглядів файлів .sq. CRUD-операції описані на SQL-мові (рис 2.4).

```
/**
 * ----------------------> Select methods <----------------------
 */
selectAllBookGenress:
SELECT * FROM BookGenres;

selectBookGenresById:
SELECT * FROM BookGenres
WHERE id = ?;

/**
 * ----------------------> Insert methods <----------------------
 */
insertBookGenres:
INSERT INTO BookGenres(id, book_id, genre_id)
VALUES( id: ?,  book_id: ?,  genre_id: ?);

/**
 * ----------------------> Update methods <----------------------
 */
updateBookGenres:
UPDATE BookGenres
SET book_id = ?, genre_id = ?
WHERE id = ?;

/**
 * ----------------------> Delete methods <----------------------
 */
deleteAllBookGenress:
DELETE FROM BookGenres;

deleteBookGenresById:
DELETE FROM BookGenres
WHERE id = ?;
```

Рисунок 2.4 – Приклад скриптів CRUD-операцій для таблиці BookGenres

Компонент DAO-класів. Компонент DAO-класів повинен реалізовувати інтерфейси взаємодії с кожною таблицею БД на мові програмування Kotlin. Цей компонент є прошарком між БД та мобільним застосуванням, DAO-інтерфейс абстрагує сутності БД, визначає загальні методи використання з'єднання з відповідною таблицею БД.

Для функціонування DAO (рис 2.5) набір SQL-запитів мінімально повинен мати усі стандартні CRUD-операції.

```
class BookGenresDao(sqlDriver: SqlDriver) {

    private val bookGenresQueries: BookGenresQueries =
        AppDatabase(sqlDriver).bookGenresQueries

    fun selectAll():List<BookGenres> =
        bookGenresQueries.selectAllBookGenress().executeAsList()

    fun selectBookGenresById(id: Long):BookGenres =
        bookGenresQueries.selectBookGenresById(id).executeAsOne()

    fun insert(bookGenres: BookGenres) {
        bookGenresQueries.insertBookGenres(
            id = bookGenres.id,
                        book_id = bookGenres.book_id,
                        genre_id = bookGenres.genre_id
        )
    }

    fun update(bookGenres: BookGenres) {
        bookGenresQueries.updateBookGenres(
            id = bookGenres.id,
                        book_id = bookGenres.book_id,
                        genre_id = bookGenres.genre_id
        )
    }

    fun deleteAll() {
        bookGenresQueries.deleteAllBookGenress()
    }

    fun deleteBookGenresById(id: Long) {
        bookGenresQueries.deleteBookGenresById(id)
    }
}
```

Рисунок 2.5 – Приклад DAO для BookGenres

Компонент DI-класів. Реалізація компонента DI (рис 2.6) є невід'ємною частиною у створенні первинного мультиплатформенного мобільного проєкту [16]. Введення залежностей - це метод програмування, який робить клас незалежним від його залежностей. Створення об'єктів безпосередньо всередині класу є негнучким, оскільки воно фіксує клас окремим об'єктам і унеможливлює зміну екземпляра пізніше незалежно від класу. В KMM для цього існує спеціальна бібліотека – Koin [17].

```
val databaseModule = module {
    single { ApplicationDatabase(get()) }
    single { get<ApplicationDatabase>().bookQueries }
    single { get<ApplicationDatabase>().genreQueries }
    single { get<ApplicationDatabase>().bookGenresQueries }
    single { get<ApplicationDatabase>().publisherQueries }
}

fun initKoin(sqlDriver: SqlDriver) {
    startKoin {
        val platformModule = module {
            single { sqlDriver }
        }

        // use Koin logger
        printLogger()
        // declare modules
        modules(databaseModule, platformModule)
    }
}

val koin: Koin
    get() = KoinPlatformTools.defaultContext().get()
```

Рисунок 2.6 – Реалізація компоненту DI-класів

2.2 Метод автоматизованого створення первинного мультиплатформенного мобільного проєкту

Як зазначалося вище, реалізація сервісів роботи з БД передбачає опис сутностей БД та налаштувань (DAO, CRUD, DI) на мові Kotlin та SQL.

Автоматизувати процес генерації відповідних програмних компонент пропонується шляхом транспіляції описів сутностей БД та налаштувань (вхідних описів), представлених на спеціально розробленій предметно-орієнтованій мові, у коди програмних компонент на мові Kotlin та SQL (базовий код).

Процес транспіляції описів на вхідній предметно-орієнтованій мові у базовий код можна поділити на наступні етапи:

– опис вхідних даних (сутностей БД та налаштувань) для створення первинного мультиплатформенного мобільного проєкту на предметно-орієнтованій мові;

– лексичний аналіз вхідного тексту;

– синтаксичний аналіз вхідного тексту і створення дерева синтаксичного розбоу;

– обхід дерева синтаксичного розбору та збереження даних у тимчасові об’єкти (сутності БД та налаштувань DAO, CRUD, DI);

– валідація тимчасових об’єктів;

– генерація мультиплатформенного коду.

Розглянемо більш детально кожен з цих етапів.

Опис вхідних даних на предметно-орієнтованій мові. Оскільки, задача представлення вхідних даних для створення первинного мультиплатформенного мобільного проєкту потребує найбільш спрощену та наочну форму опису усіх необхідних його елементів, то для розв’язання цієї задачі доцільно розробити спеціальну предметно-орієнтовану мову, яка має певну схожість з офіційною мовою розробки мобільних застосувань під Android – Kotlin, а тому не є важкою у

вивченні розробниками [18]. Також, розроблена предметно-орієнтована мова реалізує рішення для конкретного виду предметної області, а отже абстрагуючи проблему, спрощує опис вхідних даних для створення первинного мультиплатфомренного мобільного проєкту.

Вхідними даними для створення первинного мультиплатформенного первинного мобільного проєкту є опис на розробленій предметно-орієнтованій мові сутностей БД та налаштувань (DAO, DI, CRUD).

Лексичний аналіз вхідного тексту. Лексичний аналіз - це перетворення послідовності символів вхідного тексту (лексем) в послідовність токенів, що представляють собою послідовність символів, яка розглядається як одиниця в граматиці предметно-орієнтованої мови, та визначення типів цих токенів [19].

Токенами для даної проблемно-орієнтованої мови є нескінченна кількість її понять, оскільки задача опису сутностей БД та їх полів не може бути вирішена кінцевою їх кількістю.

Після зчитування токени повинні бути збережені для подальшого їх використання у вигляді спискової структури.

Синтаксичний аналіз вхідного тексту. Синтаксичний аналіз - це процес аналізу вхідної послідовності символів з метою розбору її граматичної структури згідно із формальною граматикою предметно-орієнтованої мови [20]. Результатом синтаксичного аналізу є побудова на основі правил граматики предметно-орієнтованої мови дерева синтаксичного розбору (AST) [21].

В рамках даного методу для вирішення задач синтаксичного аналізу пропонується використовувати допоміжні фреймворки, зокрема, фреймворк ANLR.

Принципи побудови AST грунтуються на використанні LL-граматики, як різновиду контекстно-вільної граматики, оскільки вхідний текст може містити описи нескінченної кількості сутностей і їх характеристик, що призводить до можливості аналізу нескінченної кількості токенів.

Створення AST починається з кореневого вузла (початковий символ граматики) у відповідності із правилами граматики проблемно-орієнтованої мови.

Правила створення AST фреймворком ANTLR передбачають використання алгоритму рекурсивного спуску, який будується на основі рекурсивних перетворень, що виконуються для кожного зчитаного токену [22].

Приклад дерева AST наведено на рисунку 2.9.

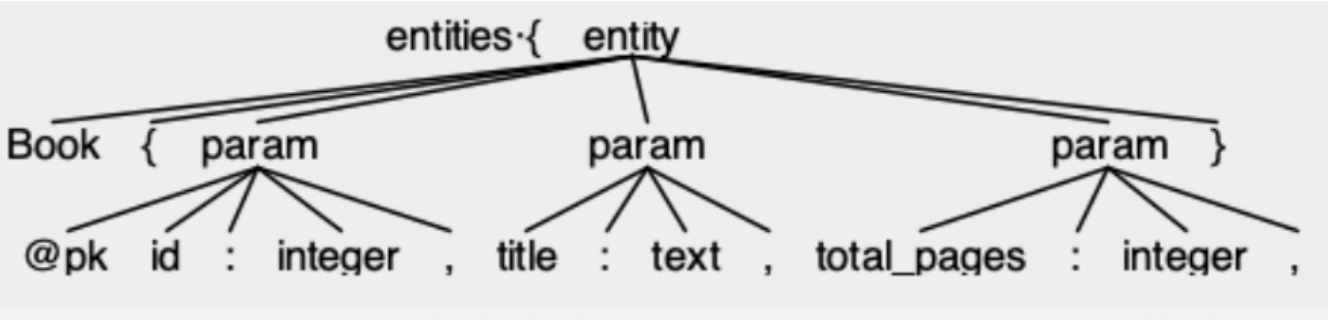


Рисунок 2.7 – Фрагмент дерева синтаксичного розбору

Обхід синтаксичного дерева та збереження даних у тимчасові об'єкти. В залежності від задач, використовуються різні способи обходу AST:

- прямий обхід (NLR);
- центрований обхід (LNR);
- зворотний обхід (LRN).

Дерево загального вигляду. Однак, в рамках даного методу для вирішення задач синтаксичного аналізу використовується фреймворк ANLR, правила якого передбачають використання NLR-обходу AST, то саме цей вид обходу реалізується при обробці дерева синтаксичного розбору.

Результатом обходу синтаксичного дерева є створення тимчасових об'єктів сутностей БД та налаштувань DAO, CRUD, DI. Тимчасові об'єкти представлені у вигляді спеціально створених класів на мові програмування Kotlin.

Валідація тимчасових об'єктів. Валідація даних означає оцінку якості вхідних даних перед генерацією мультиплатформенного коду [25]. Метою перевірки даних є запобігання помилкам у створених тимчасових об'єктах.

Валідація тимчасових об'єктів виконується на основі попередньо сформованих вимог до валідації, реалізованих у вигляді методів спеціально створеного класу Validator.

Генерація мультиплатформенного коду. Способом генерації коду є спеціально створений шаблонний движок, який використовує шаблони, що представляють собою рядки на мові програмування Kotlin. Такі шаблони створені для кожної сутності БД, CRUD-операцій, DAO-класів, класу управління БД, DI-класів у вигляді Kotlin- та SQL-коду. Шаблони містять спеціальні позначення, які інтерпретуються шаблонним движком. Генерація базового мультиплатформенного коду виконується шляхом вставки даних з тимчасових об'єктів у відповідні шаблони.

Вхідними даними для шаблонного движка є тимчасові об’єкти, що містять інформацію про сутності БД та налаштування DAO, CRUD та DI.

Результатом роботи шаблонного движка є базовий мультиплатформенний SQL- та Koltin-код: компонент управління БД, компоненти DAO-класів та DI-класів представлені у вигляді Kotlin-коду; компоненти опису таблиць БД та CRUD-операцій - у вигляді SQL-коду.

Загалом, метод автоматизованого створення первинного мультиплатформенного мобільного проєкту представимо наступною послідовністю кроків:

Крок 1. Опис вхідних даних на предметно-орієнтованій мові (сутностей БД та налаштувань DAO, CRUD, DI).

Крок 2. Лексичний аналіз вхідного тексту. Формування списку зчитаних токенів на основі правил визначення їх типів.

Крок 3. Низхідний синтаксичний аналіз списку зчитаних токенів і формування синтаксичного дерева (AST) на основі правил граматики проблемно-орієнтованої мови.

Крок 4. Обхід синтаксичного дерева та збереження даних у тимчасові об'єкти (для сутностей БД та налаштувань DAO, CRUD, DI), які реалізовані у вигляді класів.

Крок 5. Валідація тимчасових об'єктів на основі попередньо сформованих вимог до валідації з використання методів класу Validator.

Крок 6. Генерація мультиплатформенного коду на основі спеціально створених шаблонів шляхом вставки даних з тимчасових об'єктів у відповідні шаблони коду.

Схему транспіляції вхідних описів сутностей БД та налаштувань в базовий загальний код первинного мультиплатформенного мобільного проєкту можна представити наступним чином (рис. 2.10).

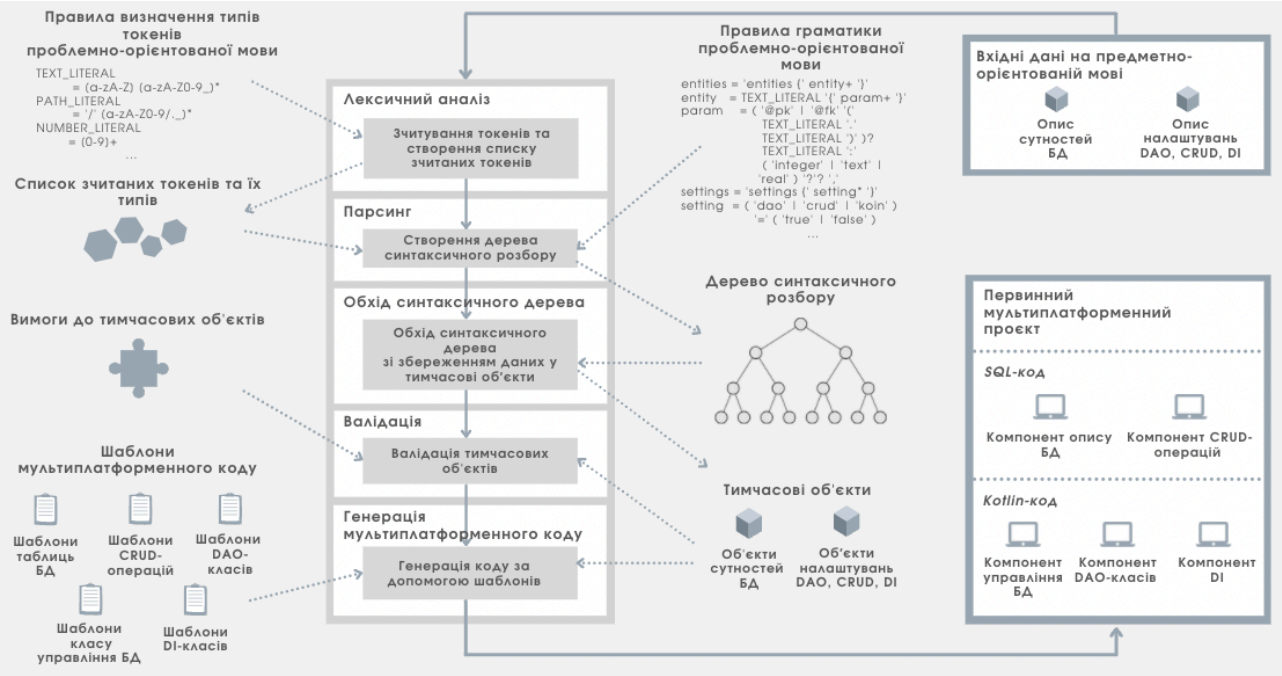


Рисунок 2.8 – Схема транспіляції вхідних описів у базовий мультиплатформенний код

Запропонований метод пришвидшує як процес створення базового загального коду, так і загалом процес створення мультиплатформенного мобільного застосунку.

2.3 Проблемно-орієнтована мова для опису вхідних даних для створення первинного мультиплафторменного мобільного проєкту

Для вирішення задачі опису вхідних даних, потрібних для створення первинного мультиплатформенного мобільного проєкту, описаного в розділі 2.2, розроблено спеціальну предметно-орієнтовану мову.

Оскільки вхідні описи сутностей БД, налаштувань DAO, CRUD, DI та їх характеристик можуть містити нескінченну кількість назв, то граматику мови пропонується реалізувати у вигляді LL-граматики, як різновиду контекстно-вільної граматики. Дану граматику можна представити наступним чином:

$$G = \{T, N, P, S\} \quad (2)$$

T – термінальний словник;

N – нетермінальний словник;

P – множина правил граматики;

S – початковий символ граматики.

Термінальний словник розробленої проблемно-орієнтованої мови (параметр T) складається із підмножини таблиці символів ASCII, в яку входять великі та малі латинськи символи, цифри та деякі управляючі символи (табл. 2.1).

Таблиця 2.1 – Термінальний словник предметно-орієнтованої мови для автоматизованого створення первинного мультиплатформенного мобільного проєкту

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2								‘	(	)			,			
3	0	1	2	3	4	5	6	7	8	9	:	;	<			?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z					_
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}		

Множиною нетермінальних символів (параметр N) виступають конструкції опису сутностей БД та ключові слова:

$$N = \{ \text{TEXT\_LITERAL}, \text{NUMBER\_LITERAL}, \text{PATH\_LITERAL}, \text{WS\_LITERAL} \}. \quad (4)$$

TEXT_LITERAL = [a-zA-Z] [a-zA-Z0-9_]*

PATH_LITERAL = '/' [a-zA-Z0-9/._]*

NUMBER_LITERAL = [0-9]+

WS_LITERAL = [#xa#x9]+

Ключові слова - description, entities, settings, integer, text, real, dao, crud, koin, true, database, databasePakage, pathes, path, false, @pk, @fk.

Параметром S, тобто початковим символом граматики, є тег description, який визначає опис сутностей та налаштувань, потрібних для створення DAO-класів, CRUD-операцій та DI-класів.

Нетермінали даної мови виводяться на основі розробленої множини правил граматики (параметр P), представлених у розширеній нотації Бекуса-Наура:

description = 'description {' entities settings pathes '}'

entities = 'entities {' entity+ '}'

entity = TEXT_LITERAL '{' param+ '}'

param = ('@pk' | '@fk' '(' TEXT_LITERAL '.' TEXT_LITERAL ')')? TEXT_LITERAL ':' ('integer' | 'text' | 'real') '?'? ','

settings = 'settings {' setting* '}'

setting = ('dao' | 'crud' | 'koin') '=' ('true' | 'false')

pathes = 'pathes {' path* '}'

path = ('dao' | 'crud' | 'koin' | 'database' | 'databasePackage') '=' PATH_LITERAL

Описи усіх нетерміналів представлені з використанням діаграм Вірта.

Структура тіла тега description передбачає наявність опису сутностей БД в тегу entities, налаштувань для DAO, CRUD та DI в тегу settings та шляхів до

директорій в тегу pathes, в які необхідно згенерувати файли базового мультиплатформенного коду (рис. 2.11).

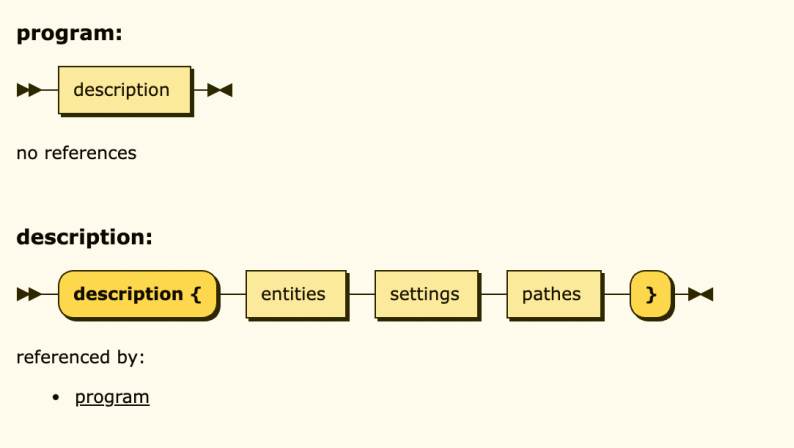


Рисунок 2.9 – Опис description елементу

В свою чергу, тег entities, повинен містити список з однією або більше entity (конкретна сутність БД). Тег entity складається з текстового літералу, який визначає назву сутності та зі списку елементів param (поля сутності БД) (рис 2.12).

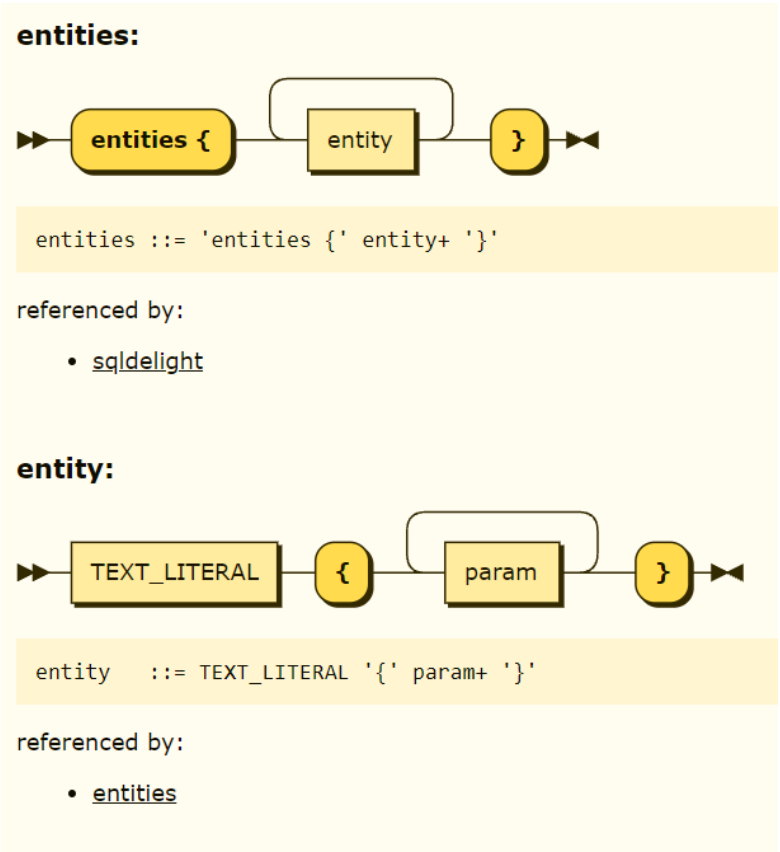


Рисунок 2.10 – Опис елементів entities та entity

Елемент param містить аналог анотації з Kotlin. Анотація повинна бути у вигляді “@pk”, якщо це поле є первинним ключем (первинний ключ – це обмеження, що дозволяє однозначно ідентифікувати запис в SQL-таблиці БД), а анотація “@fk” повинна бути присутня, якщо поле є зовнішнім ключем (зовнішні ключі дозволяють встановити зв'язок між SQL-таблицями БД) та повинна містити посилання на поле сутності, на яку вона посилається. Далі повинен йти текстовий літерал у якості ім’я поля, через двокрапку повинен бути визначений його тип (integer, text або real), потім знак питання, який визначає, чи є параметр обов’язковим. Вираз закінчується комою (рис. 2.13).

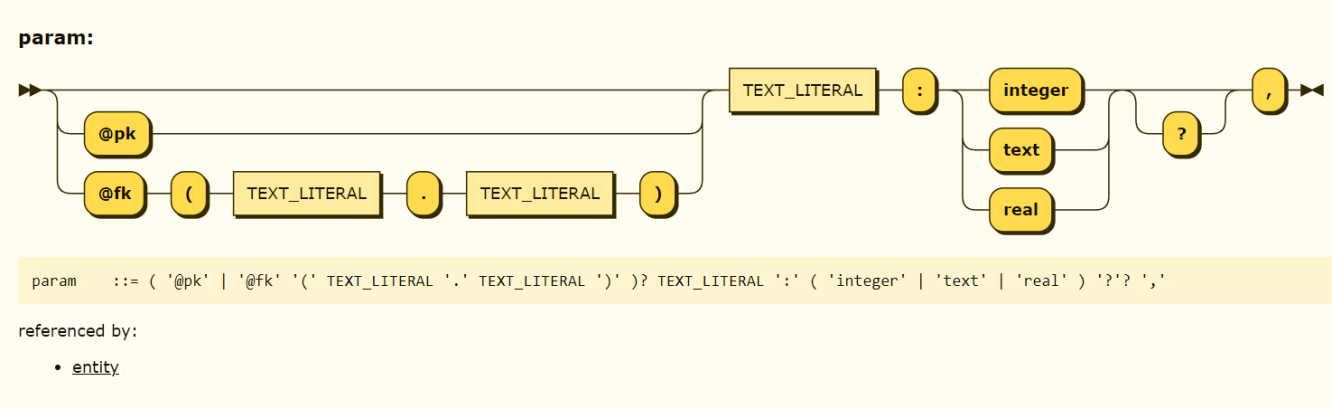


Рисунок 2.11 – Опис param елементу

Елемент settings повинен містити список елементів setting (визначає конкретне налаштування CRUD, DI або DAO). Тег setting має вигляд «параметр» = «булевське значення», тобто, визначає, потрібне це налаштування, чи ні (рис. 2.14).

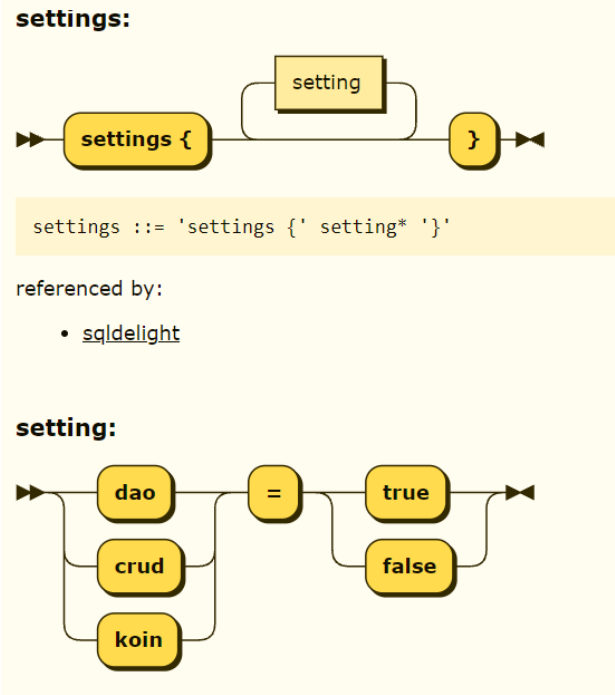


Рисунок 2.12 – Опис для setting та settings елементів

Елемент pathes повинен містити список елементів path (шлях до необхідної директорії). Тег path має вигляд «параметр» = «шлях», тобто, визначає шлях до відповідної директорії (рис. 2.14).

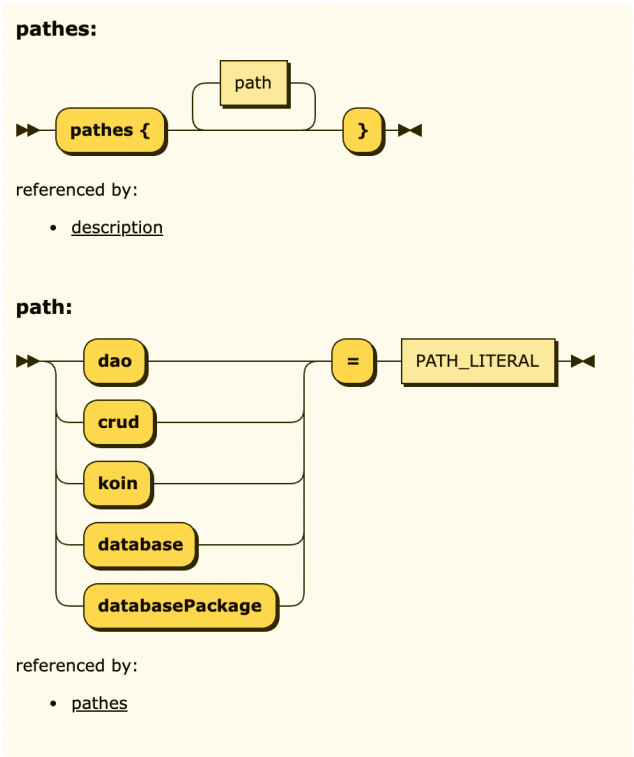


Рисунок 2.13 – Опис для path та pathes елементів

Текстовий літерал може складатися з букв верхнього та нижнього регістру, цифр та спеціального символу «_» та «/». Першим символом текстового літералу повинна бути буква в верхньому або нижньому регістрі (рис. 2.16).

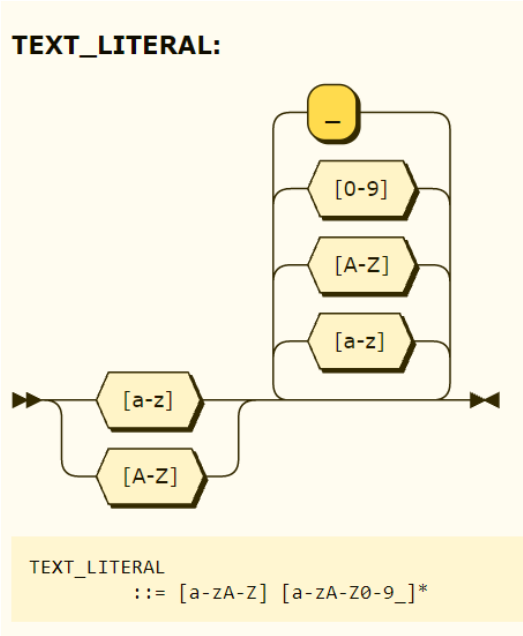


Рисунок 2.14 – Опис для text_literal елементу

Чисельний літерал (рис. 2.17) складається як мінімум з однієї цифри від 0 до 9.

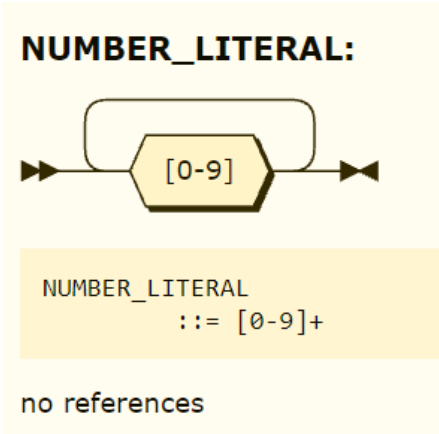


Рисунок 2.15 – Опис для number_literal елементу

Нижче наведений приклад опису вхідних даних методу автоматизованого створення первинного мультиплатформенного мобільного проєкту на розробленій предметно-орієнтованій мові для тестової предметної області.

```
description {
    entities {
        Book {
            @pk
            id                  : integer,
            title               : text,
            total_pages         : integer,
            rating              : real,
            @fk(Publisher.id)
            publisher_id        : integer,
            @fk(Author.id)
            author_id           : integer,
        }
        BookGenres {
            @pk
            id                  : integer,
            @fk(Book.id)
            book_id             : integer,
            @fk(Genre.id)
            genre_id            : integer,
        }
        Publisher {
            @pk
            id                  : integer,
            name                : text,
        }
        Genre {
            @pk
            id                  : integer,
            name                : text,
        }
        Author {
            @pk
            id                  : integer,
            name                : text,
        }
    }
    settings {
        dao = true
        crud = true
        koin = true
    }
    pathes {
        databasePackage = /com.diploma.database.AppDatabase
        dao = /Users/eddyshvets
        crud = /Users/eddyshvets
        koin = /Users/eddyshvets
        database = /Users/eddyshvets
    }
}
```

2.4 Висновки по розділу

Проведено аналіз наявного процесу створення первинного мультиплатформенного мобільного проєкту, який показав, що його розробка в ручному режимі значної кількості рутинних дій.

Запропоновано метод автоматизованого створення первинного мультиплатформенного мобільного проєкту, який передбачає транспіляцію вхідних описів сутностей БД та налаштувань DAO, CRUD, DI у SQL- та Kotlin-компоненти загального коду мультиплатформенного мобільного проєкту.

Розроблена проблемно-орієнтована мова для опису сутностей БД та налаштувань DAO, CRUD, DI, заснована на LL-граматиці, як різновиді контекстно-вільної граматики, яка має певну схожість з офіційною мовою розробки мобільних застосувань під Android – Kotlin, а тому не є важкою у вивченні розробниками.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ ПЕРВИННОГО МУЛЬТИПЛАТФОРМЕННОГО МОБІЛЬНОГО ПРОЄКТУ

Розробка мультиплатформенних застосунків за допомогою технології KMM ведеться у спеціалізованій IDE – Android Studio, яка є офіційним інтегрованим середовище (IDE) для розробки додатків Android на основі IntelliJ IDEA. Android Studio імплементує принципи SOLID, зокрема, принцип відкритості/закритості, тому є відкритою для розширення свого функціонала, але закритою для його модифікацій [26].

Зручним рішенням, що допомагає гнучко підлаштовувати роботу IDE під свої потреби є плагіни [27]. JetBrains дозволяє розширювати свою IDE, написавши власний плагін.

Враховуючи, що розробка мультиплатформенних застосувань ведеться у Android Studio, а способом розширення її функціоналу є плагіни, доцільною є реалізація ПЗ для автоматизованого створення первинного мультиплатформенного мобільного проєкту саме у вигляді плагіну. Оскільки, Android Studio є модифікованим варіантом IDE IntelliJ IDEA від компанії JetBrains, то плагін пропонується реалізувати саме у цій IDE [28].

Як правило, розробка плагінів ведеться на мові програмування Kotlin або Java. Оскільки мова програмування Kotlin має ряд переваг перед Java [29] та розробка мобільних застосунків ведеться саме на Kotlin, то було вирішено розробити плагін саме на мові програмування Kotlin.

3.1 Архітектура плагіну

Як було зазначено вище, плагін, що розробляється, є розширенням Android Studio. Архітектура будь-якого плагіну має наступні особливості:

– кожен плагін має маніфест (plugin.xml) [31], який визначає вміст плагіну і спосіб його підключення до Android Studio;

– кожен плагін отримує свій власний завантажувач класів, а Android Studio використовує механізм PicoContainer для керування завантаженням усіх своїх класів, інтерфейсів та об’єктів (DI), які плагін надає Android Studio [32].

Android Studio повністю керує життєвим циклом плагіна і робить це за допомогою компонентів PicoContainer.

Реалізація плагіну потребує надання Iteliji IDEA інформації про його функціональність, щоб IDE могла належним чином завантажити даний плагін та дозволити користувачам взаємодіяти з ним, зокрема, за допомогою комбінацій клавіш, меню, панелей інструментів тощо. Для коректного відображення плагіну IDE потрібно також надати інформацію стосовно назви плагіну, асоційованих значків, інтернаціоналізованих пакетів рядків тощо. Вся ця декларативна інформація про плагін зберігається у plugin.xml, який є точкою входу в код плагіну.

Одним з основних способів взаємодії користувачів з плагіном є «дії» [33]. Дії можна викликати, скориставшись пошуком у будь-якому місці коду (Shift + Shift) і ввівши назву дії; натиснувши відповідну комбінацію клавіш, щоб викликати цю дію; клацнувши на панелі інструментів, що містить дію тощо.

Усі дії, які визначає плагін, повинні бути явно перераховані в plugin.xml файлі (рис. 3.2).

```
<idea-plugin>
    <id>org.example.PluginEdsl</id>
    <name>EDSL Plugin</name>
    <vendor email="eddy.shvets@gmail.com" url="@EddyShvets">Eddy Shvets</vendor>

    <description><![CDATA[
    Enter short description for your plugin here.<br>
    <em>most HTML tags may be used</em>
    ]]></description>

    <depends>com.intellij.modules.platform</depends>

    <extensions defaultExtensionNs="com.intellij">
    </extensions>

    <actions>
        <action id="com.custom.action.CustomAction"
                class="MethodAction"
                icon="AllIcons.Actions.Colors"
                text="Generate Multiplatform Code">
            <add-to-group group-id="RunContextGroup" anchor="last"/>
        </action>
    </actions>
</idea-plugin>
```

Рисунок 3.1 – Файл plugin.xml

Для прискорення реалізації плагіну, був використаний спеціалізований фреймворк ANTLR, який є потужним генератором парсерів. ANTLR широко використовується в наукових колах для задач створення різноманітних інструментів, формальних мов тощо.

Плагін для підтримки процесу автоматизованого створення первинного мультиплатформенного мобільного проєкту (EDSL) має наступні структурні складові (рис. 3.1):

– лексичний аналізатор предметно-орієнтованої мови;
– синтаксичний аналізатор предметно-орієнтованої мови;
– візітор дерева синтаксичного розбору;
– валідатор тимчасових об’єктів;

– генератор мультиплатформенного коду.

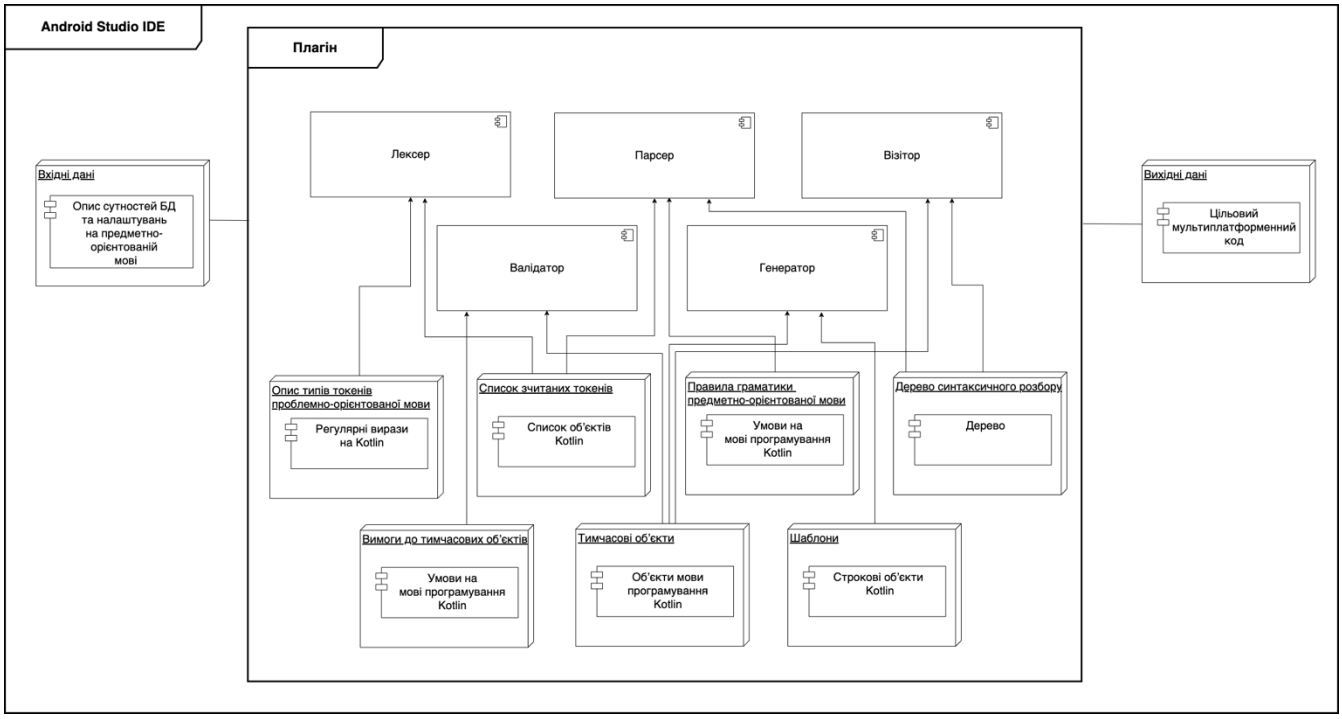


Рисунок 3.2 – Архітектура плагіну EDSL

3.2 Реалізація компонент плагіну

Кожний визначений компонент плагіну у пункті 3.2 має певні особливості реалізації, тому розглянемо їх більш детально.

Лексичний аналізатор предметно-орієнтованої мови. Реалізовно на основі класу EDSLLexer, згенерованого з використанням фреймворка ANTLR (рис. 3.3).

Клас EDSLLexer служить для реалізації лексичного аналізу вхідних описів сутностей БД та налаштувань DAO, CRUD та DI на розробленій предметно-орієнтованій мові. Лексичний аналізатор використовує опис типів токенів предметно-орієнтованої мови, який реалізовано у вигляді регулярних виразів на мові програмування Kotlin, для формування списку зчитаних токенів.

Помилки, що виникають в процесі лексичного аналізу плагін відображає у спеціальному діалоговому вікні.

Рисунок 3.3 – UML-діаграма класів лексичного аналізатору

Синтаксичний аналізатор предметно-орієнтованої мови. Реалізовно на основі класів EDSLParser, створеного за допомогою фреймворку ANTLR та допоміжних класів, які описують правила граматики розробленої предметно-орієнтованої мови (рис. 3.4).

Рисунок 3.4 – UML-діаграма класів синтаксичного аналізатору предметно-орієнтованої мови

Візітор дерева синтаксичного розбору. Реалізовано на основі класів EDSLVisitor, EDSLBaseVisitor, EDSLListener, EntityCreator (рис. 3.5). Візітор здійснює обхід дерева, використовуючи два типових підхода: Visitor та Listener. У Visitor для обробки нащадків будь-якого вузла необхідно викликати вручну методи їх обходу. У Listener методи відвідування всіх нащадків викликаються автоматично. У Listener існує метод, який викликається на початку відвідування вузла (enterNode) і метод, який викликається після відвідування вузла (exitNode). Реалізацією цих підходів є EDSLVisitor, EDSLBaseVisitor та EDSLListener відповідно. Зокрема, клас EntityCreator реалізує збереження даних у тимчасові об’єкти, представлених у вигляді списків на мові програмування Kotlin.

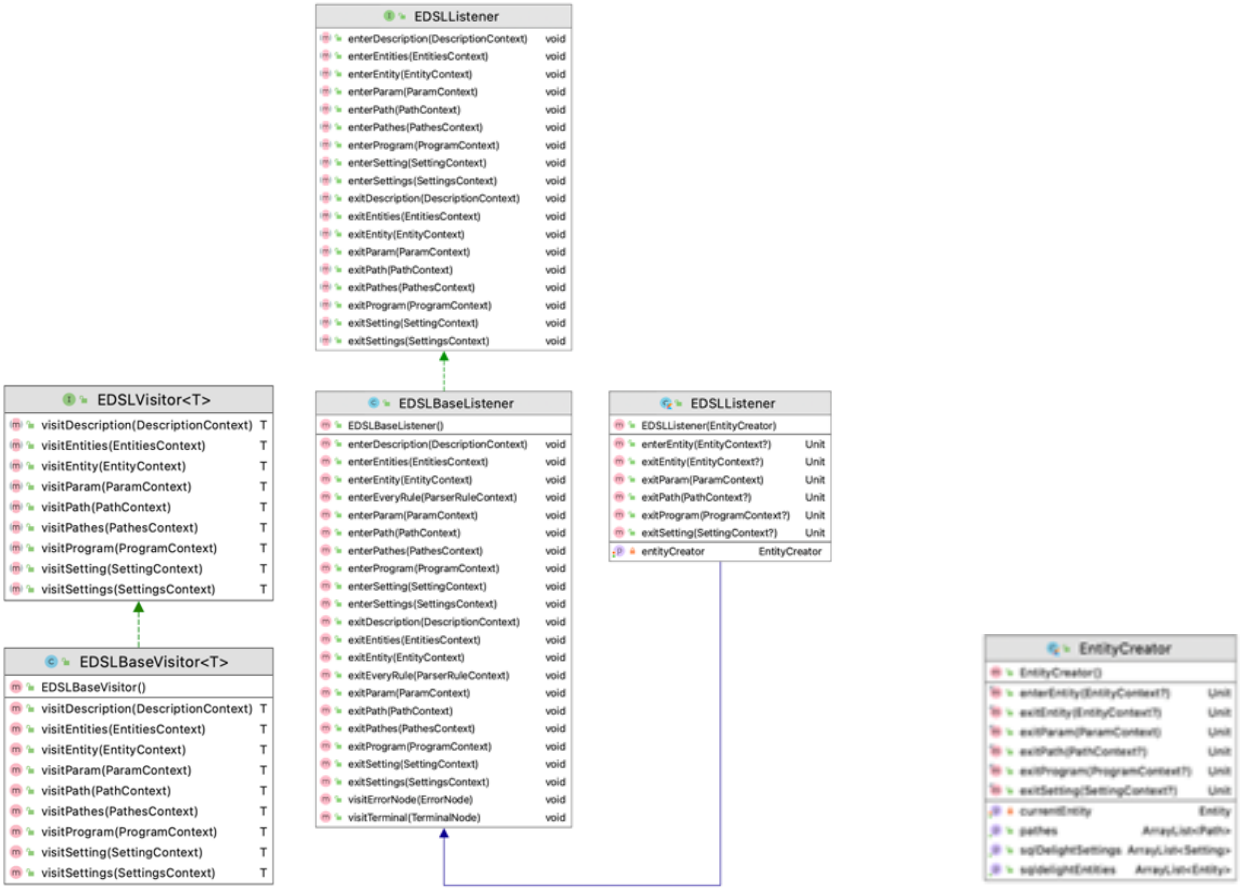


Рисунок 3.5 – UML-діаграма класів синтаксичного аналізатора предметно-орієнтованої мови

Валідатор тимчасових об'єктів. Цей модуль містить опис класів тимчасових об'єктів на мові програмування Kotlin та власне клас валідатору - Validator. В цьому модулі описані такі тимчасові об'єкти як Entity, Param, Fk, Setting, SettingType, Path. Кожен з цих об'єктів описує відповідний йому нетермінал створеної предметно-орієнтованої мови (рис. 3.6).

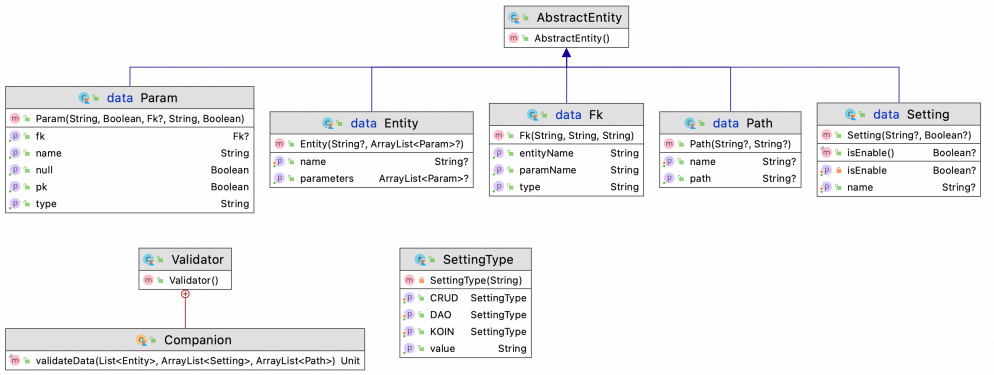


Рисунок 3.6 – UML-діаграма класів модулю валідатора тимчасових об'єктів

Генератор цільовго мультиплатформенного коду. Цей модуль містить класи, які зберігають шаблони мультиплатформенного коду та власне відповідні їм генератори. Шаблони та генератори створені для класів введення залежностей, класу управління БД, класів DAO, класів CRUD-операцій та таблиць БД.

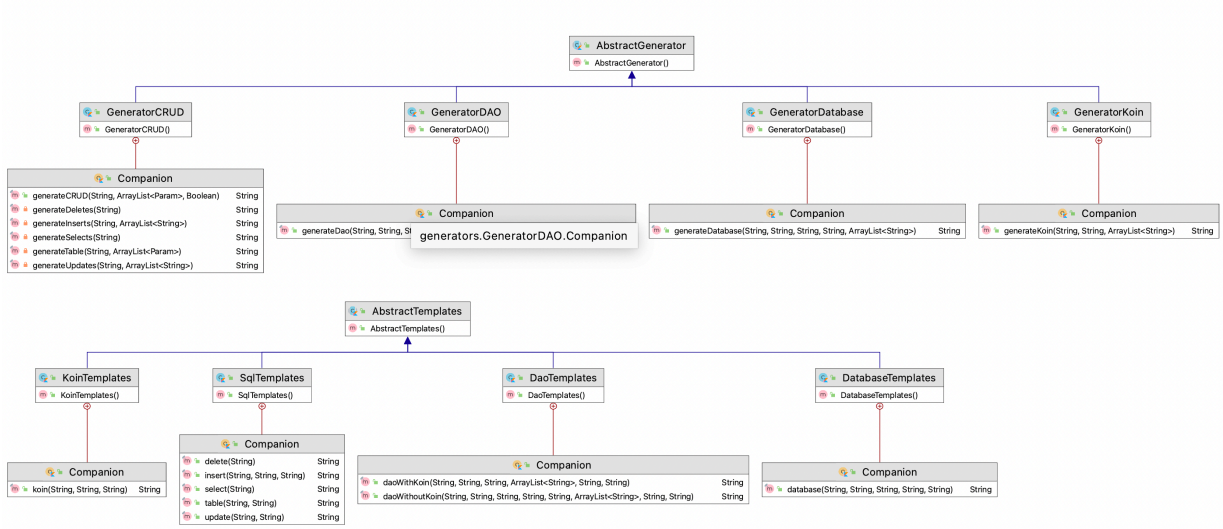


Рисунок 3.7 – UML-діаграма класів модулю генератору цільового мультиплатформенного коду

3.3 Висновки по розділу

Враховуючи, що розробка мультиплатформенних застосувань ведеться у Android Studio, а способом розширення її функціоналу є плагіни, доцільною є реалізація ПЗ для автоматизованого створення первинного мультиплатформенного мобільного проєкту саме у вигляді плагіну.

Розроблено плагін, який служить для автоматизованого створення первинного мультиплатформенного мобільного проєкту, що представляє собою базовий мультиплатформенний Kotlin- та SQL-код shared-модуля мобільного застосунку для реалізації сервісів роботи з БД. Розроблений плагін реалізує запропонований метод автоматизованого створення первинного мультиплатформенного мобільного застосунку.

Функціональність розробленого плагіна реалізується за допомогою програмних компонентів, що підтримують основні етапи процесу траспіляції вхідних описів (сутностей БД та налаштувань DAO, CRUD, DI) в базовий мультиплатформенний Kotlin- та SQL-код shared-модуля мобільного застосунку, а саме:

- лексичний аналізатор предметно-орієнтованої мови;
- синтаксичний аналізатор предметно-орієнтованої мови;
- візітор дерева синтаксичного розбору;
- валідатор тимчасових об’єктів;
- генератор мультиплатформенного коду.

Педставлена реалізація кожного компоненту плагіну у вигляді UML-діаграм.

4 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

Потреба в автоматизації виникає, коли є необхідність у прискоренні певних бізнес-процесів або в оптимізації та спрощенні певної задачі. Як правило, автоматизація дозволяє або скоротити витрати часу розробників, або збільшити їхню продуктивність роботи.

4.1 Мета та порядок досліджень

Для дослідження ефективності запропонованого методу автоматизованого створення первинного мультиплатформенного мобільного проєкту пропонується провести наступні експериментальні дослідження:

– оцінку кількості слів та рядків базового Kotlin- та SQL-коду первинного мультиплатформенного мобільного проєкту;

– оцінку часу створення первинного мультиплатформенного мобільного проєкту.

4.2 Оцінка кількості слів та рядків базового Kotlin- та SQL-коду первинного мультиплатформенного мобільного проєкту

Для визначення кількості рядків базового Kotlin- та SQL-код розглянемо реалізацію мультиплатформенної БД, представлену в пункті 2.1 у другого розділу даної магістерської роботи (тестова предметна область).

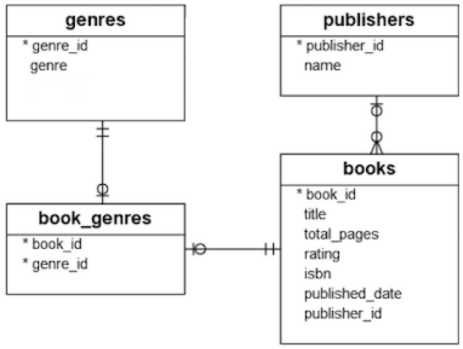


Рисунок 4.1 – Приклад БД

Як показав аналіз, мультиплатформенний проєкт для реалізації сервісів роботи з БД в ручному режимі містить 377 рядків коду (або 817 слів).

Мультиплатформенний же проєкт, реалізований із застосуванням розробленого плагіну, потребує формування вхідних описів сутностей БД на предметно-орієнтованій мові розміром всього 29 рядків коду (або 69 слів) (табл. 4.1).

Кількість рядків коду, що використовуються для опису налаштувань DAO, CRUD, DI на предметно-орієнтованій мові, можна не враховувати, оскільки дані налаштування визначаються в описах один раз та при додаванні сутностей БД не змінюють розмір результуючого базового мультиплатформенного коду.

Таблиця 4.1 – Кількість рядків коду первинного мультиплатформенного проєкту

Кількість	Ручний процес		Автоматизований процес	
	Рядків	Слів	Рядків	Слів
Первинний мультиплатформенний проєкт	377	817	29	69

Отже, після проведеного дослідження на конкретному прикладі, можна зробити висновок, що використання вхідних описів БД та налаштувань DAO, CRUD та DI на розробленій предметно-орієнтованій мові для генерації мультиплатформенного мобільного застосунку містить в 13 раз менше коду, ніж при реалізації даного проєкту виключно в ручному режимі.

4.3 Оцінка часу створення первинного мультиплатформенного мобільного проєкту

Класичним прикладом оцінки ефективності автоматизації є кількість часу на створення мультиплатформенного мобільного проєкту [34]. При підрахунках не

будемо враховувати час на вивчення розробниками синтаксису створеної предметно-орієнтованої мови, тому що розроблена мова характеризується простотою лексичних конструкцій та схожістю з офіційною мовою розробки під платформу Android – Kotlin.

Спираючись на приклад реалізації, який використовувався для підрахунку кількості рядків коду у пункті 4.2, визначимо скільки слів потрібно написати розробнику на мові програмування Kotlin, щоб отримати базовий Kotlin- та SQL-код первинного мультиплатформенного мобільного проєкту.

Визначимо асимптотичну оцінку, а саме нижню межу кількості часу, потрібного для написання мультиплатформенного коду [35]. Підрахунок нижньої межі зробимо за наступною формулою:

$$t_{min} = \Omega(n) \quad (3)$$

де: t_{min} –кількість часу;

Ω – нижня оцінка;

n – кількість набраних слів;

Враховуючи, що середня швидкість набору слів людиною – це 40 слів за хвилину [36], то розробнику для написання 817 слів коду достатньо близько 20 хвилин, а для опису вхідних даних на розробленій предметно-орієнтованій мові - близько 1 хвилини. Часом генерації цільового мультиплатформенного коду можна знехтувати, оскільки код генерується менше однієї секунди.

Виходячи с підрахунків, маємо дві нижні межі часу створення первинного мультиплатформенного проєкту (табл. 4.2):

– для ручного процесу $t_{min} = 20$ хвилин
– для автоматизованого процесу $t_{min} = 1$ хвилина

Таблиця 4.2 – Нижні межі часу, витрачених на реалізацію наведеного прикладу

Нижня межа часу	Ручний процес	Автоматизований процес
	Час (хв)	Час (хв)
Первинний мультиплатформенний проєкт	20	1

Як показує аналіз даного прикладу нижні межі часу ручного і автоматизованого процесу відрізняється у 20 разів.

Отже, щоб ручний процес створення первинного мультиплатформенного мобільного проєкту був співставним з автоматизованим процесом, складність розробленої предметно-орієнтованої мови повинна бути як мінімум у 20 разів вище за складність мови програмування Kotlin. Оскільки, розроблена предметно-орієнтована мова є схожою з офіційною мовою програмування для розробки Android-застосунків, то її складність не відрізняється від складності мови програмування Kotlin. Виходячи з цього, автоматизований процес створення первинного мультиплатформенного мобільного проєкту має ефективність по часу мінімум в 20 разів, тому заощаджує багато часу розробнику.

4.4 Висновки по розділу

Використання вхідних описів БД та налаштувань DAO, CRUD та DI на розробленій предметно-орієнтованій мові для генерації мультиплатформенного мобільного застосунку містить в 13 раз менше коду, ніж при реалізації даного проєкту виключно в ручному режимі.

Як показує аналіз даного прикладу нижні межі часу ручного і автоматизованого процесу відрізняється у 20 разів.

Отже, щоб ручний процес створення первинного мультиплатформенного мобільного проєкту був співставним з автоматизованим процесом, складність розробленої предметно-орієнтованої мови повинна бути як мінімум у 20 разів вище за складність мови програмування Kotlin. Оскільки, розроблена предметно-орієнтована мова є схожою з офіційною мовою програмування для розробки Android-застосунків, то її складність не відрізняється від складності мови програмування Kotlin. Виходячи з цього, автоматизований процес створення первинного мультиплатформенного мобільного проєкту має ефективність по часу мінімум в 20 разів, тому заощаджує багато часу розробнику.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Метод та засоби автоматизованого створення первинного мультиплатформенного мобільного проєкту. Метою роботи є пришвидшення процесу створення мультиплатформенного mobile- проєкту за рахунок автоматизації його етапів.	Програмне забезпечення буде використовуватися у процесы створення первинного мультиплатформенного мобільного проєкту.	Процес створення первинного мультиплатформенного мобільного проєкту значно пришвидшується.

Таблиця 5.2 — Опис ідеї стартап-проєкту

№	Техніко-економічні характери-тики ідеї	Продукція конкурентів	W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт			
1	Автоматизація створення первинного мультиплатформенного мобільного проєкту	Наявний			S
2	Генерація цільовго мультиплатформенного коду за рахунок шаблонів коду	Наявний		N	
3	Реалізація у вигляді плагіну Android Studio	Наявний			S
4	Використання створеної предметно-орієнтованої мови для опису вхідних даних	Наявний			S

5.2 Технологічний аудит ідеї проєкту

Таблиця 5.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Автоматизація створення первинного мультиплатформенного мобільного проєкту	Власний вперше запропонований метод автоматизованого створення первинного мультиплатформенного мобільного проєкту	Наявна	Доступна
2	Генерація цільовго мультиплатформенного коду за рахунок шаблонів коду	Написання власного шаблонного движка	Наявна	Доступна
3	Реалізація у вигляді плагіну Android Studio	Плагін розроблений на мові програмування Kotlin з використання Plugin SDK від JetBrains	Наявна	Доступна
4	Використання створеної предметно-орієнтованої мови для опису вхідних даних	Власно розроблена предметна-орієнтована мова для опису вхідних даних	Наявна	Доступна
Обрана технологія реалізації ідеї проєкту:1				

Висновок: обрано технологію Plugin SDK та власний метод автоматизованого створення первинного мультиплатформенного мобільного проєкту з власною предметно-орієнтовано мову для опису вхідних даних.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 5.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	0
2	Загальний обсяг продаж, грн./ум.од	Невідомо
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Немає обмежень
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі або по ринку, %	Невідома

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, нульову кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Швидке створення первинного мультиплатформенного мобільного проєкту	Mobile-розробники	Відсутні	Автоматизація процесу створення первинного мультиплатформенного мобільного проєкту

Таблиця 5.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Недостатнє фінансування	Отримання замалої кількості коштів від інвесторів	Маркетингові заходи з пошуку клієнтів

Таблиця 5.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Вихід на ринок, зменшення монополії, надання нових рішень у сфері	Розробка нової функціональності, вихід нової продукції на ринок

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: немає конкуренції	немає конкуренції	немає конкуренції
2	Рівень конкурентної боротьби: немає конкуренції	немає конкуренції	немає конкуренції
3	Галузева ознака: немає конкуренції	немає конкуренції	немає конкуренції
4	Конкуренція за видами товарів: немає конкуренції	немає конкуренції	немає конкуренції
5	Характер конкурентних переваг: цінова та не цінова	немає конкуренції	немає конкуренції
6	За інтенсивністю: немає конкуренції	немає конкуренції	немає конкуренції

Таблиця 5.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	немає конкуренції	немає конкуренції	немає конкуренції	немає конкуренції	немає конкуренції
Висновки	немає конкуренції	немає конкуренції	немає конкуренції	немає конкуренції	немає конкуренції

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок:

Для виходу на ринок продукт повинен мати функціонал автоматизованого створення первинного мультиплатфомренного мобільного проєкту, тому що продукт такого роду відсутній на ринку. Але незважаючи на це, продукт все одно повинен задовольняти потреби користувачів в повній мірі, мати необхідний та достатній функціонал з конфігурування та підтримку зі сторони розробників.

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Точність	Використання сучасних технологій для отримання найбільш швидких результатів даних
2	Ціна	Можливість давати більш швидкі результати автоматизованого створення первинного мультиплатформенного проєкту
3	Орієнтованість на кінцевого споживача	Продукт орієнтований на взаємодію з клієнтами (mobile-розробниками)

Таблиця 5.11 — Порівняльний аналіз сильних та слабких сторін системи кешування мало змінних даних

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Точність	17	-	-	-	-	-	-	-
2	Ціна	19	-	-	-	-	-	-	-
3	Орієнтованість на кінцевого споживача	18	-	-	-	-	-	-	-

Таблиця 5.12 — SWOT аналіз стартап-проєкту

Сильні сторони (S):	Слабкі сторони (W):
– Порівняно низька ціна – Унікальний продукт	– Немає
Можливості (O):	Загрози (T):
– Додавання нового функціоналу	– Недостатнє фінансування

Таблиця 5.13 — Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, даний ресурс - наявний	1-2 місяці
2	Реклама	Залучення коштів з інвестицій для реклами товару	2-3 місяці
3	Презентація товару на ІТ-заходах	Ресурс – час, наявні	1-3 місяці

5.4 Розроблення ринкової стратегії проєкту

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії, які займаються розробкою мультиплафторменних застосунків на технології KMM	Висока	Високий	Слабка	Легко

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є компанії, які займаються розробкою мультиплатформенних застосунків на технології KMM. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки надається стандартизований продукт з можливістю розширення функціональності за домовленістю.

Таблиця 5.15 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що взагалі відсутня на ринку	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали	Зниження ступеню замінності товару; Прихильність клієнтів; Унікальні властивості товару;	Стратегія диференціації

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Так	Конкурентів немає	Конкурентів немає	Конкурентів немає

Таблиця 5.17 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Легкість розуміння, зручний інтерфейс, надійність	Стратегія диференціації	Унікальний продукт	Економія часу, зручність застосування
2	Швидка генерація цільового мультиплатформенного коду	Стратегія диференціації	Потужна рекламна компанія	Швидкість роботи
3	Невелика вартість	Спеціальні пропозиції	Період безкоштовного користування	Доступність

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття ніші.

5.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Швидкість отримання результату;	Швидка генерація цільового мультиплатформенного коду	Відсутність потреби писати цільовий мультиплатформенний код вручну
2	Зручність застосування	Плагін зручно використовувати, тому що він встановлюється до Android Studio	Mobile-розробники звикли працювати з плагінами Android Studio

Таблиця 5.19 — Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>1. Товар за задумом</td><td colspan="3">Плагін Android Studio для автоматичного створення первинного мультиплатформенного мобільного проєкту</td></tr>
<tr><td rowspan="8">2. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх/Тл/Е/Ор</td></tr>
<tr><td>Якість</td><td>Нм</td><td>Тл</td></tr>
<tr><td>Точність</td><td>Нм</td><td>Тл</td></tr>
<tr><td>Встановлення в Android Studio</td><td>М</td><td>Тх</td></tr>
<tr><td>Ціна</td><td>Нм</td><td>Е</td></tr>
<tr><td colspan="3">Якість: швидкість автоматизації створення первинного мультиплатформенного мобільного проєкту</td></tr>
<tr><td colspan="3">Пакування: відсутнє</td></tr>
<tr><td colspan="3">Марка: Android Studio Plugin</td></tr>
<tr><td rowspan="2">3. Товар із підкріпленням</td><td colspan="3">До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару</td></tr>
<tr><td colspan="3">Після продажу:,підтримка з боку розробника</td></tr>
</table>

За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності.

Таблиця 5.20 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
-	-	15 000 / міс та вище	250 грн в місяць

Таблиця 5.21 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Орієнтація на безкоштовний функціонал	1.Забезпечення швидкої автоматизації процесу створення первинного мультиплатформенного мобільного проєкту 2.Пробний безкоштовний період користування повним функціоналом	Середня	Безпосередня

Таблиця 5.22 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуютьс я цільові клієнти	Ключові позиції, обрані для позиціонуван ня	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Орієнтація на якісний плагін	GitHub	Співвідноше нн я ціна/якість	Якість продукту та швидкість роботи	Цінуй свій час

Як результат було створено маркетингову програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

5.6 Висновки по розділу

В п’ятому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку, як висновок було вказано, що існує можливість ринкової комерціалізації проєкту.

Розглянувши потенційні групи клієнтів, бар’єри входження, стан конкуренції та конкурентоспроможність проєкту було встановлено, що проєкт є перспективним.

Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

ВИСНОВКИ

Проведений огляд предметної області показав, що технологія КММ для створення мультиплатформенного мобільного застосунку має значні переваги у порівнянні із іншими сучасними технологіями, зокрема, Flutter та React Native.

Оскільки, типовим функціоналом при розробці мультиплатформенного мобільного застосунку є реалізація сервісів роботи з БД, що потребує виконання значної кількості рутинних дій в ручному режимі, то актуальною є задача автоматизації створення програмних компонент для реалізації відповідних сервісів в рамках так званого первинного мультиплатформенного мобільного проєкту, під яким розуміється частина shared-модуля, що містить код відповідних компонент (базовий код).

Процес створення shared-модуля мультиплатформенного мобільного проєкту пропонується здійснювати у два етапи: генерація первинного мультиплатформенного проєкту та подальша реалізація структурних компонент вручну.

Проведено аналіз наявного процесу створення первинного мультиплатформенного мобільного проєкту, який показав, що його розробка в ручному режимі значної кількості рутинних дій.

Запропоновано метод автоматизованого створення первинного мультиплатформенного мобільного проєкту, який передбачає транспіляцію вхідних описів сутностей БД та налаштувань DAO, CRUD, DI у SQL- та Kotlin-компоненти загального коду мультиплатформенного мобільного проєкту.

Розроблена проблемно-орієнтована мова для опису сутностей БД та налаштувань DAO, CRUD, DI, заснована на LL-граматиці, як на різновиді контекстно-вільної граматики, яка має певну схожість з офіційною мовою розробки мобільних застосувань під Android – Kotlin.

Враховуючи, що розробка мультиплатформенних застосувань ведеться у Android Studio, а способом розширення її функціоналу є плагіни, доцільною є реалізація ПЗ для автоматизованого створення первинного мультиплатформенного мобільного проєкту саме у вигляді плагіну.

Розроблено плагін, який дозволяє автоматизовано створювати первинний мультиплатформенний мобільний проєкт, що представляє собою реалізацію сервісів роботи з БД, який реалізує запропонований метод автоматизованого створення первинного мультиплатформенного мобільного проєкту.

Плагін містить компоненти, що віповідають основним етапам траспіляції вхідного опису в базовий мультиплатформенний Kotlin- та SQL-код, а саме:

- лексичний аналізатор предметно-орієнтованої мови;
- синтаксичний аналізатор предметно-орієнтованої мови;
- візітор дерева синтаксичного розбору;
- валідатор тимчасових об’єктів;
- генератор мультиплатформенного коду.

Педставлена реалізація кожного компоненту плагіну у вигляді UML-діаграм.

Оцінка кількості строк коду показала, що опис вхідних даних на розробленій предметно-орієнтованій мові в 13 раз займає менше коду ніж ручне написання базового мультиплатформенного коду. Оцінка кількості часу на створення сервісів роботи з БД показала, що нижня межа часу ручного процесу відрізняється від автоматизованого процесу в 20 разів.

Отже, щоб ручний процес створення первинного мультиплатформенного мобільного проєкту був співставним з автоматизованим процесом, складність розробленої предметно-орієнтованої мови повинна бути як мінімум у 20 разів вище за складність мови програмування Kotlin. Оскільки, розроблена предметно-орієнтована мова є схожою з офіційною мовою програмування для розробки Android-застосунків, то її складність не відрізняється від складності мови програмування Kotlin. Виходячи з цього, автоматизований процес створення

первинного мультиплатформенного мобільного проєкту має ефективність по часу мінімум в 20 разів, тому заощаджує багато часу розробнику.

В п’ятому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку, як висновок було вказано, що існує можливість ринкової комерціалізації проєкту.

Розглянувши потенційні групи клієнтів, бар’єри входження, стан конкуренції та конкурентоспроможність проєкту було встановлено, що проєкт є перспективним.

Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

Наукова новизна роботи полягає у:

– розроблена проблемно-орієнтована мова для опису сутностей БД та налаштувань DAO, CRUD, DI, які є вхідними даними для автоматизованого створення первинного мультиплатформенного мобільного проєкту;

– запропоновано метод автоматизованого створення первинного мультиплатформенного мобільного проєкту, який передбачає транспіляцію вхідних описів сутностей БД та налаштувань DAO, CRUD, DI у SQL та Kotlin компоненти загального коду мультиплатформенного мобільного -проєкту.

Усі поставленні задачі наукової роботи були виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Frank Ableson. Android in Action. / Frank Ableson. // O'Reilly Media. – 2011. – P. 132-146.

2) Mobile Platforms [Електронний ресурс]. – 2015. – Режим доступу: https://www.computerworld.com/article/3269204/confused-about-mobile-platforms-you-re-not-alone-here-s-clarity.html.

3) Rajiv Ramnath. Android 3 SDK Programming For Dummies. / Rajiv Ramnath. // O'Reilly Media. – 2011. – P. 256-270.

4) Mobile Devices [Електронний ресурс]. – 2013. – Режим доступу: https://qz.com/472767/there-are-now-more-than-24000-different-android-devices/.

5) Shaun Lewis. Native Mobile Development: A Cross-Reference for iOS and Android. / Shaun Lewis. // Elephant Publishing. – 2019. – P. 112-191.

6) Multiplatform Development [Електронний ресурс]. – 2019. – Режим доступу: https://hackernoon.com/9-popular-cross-platform-tools-for-app-development-in-2019-53765004761b7.

7) Alberto Miola. Flutter Complete Reference: Create beautiful, fast and native apps for any device. / Alberto Miola. // Pengiun Classics. – 2020. – P. 32-56.

8) Robin Wieruch. The Road to learn React: Your journey to master plain yet pragmatic React.js. / Robin Wieruch // Berkley. – 2018. – P. 156-204.

9) Crossplatform development [Електронний ресурс]. – 2019. – Режим доступу: https://en.wikipedia.org/wiki/Cross-platform_software.

10) KMM Structure [Електронний ресурс]. – 2020. – Режим доступу: https://kotlinlang.org/docs/kmm-understand-project-structure.html.

11) Alan Beaulieu. Learning SQL: Generate, Manipulate, and Retrieve Data. / Alan Beaulieu // DAW. – 2018. – P. 98-127.

12) CRUD [Електронний ресурс]. – 2018. – Режим доступу: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.

13) Koin [Електронний ресурс]. – 2016. – Режим доступу: https://www.raywenderlich.com/9457-dependency-injection-with-koin.

14) SqlDelight [Електронний ресурс]. – 2020. – Режим доступу: https://cashapp.github.io/sqldelight.

15) SqlDelight .sq files [Електронний ресурс]. – 2020.– Режим доступу: https://cashapp.github.io/sqldelight/multiplatform_sqlite/.

16) Mark Seemann. Dependency Injection Principles, Practices, and Patterns. / Mark Seemann. // Dutton. – 2012. – P. 56-120.

17) Koin DI [Електронний ресурс]. – 2021. – Режим доступу: https://insert-koin.io/.

18) Kotlin [Електронний ресурс]. – 2018. – Режим доступу: https://en.wikipedia.org/wiki/Kotlin_(programming_language)#:~:text=On%207%20May%202019%2C%20Google,to%20the%20standard%20Java%20compiler.

19) Pascual Cantos-Gómez. Lexical Collocation Analysis: Advances and Applications. / Pascual Cantos-Gómez. // Plume. – 2018. – P. 35-74.

20) Tibor Kiss. Syntax - Theory and Analysis. / Tibor Kiss. // Pengiun Classics. – 2015. – P. 120-165.

21) AST [Електронний ресурс]. – 2011. – Режим доступу: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

22) ANTLR [Електронний ресурс]. – 2017. – Режим доступу: https://www.antlr.org/.

23) Parse Tree [Електронний ресурс]. – 2012. – Режим доступу: http://aliev.me/runestone/Trees/ParseTree.html.

24) Listener Visitor [Електронний ресурс]. – 2019. – Режим доступу: https://saumitra.me/blog/antlr4-visitor-vs-listener-pattern.

25) Validation [Електронний ресурс]. – 2010. – Режим доступу: https://en.wikipedia.org/wiki/Verification_and_validation.

26) Android Studio [Електронний ресурс]. – 2017. – Режим доступу: https://developer.android.com/studio?gclid=CjwKCAiAwKyNBhBfEiwA_mrUMsj8BjIR16zmWgivFS26-OjQRTPB3b1cGmJznc_nYD7.

27) InteliJ IDEA [Електронний ресурс]. – 2016. – Режим доступу: https://ru.wikipedia.org/wiki/IntelliJ_IDEA

28) Плагіни [Електронний ресурс]. – 2016. – Режим доступу: https://www.jetbrains.com/help/idea/managing-plugins.html.

29) Переваги Kotlin [Електронний ресурс]. – 2018. – Режим доступу: https://lampalampa.net/kotlin-vs-java-chto-luchshe-dlya-android-razrabotki/.

30) Діалогові вікна Android Studio [Електронний ресурс]. – 2019. – Режим доступу: http://developer.alexanderklimov.ru/android/dialogfragment_alertdialog.php.

31) Файл plugin.xml [Електронний ресурс]. – 2016. – Режим доступу: https://plugins.jetbrains.com/docs/intellij/plugin-configuration-file.html.

32) PicoContainer [Електронний ресурс]. – 2021. – Режим доступу: http://picocontainer.com/.

33) Механізм дій плагіну [Електронний ресурс]. – 2016. – Режим доступу: https://plugins.jetbrains.com/docs/intellij/plugin-actions.html.

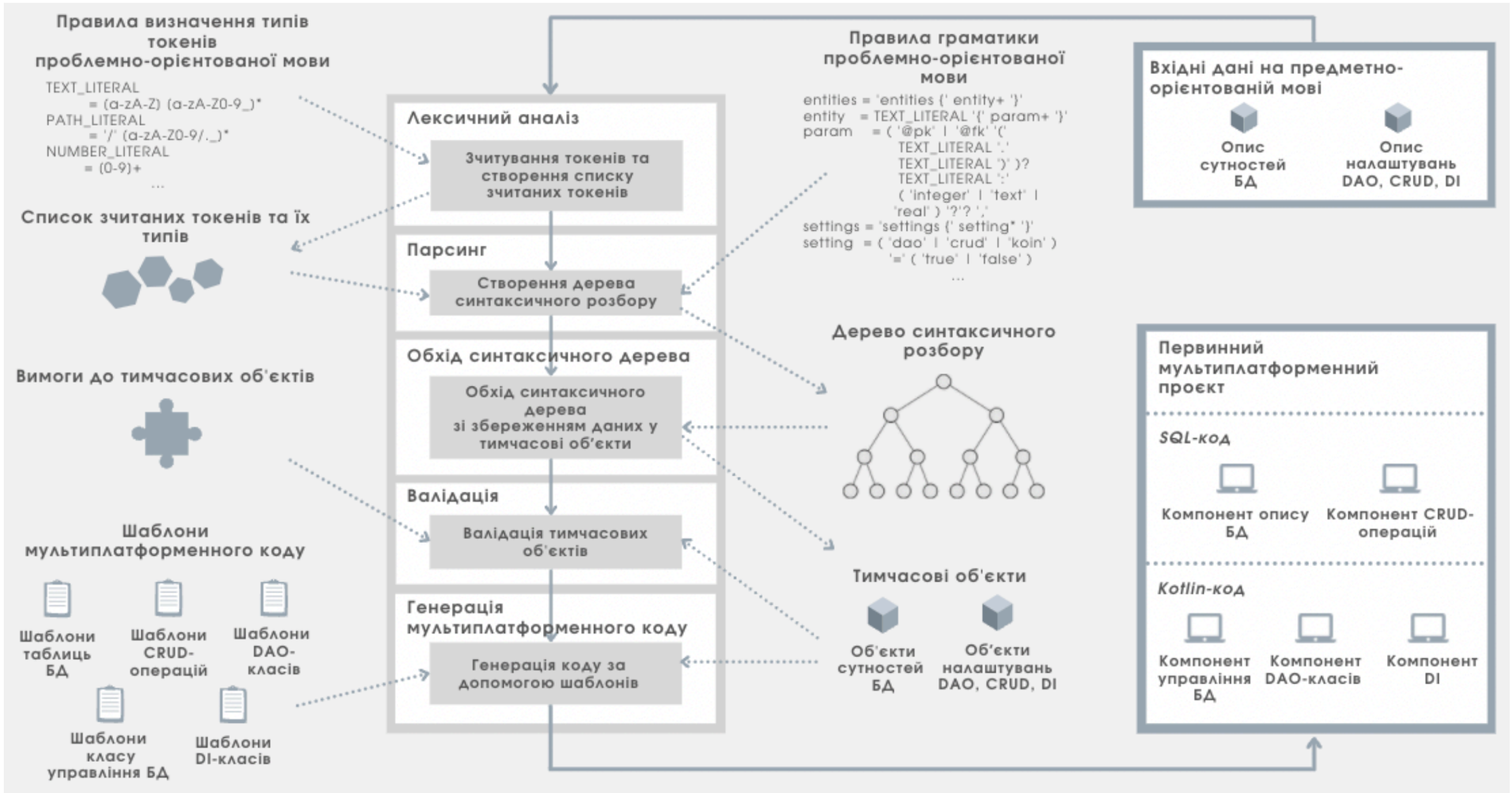
34) Автоматизація [Електронний ресурс]. – 2013. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D1%96%D1%8F.

35) Асимптотична складність [Електронний ресурс]. – 2014. – Режим доступу:https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%87%D0%B8%D1

36) Середня кількість набору слів людиною [Електронний ресурс]. – 2021. – Режим доступу:https://www.typingpal.com/en/typing-test#:~:text=The%20average%20typing%20speed%20is,to%2070%20words%20per%20minute.

ДОДАТОК А ГРАФІЧНІ МАТЕРІАЛИ

Схема транспіляції вхідних описів сутностей БД та налаштувань в базовий загальний код первинного мультиплатформенного мобільного проєкту



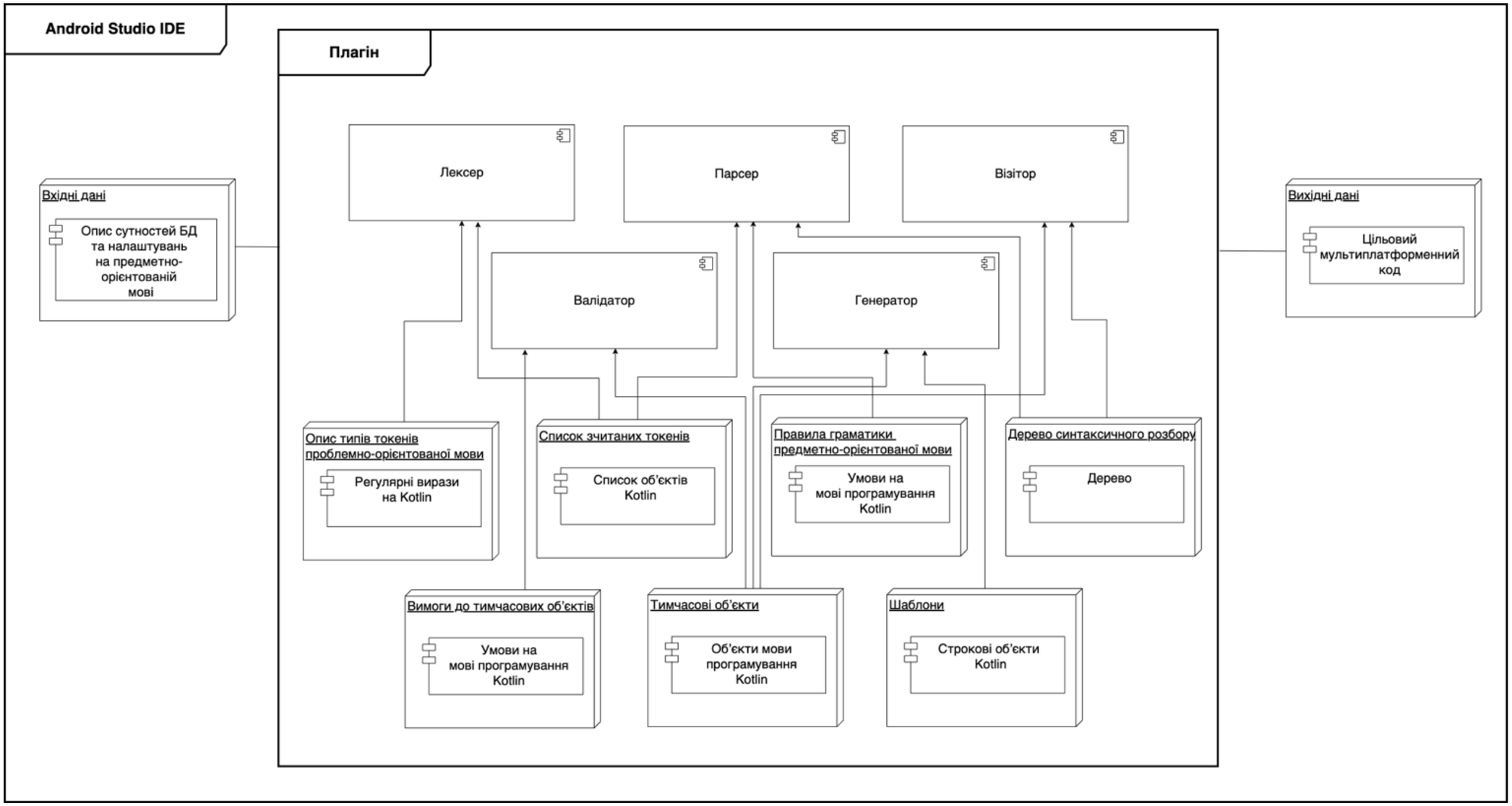
Демонстраційний плакат до магістерської дисертації

Архітектура програмного забезпечення

Виконав студент гр. ІП-02мп Швець Е.Я.

Керівник Муха І.П.

Архітектура плагіну



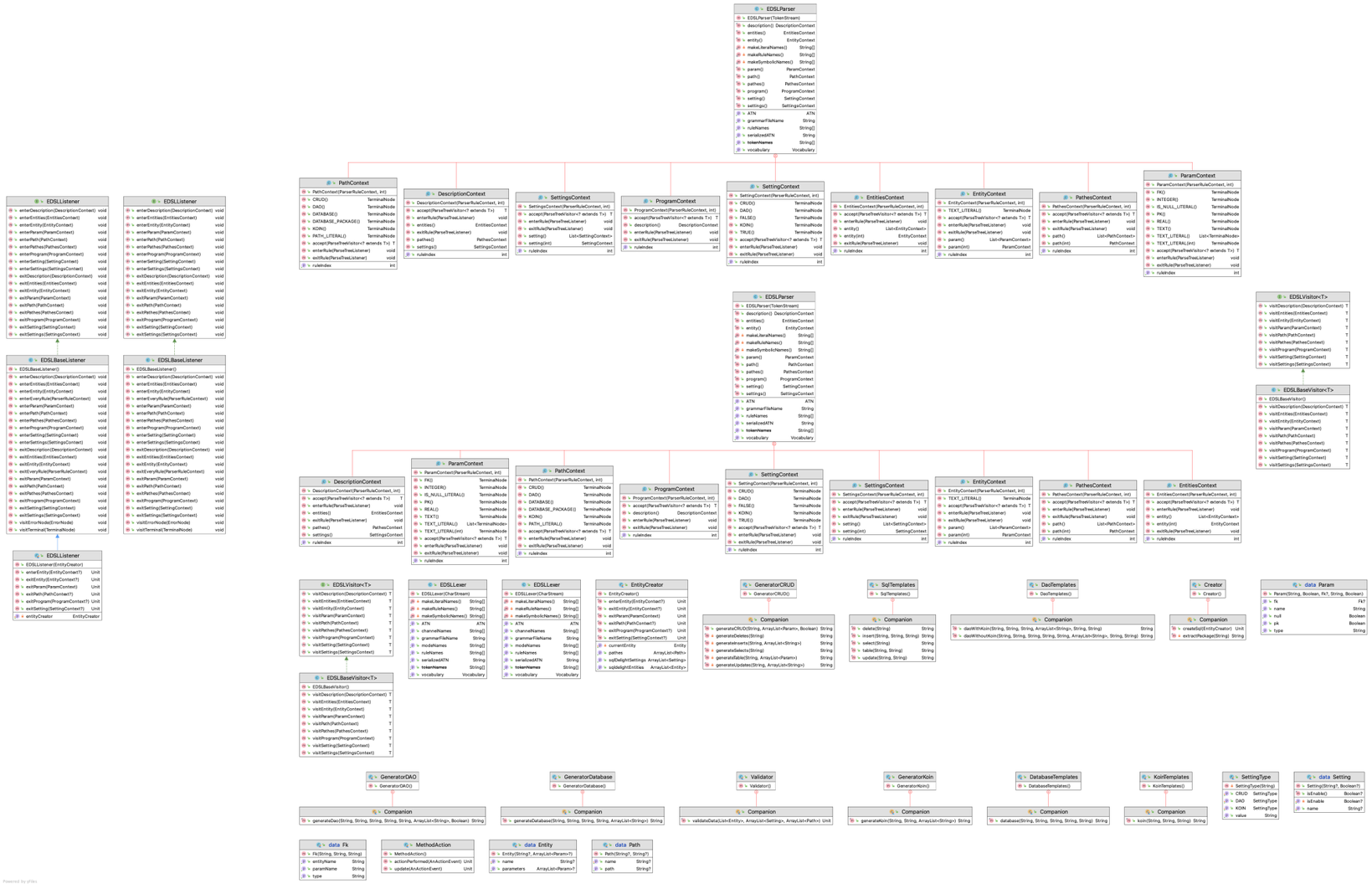
Демонстраційний плакат до магістерської дисертації

Архітектура плагіну

Виконав студент гр. ІП-02мп Швець Е.Я.

Керівник Муха І.П.

UML-діаграма класів плагіну



Powered by yFiles

Демонстраційний плакат до магістерської дисертації

Uml-діаграма класів плагіну

Виконав студент гр. ІП-02мп Швець Е.Я.

Керівник Муха І.П.

ДОДАТОК Б ЛІСТИНГ КОДУ

```
package edsl;// Generated from /Users/eddyshvets/IdeaProjects/antlr4/src/EDSL.g4
by ANTLR 4.9.2
import org.antlr.v4.runtime.Lexer;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.TokenStream;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.misc.*;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class EDSLLexer extends Lexer {
   static { RuntimeMetaData.checkVersion("4.9.2", RuntimeMetaData.VERSION); }

   protected static final DFA[] _decisionToDFA;
   protected static final PredictionContextCache _sharedContextCache =
      new PredictionContextCache();
   public static final int
      T__0=1, T__1=2, T__2=3, T__3=4, T__4=5, T__5=6, T__6=7, T__7=8, T__8=9,
      T__9=10, T__10=11, T__11=12, INTEGER=13, TEXT=14, REAL=15, DAO=16,
CRUD=17,
      KOIN=18, DATABASE=19, DATABASE_PACKAGE=20, TRUE=21, FALSE=22, PK=23,
FK=24,
      TEXT_LITERAL=25, PATH_LITERAL=26, NUMBER_LITERAL=27, IS_NULL_LITERAL=28,
      WS_LITERAL=29;
   public static String[] channelNames = {
      "DEFAULT_TOKEN_CHANNEL", "HIDDEN"
   };

   public static String[] modeNames = {
      "DEFAULT_MODE"
   };

   private static String[] makeRuleNames() {
      return new String[] {
         "T__0", "T__1", "T__2", "T__3", "T__4", "T__5", "T__6", "T__7", "T__8",
         "T__9", "T__10", "T__11", "INTEGER", "TEXT", "REAL", "DAO", "CRUD",
"KOIN",
         "DATABASE", "DATABASE_PACKAGE", "TRUE", "FALSE", "PK", "FK",
"TEXT_LITERAL",
         "PATH_LITERAL", "NUMBER_LITERAL", "IS_NULL_LITERAL", "WS_LITERAL"
      };
   }
   public static final String[] ruleNames = makeRuleNames();

   private static String[] makeLiteralNames() {
      return new String[] {
         null, "'description {'", "'}'", "'entities {'", "'{'", "'('", "'.'",
         "')'", "':'", "','", "'settings {'", "'='", "'pathes {'", "'integer'",
         "'text'", "'real'", "'dao'", "'crud'", "'koin'", "'database'",
"'databasePackage'",
         "'true'", "'false'", "'@pk'", "'@fk'", null, null, null, "'?'"
      };
   }
   private static final String[] _LITERAL_NAMES = makeLiteralNames();
```

```
    private static String[] makeSymbolicNames() {
        return new String[] {
            null, null, null, null, null, null, null, null, null, null, null, null,
            null, "INTEGER", "TEXT", "REAL", "DAO", "CRUD", "KOIN", "DATABASE",
"DATABASE_PACKAGE",
            "TRUE", "FALSE", "PK", "FK", "TEXT_LITERAL", "PATH_LITERAL",
"NUMBER_LITERAL",
            "IS_NULL_LITERAL", "WS_LITERAL"
        };
    }
    private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
    public static final Vocabulary VOCABULARY = new
VocabularyImpl(_LITERAL_NAMES, _SYMBOLIC_NAMES);

    /**
     * @deprecated Use {@link #VOCABULARY} instead.
     */
    @Deprecated
    public static final String[] tokenNames;
    static {
        tokenNames = new String[_SYMBOLIC_NAMES.length];
        for (int i = 0; i < tokenNames.length; i++) {
            tokenNames[i] = VOCABULARY.getLiteralName(i);
            if (tokenNames[i] == null) {
                tokenNames[i] = VOCABULARY.getSymbolicName(i);
            }

            if (tokenNames[i] == null) {
                tokenNames[i] = "<INVALID>";
            }
        }
    }

    @Override
    @Deprecated
    public String[] getTokenNames() {
        return tokenNames;
    }

    @Override

    public Vocabulary getVocabulary() {
        return VOCABULARY;
    }


    public EDSLLexer(CharStream input) {
        super(input);
        _interp = new
LexerATNSimulator(this,_ATN,_decisionToDFA,_sharedContextCache);
    }

    @Override
    public String getGrammarFileName() { return "EDSL.g4"; }

    @Override
    public String[] getRuleNames() { return ruleNames; }

    @Override
    public String getSerializedATN() { return _serializedATN; }

    @Override
```

```
    public String[] getChannelNames() { return channelNames; }

    @Override
    public String[] getModeNames() { return modeNames; }

    @Override
    public ATN getATN() { return _ATN; }

    public static final String _serializedATN =
        "\3\u608b\ua72a\u8133\ub9ed\u417c\u3be7\u7786\u5964\2\37\u00e2\b\1\4\2"+
        "\t\2\4\3\t\3\4\4\t\4\4\5\t\5\4\6\t\6\4\7\t\7\4\b\t\b\4\t\t\t\4\n\t\n\4"+
        "\13\t\13\4\f\t\f\4\r\t\r\4\16\t\16\4\17\t\17\4\20\t\20\4\21\t\21\4\22"+
        "\t\22\4\23\t\23\4\24\t\24\4\25\t\25\4\26\t\26\4\27\t\27\4\30\t\30\4\31"+
        "\t\31\4\32\t\32\4\33\t\33\4\34\t\34\4\35\t\35\4\36\t\36\3\2\3\2\3\2\3"+
        "\2\3\2\3\2\3\2\3\2\3\2\3\2\3\2\3\2\3\2\3\2\3\2\3\3\3\3\3\4\3\4\3\4\3\4\3\4\3\4"+
        "\3\4\3\4\3\4\3\4\3\4\3\4\3\5\3\5\3\6\3\6\3\7\3\7\3\b\3\b\3\t\3\t\3\n\3"+
        "\n\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\f\3\f\3\r"+
        "\3\r\3\r\3\r\3\r\3\r\3\r\3\r\3\r\3\16\3\16\3\16\3\16\3\16\3\16\3\16\3"+
        "\16\3\17\3\17\3\17\3\17\3\17\3\20\3\20\3\20\3\20\3\20\3\21\3\21\3\21\3"+
        "\21\3\22\3\22\3\22\3\22\3\22\3\23\3\23\3\23\3\23\3\23\3\24\3\24\3\24\3"+
        "\24\3\24\3\24\3\24\3\24\3\25\3\25\3\25\3\25\3\25\3\25\3\25\3\25\3\25\3"+
        "\25\3\25\3\25\3\25\3\25\3\25\3\25\3\25\3\26\3\26\3\26\3\26\3\26\3\27\3"+
        "\27\3\27\3\27\3\27\3\30\3\30\3\30\3\30\3\31\3\31\3\31\3\31\3\32\3"+
        "\32\7\32\u00c9\n\32\f\32\16\32\u00cc\13\32\3\33\3\33\7\33\u00d0\n\33\f"+
        "\33\16\33\u00d3\13\33\3\34\6\34\u00d6\n\34\r\34\16\34\u00d7\3\35\3\35"+
        "\3\36\6\36\u00dd\n\36\r\36\16\36\u00de\3\36\3\36\2\2\37\3\3\5\4\7\5\t"+
        "\6\13\7\r\b\17\t\21\n\23\13\25\f\27\r\31\16\33\17\35\20\37\21!\22#\23"+
        "%\24\'\25)\26+\27-\30/\31\61\32\63\33\65\34\67\359\36;\37\3\2\b\4\2C\\"+
        "c|\6\2\62;C\\aac|\3\2\61\61\6\2\60;C\\aac|\3\2\62;\4\2\13\f\"\"\2\u00e5"+
        "\2\3\3\2\2\2\5\3\2\2\2\7\3\2\2\2\t\3\2\2\2\13\3\2\2\2\r\3\2"+
        "\2\2\17\3\2\2\2\21\3\2\2\2\23\3\2\2\2\25\3\2\2\2\27\3\2\2\2"+
        "\2\31\3\2\2\2\33\3\2\2\2\35\3\2\2\2\37\3\2\2\2!\3\2\2\2\2#\3\2"+
        "\2\2%\3\2\2\2\'\3\2\2\2)\3\2\2\2+\3\2\2\2-\3\2\2\2/\3\2\2"+
        "\2\61\3\2\2\2\63\3\2\2\2\65\3\2\2\2\67\3\2\2\29\3\2\2\2;\3"+
        "\2\2\2\3=\3\2\2\2\5K\3\2\2\2\7M\3\2\2\2\tX\3\2\2\2\13Z\3\2\2\2\r\\\3\2"+
        "\2\17^\3\2\2\2\21`\3\2\2\2\23b\3\2\2\2\25d\3\2\2\2\27o\3\2\2\2\31q\3"+

"\2\2\2\33z\3\2\2\2\35\u0082\3\2\2\2\37\u0087\3\2\2\2!\u008c\3\2\2\2#\u0090"+
        "\3\2\2\2%\u0095\3\2\2\2\'\u009a\3\2\2\2)\u00a3\3\2\2\2+\u00b3\3\2\2\2"+
        "-
\u00b8\3\2\2\2/\u00be\3\2\2\2\61\u00c2\3\2\2\2\63\u00c6\3\2\2\2\65\u00cd"+
        "\3\2\2\2\67\u00d5\3\2\2\29\u00d9\3\2\2\2;\u00dc\3\2\2\2=>\7f\2\2>?\7g"+
        "\2\2?@\7u\2\2@A\7e\2\2AB\7t\2\2BC\7k\2\2CD\7r\2\2DE\7v\2\2EF\7k\2\2FG"+
        "\7q\2\2GH\7p\2\2HI\7\"\2\2IJ\7}\2\2J\4\3\2\2\2KL\7\177\2\2L\6\3\2\2\2"+
        "MN\7g\2\2NO\7p\2\2OP\7v\2\2PQ\7k\2\2QR\7v\2\2RS\7k\2\2ST\7g\2\2TU\7u\2"+
        "\2UV\7\"\2\2VW\7}\2\2W\b\3\2\2\2XY\7}\2\2Y\n\3\2\2\2Z[\7*\2\2[\f\3\2\2"+
        "\2\\]\7\60\2\2]\16\3\2\2\2^_\7+\2\2_\20\3\2\2\2`a\7<\2\2a\22\3\2\2\2b"+
        "c\7.\2\2c\24\3\2\2\2de\7u\2\2ef\7g\2\2fg\7v\2\2gh\7v\2\2hi\7k\2\2ij\7"+
        "p\2\2jk\7i\2\2kl\7u\2\2lm\7\"\2\2mn\7}\2\2n\26\3\2\2\2op\7?\2\2p\30\3"+
        "\2\2\2qr\7r\2\2rs\7c\2\2st\7v\2\2tu\7j\2\2uv\7g\2\2vw\7u\2\2wx\7\"\2\2"+
        "xy\7}\2\2y\32\3\2\2\2z{\7k\2\2{|\7p\2\2|}\7v\2\2}~\7g\2\2~\177\7i\2\2"+
        "\177\u0080\7g\2\2\u0080\u0081\7t\2\2\u0081\34\3\2\2\2\u0082\u0083\7v\2"+
        "\2\u0083\u0084\7g\2\2\u0084\u0085\7z\2\2\u0085\u0086\7v\2\2\u0086\36\3"+
        "\2\2\2\u0087\u0088\7t\2\2\u0088\u0089\7g\2\2\u0089\u008a\7c\2\2\u008a"+
        "\u008b\7n\2\2\u008b
\3\2\2\2\u008c\u008d\7f\2\2\u008d\u008e\7c\2\2\u008e"+

"\u008f\7q\2\2\u008f\"\3\2\2\2\u0090\u0091\7e\2\2\u0091\u0092\7t\2\2\u0092"+

"\u0093\7w\2\2\u0093\u0094\7f\2\2\u0094$\3\2\2\2\u0095\u0096\7m\2\2\u0096"+

"\u0097\7q\2\2\u0097\u0098\7k\2\2\u0098\u0099\7p\2\2\u0099&\3\2\2\2\u009a"+
```

```
      "\u009b\7f\2\2\u009b\u009c\7c\2\2\u009c\u009d\7v\2\2\u009d\u009e\7c\2\2"+
      "\u009e\u009f\7d\2\2\u009f\u00a0\7c\2\2\u00a0\u00a1\7u\2\2\u00a1\u00a2"+

"\7g\2\2\u00a2(\3\2\2\2\u00a3\u00a4\7f\2\2\u00a4\u00a5\7c\2\2\u00a5\u00a6"+
      "\7v\2\2\u00a6\u00a7\7c\2\2\u00a7\u00a8\7d\2\2\u00a8\u00a9\7c\2\2\u00a9"+
      "\u00aa\7u\2\2\u00aa\u00ab\7g\2\2\u00ab\u00ac\7R\2\2\u00ac\u00ad\7c\2\2"+
      "\u00ad\u00ae\7e\2\2\u00ae\u00af\7m\2\2\u00af\u00b0\7c\2\2\u00b0\u00b1"+

"\7i\2\2\u00b1\u00b2\7g\2\2\u00b2*\3\2\2\2\u00b3\u00b4\7v\2\2\u00b4\u00b5"+

"\7t\2\2\u00b5\u00b6\7w\2\2\u00b6\u00b7\7g\2\2\u00b7,\3\2\2\2\u00b8\u00b9"+
      "\7h\2\2\u00b9\u00ba\7c\2\2\u00ba\u00bb\7n\2\2\u00bb\u00bc\7u\2\2\u00bc"+

"\u00bd\7g\2\2\u00bd.\3\2\2\2\u00be\u00bf\7B\2\2\u00bf\u00c0\7r\2\2\u00c0"+

"\u00c1\7m\2\2\u00c1\60\3\2\2\2\u00c2\u00c3\7B\2\2\u00c3\u00c4\7h\2\2\u00c4"+
      "\u00c5\7m\2\2\u00c5\62\3\2\2\2\u00c6\u00ca\t\2\2\2\u00c7\u00c9\t\3\2\2"+

"\u00c8\u00c7\3\2\2\2\u00c9\u00cc\3\2\2\2\u00ca\u00c8\3\2\2\2\u00ca\u00cb"+
      "\3\2\2\2\u00cb\64\3\2\2\2\u00cc\u00ca\3\2\2\2\u00cd\u00d1\t\4\2\2\u00ce"+
      "\u00d0\t\5\2\2\u00cf\u00ce\3\2\2\2\u00d0\u00d3\3\2\2\2\u00d1\u00cf\3\2"+

"\2\2\u00d1\u00d2\3\2\2\2\u00d2\66\3\2\2\2\u00d3\u00d1\3\2\2\2\u00d4\u00d6"+

"\t\6\2\2\u00d5\u00d4\3\2\2\2\u00d6\u00d7\3\2\2\2\u00d7\u00d5\3\2\2\2\u00d7"+
      "\u00d8\3\2\2\2\u00d88\3\2\2\2\u00d9\u00da\7A\2\2\u00da:\3\2\2\2\u00db"+
      "\u00dd\t\7\2\2\u00dc\u00db\3\2\2\2\u00dd\u00de\3\2\2\2\u00de\u00dc\3\2"+
      "\2\u00de\u00df\3\2\2\2\u00df\u00e0\3\2\2\2\u00e0\u00e1\b\36\2\2\u00e1"+
      "<\3\2\2\2\7\2\u00ca\u00d1\u00d7\u00de\3\b\2\2";
   public static final ATN _ATN =
      new ATNDeserializer().deserialize(_serializedATN.toCharArray());
   static {
      _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
      for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
         _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
      }
   }
}


package edsl;// Generated from /Users/eddyshvets/IdeaProjects/antlr4/src/EDSL.g4
by ANTLR 4.9.2
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.misc.*;
import org.antlr.v4.runtime.tree.*;
import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class EDSLParser extends Parser {
   static { RuntimeMetaData.checkVersion("4.9.2", RuntimeMetaData.VERSION); }

   protected static final DFA[] _decisionToDFA;
   protected static final PredictionContextCache _sharedContextCache =
      new PredictionContextCache();
   public static final int
      T__0=1, T__1=2, T__2=3, T__3=4, T__4=5, T__5=6, T__6=7, T__7=8, T__8=9,
      T__9=10, T__10=11, T__11=12, INTEGER=13, TEXT=14, REAL=15, DAO=16,
CRUD=17,
      KOIN=18, DATABASE=19, DATABASE_PACKAGE=20, TRUE=21, FALSE=22, PK=23,
```

```
FK=24,
      TEXT_LITERAL=25, PATH_LITERAL=26, NUMBER_LITERAL=27, IS_NULL_LITERAL=28,
      WS_LITERAL=29;
   public static final int
      RULE_program = 0, RULE_description = 1, RULE_entities = 2, RULE_entity =
3,
      RULE_param = 4, RULE_settings = 5, RULE_setting = 6, RULE_pathes = 7,
      RULE_path = 8;
   private static String[] makeRuleNames() {
      return new String[] {
         "program", "description", "entities", "entity", "param", "settings",
         "setting", "pathes", "path"
      };
   }
   public static final String[] ruleNames = makeRuleNames();

   private static String[] makeLiteralNames() {
      return new String[] {
         null, "'description {'", "'}'", "'entities {'", "'{'", "'('", "'.'",
         "')'", "':'", "','", "'settings {'", "'='", "'pathes {'", "'integer'",
         "'text'", "'real'", "'dao'", "'crud'", "'koin'", "'database'",
"'databasePackage'",
         "'true'", "'false'", "'@pk'", "'@fk'", null, null, null, "'?'"
      };
   }
   private static final String[] _LITERAL_NAMES = makeLiteralNames();
   private static String[] makeSymbolicNames() {
      return new String[] {
         null, null, null, null, null, null, null, null, null, null, null, null,
         null, "INTEGER", "TEXT", "REAL", "DAO", "CRUD", "KOIN", "DATABASE",
"DATABASE_PACKAGE",
         "TRUE", "FALSE", "PK", "FK", "TEXT_LITERAL", "PATH_LITERAL",
"NUMBER_LITERAL",
         "IS_NULL_LITERAL", "WS_LITERAL"
      };
   }
   private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
   public static final Vocabulary VOCABULARY = new
VocabularyImpl(_LITERAL_NAMES, _SYMBOLIC_NAMES);

   /**
    * @deprecated Use {@link #VOCABULARY} instead.
    */
   @Deprecated
   public static final String[] tokenNames;
   static {
      tokenNames = new String[_SYMBOLIC_NAMES.length];
      for (int i = 0; i < tokenNames.length; i++) {
         tokenNames[i] = VOCABULARY.getLiteralName(i);
         if (tokenNames[i] == null) {
            tokenNames[i] = VOCABULARY.getSymbolicName(i);
         }

         if (tokenNames[i] == null) {
            tokenNames[i] = "<INVALID>";
         }
      }
   }

   @Override
   @Deprecated
   public String[] getTokenNames() {
```

```
      return tokenNames;
   }

   @Override

   public Vocabulary getVocabulary() {
      return VOCABULARY;
   }

   @Override
   public String getGrammarFileName() { return "EDSL.g4"; }

   @Override
   public String[] getRuleNames() { return ruleNames; }

   @Override
   public String getSerializedATN() { return _serializedATN; }

   @Override
   public ATN getATN() { return _ATN; }

   public EDSLParser(TokenStream input) {
      super(input);
      _interp = new
ParserATNSimulator(this,_ATN,_decisionToDFA,_sharedContextCache);
   }

   public static class ProgramContext extends ParserRuleContext {
      public DescriptionContext description() {
         return getRuleContext(DescriptionContext.class,0);
      }
      public ProgramContext(ParserRuleContext parent, int invokingState) {
         super(parent, invokingState);
      }
      @Override public int getRuleIndex() { return RULE_program; }
      @Override
      public void enterRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterProgram(this);
      }
      @Override
      public void exitRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitProgram(this);
      }
      @Override
      public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
         if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitProgram(this);
         else return visitor.visitChildren(this);
      }
   }

   public final ProgramContext program() throws RecognitionException {
      ProgramContext _localctx = new ProgramContext(_ctx, getState());
      enterRule(_localctx, 0, RULE_program);
      try {
         enterOuterAlt(_localctx, 1);
         {
         setState(18);
         description();
         }
```

```
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class DescriptionContext extends ParserRuleContext {
        public EntitiesContext entities() {
            return getRuleContext(EntitiesContext.class,0);
        }
        public SettingsContext settings() {
            return getRuleContext(SettingsContext.class,0);
        }
        public PathesContext pathes() {
            return getRuleContext(PathesContext.class,0);
        }
        public DescriptionContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_description; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterDescription(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitDescription(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitDescription(this);
            else return visitor.visitChildren(this);
        }
    }

    public final DescriptionContext description() throws RecognitionException {
        DescriptionContext _localctx = new DescriptionContext(_ctx, getState());
        enterRule(_localctx, 2, RULE_description);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(20);
            match(T__0);
            setState(21);
            entities();
            setState(22);
            settings();
            setState(23);
            pathes();
            setState(24);
            match(T__1);
            }
        }
```

```
      catch (RecognitionException re) {
         _localctx.exception = re;
         _errHandler.reportError(this, re);
         _errHandler.recover(this, re);
      }
      finally {
         exitRule();
      }
      return _localctx;
   }

   public static class EntitiesContext extends ParserRuleContext {
      public List<EntityContext> entity() {
         return getRuleContexts(EntityContext.class);
      }
      public EntityContext entity(int i) {
         return getRuleContext(EntityContext.class,i);
      }
      public EntitiesContext(ParserRuleContext parent, int invokingState) {
         super(parent, invokingState);
      }
      @Override public int getRuleIndex() { return RULE_entities; }
      @Override
      public void enterRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterEntities(this);
      }
      @Override
      public void exitRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitEntities(this);
      }
      @Override
      public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
         if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitEntities(this);
         else return visitor.visitChildren(this);
      }
   }

   public final EntitiesContext entities() throws RecognitionException {
      EntitiesContext _localctx = new EntitiesContext(_ctx, getState());
      enterRule(_localctx, 4, RULE_entities);
      int _la;
      try {
         enterOuterAlt(_localctx, 1);
         {
         setState(26);
         match(T__2);
         setState(28);
         _errHandler.sync(this);
         _la = _input.LA(1);
         do {
            {
            {
            setState(27);
            entity();
            }
            }
            setState(30);
            _errHandler.sync(this);
            _la = _input.LA(1);
```

```
            } while ( _la==TEXT_LITERAL );
            setState(32);
            match(T__1);
            }
         }
         catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
         }
         finally {
            exitRule();
         }
         return _localctx;
      }

      public static class EntityContext extends ParserRuleContext {
         public TerminalNode TEXT_LITERAL() { return
getToken(EDSLParser.TEXT_LITERAL, 0); }
         public List<ParamContext> param() {
            return getRuleContexts(ParamContext.class);
         }
         public ParamContext param(int i) {
            return getRuleContext(ParamContext.class,i);
         }
         public EntityContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
         }
         @Override public int getRuleIndex() { return RULE_entity; }
         @Override
         public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterEntity(this);
         }
         @Override
         public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitEntity(this);
         }
         @Override
         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitEntity(this);
            else return visitor.visitChildren(this);
         }
      }

      public final EntityContext entity() throws RecognitionException {
         EntityContext _localctx = new EntityContext(_ctx, getState());
         enterRule(_localctx, 6, RULE_entity);
         int _la;
         try {
            enterOuterAlt(_localctx, 1);
            {
            setState(34);
            match(TEXT_LITERAL);
            setState(35);
            match(T__3);
            setState(37);
            _errHandler.sync(this);
            _la = _input.LA(1);
            do {
```

```
            {
            {
            setState(36);
            param();
            }
            }
            setState(39);
            _errHandler.sync(this);
            _la = _input.LA(1);
         } while ( (((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << PK) | (1L <<
FK) | (1L << TEXT_LITERAL))) != 0) );
         setState(41);
         match(T__1);
         }
      }
      catch (RecognitionException re) {
         _localctx.exception = re;
         _errHandler.reportError(this, re);
         _errHandler.recover(this, re);
      }
      finally {
         exitRule();
      }
      return _localctx;
   }

   public static class ParamContext extends ParserRuleContext {
      public List<TerminalNode> TEXT_LITERAL() { return
getTokens(EDSLParser.TEXT_LITERAL); }
      public TerminalNode TEXT_LITERAL(int i) {
         return getToken(EDSLParser.TEXT_LITERAL, i);
      }
      public TerminalNode INTEGER() { return getToken(EDSLParser.INTEGER, 0); }
      public TerminalNode TEXT() { return getToken(EDSLParser.TEXT, 0); }
      public TerminalNode REAL() { return getToken(EDSLParser.REAL, 0); }
      public TerminalNode IS_NULL_LITERAL() { return
getToken(EDSLParser.IS_NULL_LITERAL, 0); }
      public TerminalNode FK() { return getToken(EDSLParser.FK, 0); }
      public TerminalNode PK() { return getToken(EDSLParser.PK, 0); }
      public ParamContext(ParserRuleContext parent, int invokingState) {
         super(parent, invokingState);
      }
      @Override public int getRuleIndex() { return RULE_param; }
      @Override
      public void enterRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterParam(this);
      }
      @Override
      public void exitRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitParam(this);
      }
      @Override
      public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
         if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitParam(this);
         else return visitor.visitChildren(this);
      }
   }

   public final ParamContext param() throws RecognitionException {
```

```
        ParamContext _localctx = new ParamContext(_ctx, getState());
        enterRule(_localctx, 8, RULE_param);
        int _la;
        try {
           enterOuterAlt(_localctx, 1);
           {
           setState(52);
           _errHandler.sync(this);
           switch (_input.LA(1)) {
           case PK:
           case TEXT_LITERAL:
              {
              setState(44);
              _errHandler.sync(this);
              _la = _input.LA(1);
              if (_la==PK) {
                 {
                 setState(43);
                 match(PK);
                 }
              }

              }
              break;
           case FK:
              {
              {
              setState(46);
              match(FK);
              setState(47);
              match(T__4);
              setState(48);
              match(TEXT_LITERAL);
              setState(49);
              match(T__5);
              setState(50);
              match(TEXT_LITERAL);
              setState(51);
              match(T__6);
              }
              }
              break;
           default:
              throw new NoViableAltException(this);
           }
           setState(54);
           match(TEXT_LITERAL);
           setState(55);
           match(T__7);
           setState(56);
           _la = _input.LA(1);
           if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << INTEGER) | (1L
<< TEXT) | (1L << REAL))) != 0)) ) {
           _errHandler.recoverInline(this);
           }
           else {
              if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
              _errHandler.reportMatch(this);
              consume();
           }
           setState(58);
           _errHandler.sync(this);
```

```
            _la = _input.LA(1);
            if (_la==IS_NULL_LITERAL) {
               {
               setState(57);
               match(IS_NULL_LITERAL);
               }
            }

            setState(60);
            match(T__8);
            }
         }
         catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
         }
         finally {
            exitRule();
         }
         return _localctx;
      }

      public static class SettingsContext extends ParserRuleContext {
         public List<SettingContext> setting() {
            return getRuleContexts(SettingContext.class);
         }
         public SettingContext setting(int i) {
            return getRuleContext(SettingContext.class,i);
         }
         public SettingsContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
         }
         @Override public int getRuleIndex() { return RULE_settings; }
         @Override
         public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterSettings(this);
         }
         @Override
         public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitSettings(this);
         }
         @Override
         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitSettings(this);
            else return visitor.visitChildren(this);
         }
      }

      public final SettingsContext settings() throws RecognitionException {
         SettingsContext _localctx = new SettingsContext(_ctx, getState());
         enterRule(_localctx, 10, RULE_settings);
         int _la;
         try {
            enterOuterAlt(_localctx, 1);
            {
            setState(62);
            match(T__9);
            setState(66);
```

```
            _errHandler.sync(this);
            _la = _input.LA(1);
            while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << DAO) | (1L <<
CRUD) | (1L << KOIN))) != 0)) {
                {
                {
                setState(63);
                setting();
                }
                }
                setState(68);
                _errHandler.sync(this);
                _la = _input.LA(1);
            }
            setState(69);
            match(T__1);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class SettingContext extends ParserRuleContext {
        public TerminalNode DAO() { return getToken(EDSLParser.DAO, 0); }
        public TerminalNode CRUD() { return getToken(EDSLParser.CRUD, 0); }
        public TerminalNode KOIN() { return getToken(EDSLParser.KOIN, 0); }
        public TerminalNode TRUE() { return getToken(EDSLParser.TRUE, 0); }
        public TerminalNode FALSE() { return getToken(EDSLParser.FALSE, 0); }
        public SettingContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_setting; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterSetting(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitSetting(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitSetting(this);
            else return visitor.visitChildren(this);
        }
    }

    public final SettingContext setting() throws RecognitionException {
        SettingContext _localctx = new SettingContext(_ctx, getState());
        enterRule(_localctx, 12, RULE_setting);
        int _la;
        try {
```

```
            enterOuterAlt(_localctx, 1);
            {
            setState(71);
            _la = _input.LA(1);
            if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << DAO) | (1L <<
CRUD) | (1L << KOIN))) != 0)) ) {
            _errHandler.recoverInline(this);
            }
            else {
               if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
               _errHandler.reportMatch(this);
               consume();
            }
            setState(72);
            match(T__10);
            setState(73);
            _la = _input.LA(1);
            if ( !(_la==TRUE || _la==FALSE) ) {
            _errHandler.recoverInline(this);
            }
            else {
               if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
               _errHandler.reportMatch(this);
               consume();
            }
            }
         }
         catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
         }
         finally {
            exitRule();
         }
         return _localctx;
      }

      public static class PathesContext extends ParserRuleContext {
         public List<PathContext> path() {
            return getRuleContexts(PathContext.class);
         }
         public PathContext path(int i) {
            return getRuleContext(PathContext.class,i);
         }
         public PathesContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
         }
         @Override public int getRuleIndex() { return RULE_pathes; }
         @Override
         public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterPathes(this);
         }
         @Override
         public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitPathes(this);
         }
         @Override
         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
```

```
T>)visitor).visitPathes(this);
         else return visitor.visitChildren(this);
      }
   }

   public final PathesContext pathes() throws RecognitionException {
      PathesContext _localctx = new PathesContext(_ctx, getState());
      enterRule(_localctx, 14, RULE_pathes);
      int _la;
      try {
         enterOuterAlt(_localctx, 1);
         {
         setState(75);
         match(T__11);
         setState(79);
         _errHandler.sync(this);
         _la = _input.LA(1);
         while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << DAO) | (1L <<
CRUD) | (1L << KOIN) | (1L << DATABASE) | (1L << DATABASE_PACKAGE))) != 0)) {
            {
            {
            setState(76);
            path();
            }
            }
            setState(81);
            _errHandler.sync(this);
            _la = _input.LA(1);
         }
         setState(82);
         match(T__1);
         }
      }
      catch (RecognitionException re) {
         _localctx.exception = re;
         _errHandler.reportError(this, re);
         _errHandler.recover(this, re);
      }
      finally {
         exitRule();
      }
      return _localctx;
   }

   public static class PathContext extends ParserRuleContext {
      public TerminalNode PATH_LITERAL() { return
getToken(EDSLParser.PATH_LITERAL, 0); }
      public TerminalNode DAO() { return getToken(EDSLParser.DAO, 0); }
      public TerminalNode CRUD() { return getToken(EDSLParser.CRUD, 0); }
      public TerminalNode KOIN() { return getToken(EDSLParser.KOIN, 0); }
      public TerminalNode DATABASE() { return getToken(EDSLParser.DATABASE, 0);
}
      public TerminalNode DATABASE_PACKAGE() { return
getToken(EDSLParser.DATABASE_PACKAGE, 0); }
      public PathContext(ParserRuleContext parent, int invokingState) {
         super(parent, invokingState);
      }
      @Override public int getRuleIndex() { return RULE_path; }
      @Override
      public void enterRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).enterPath(this);
```

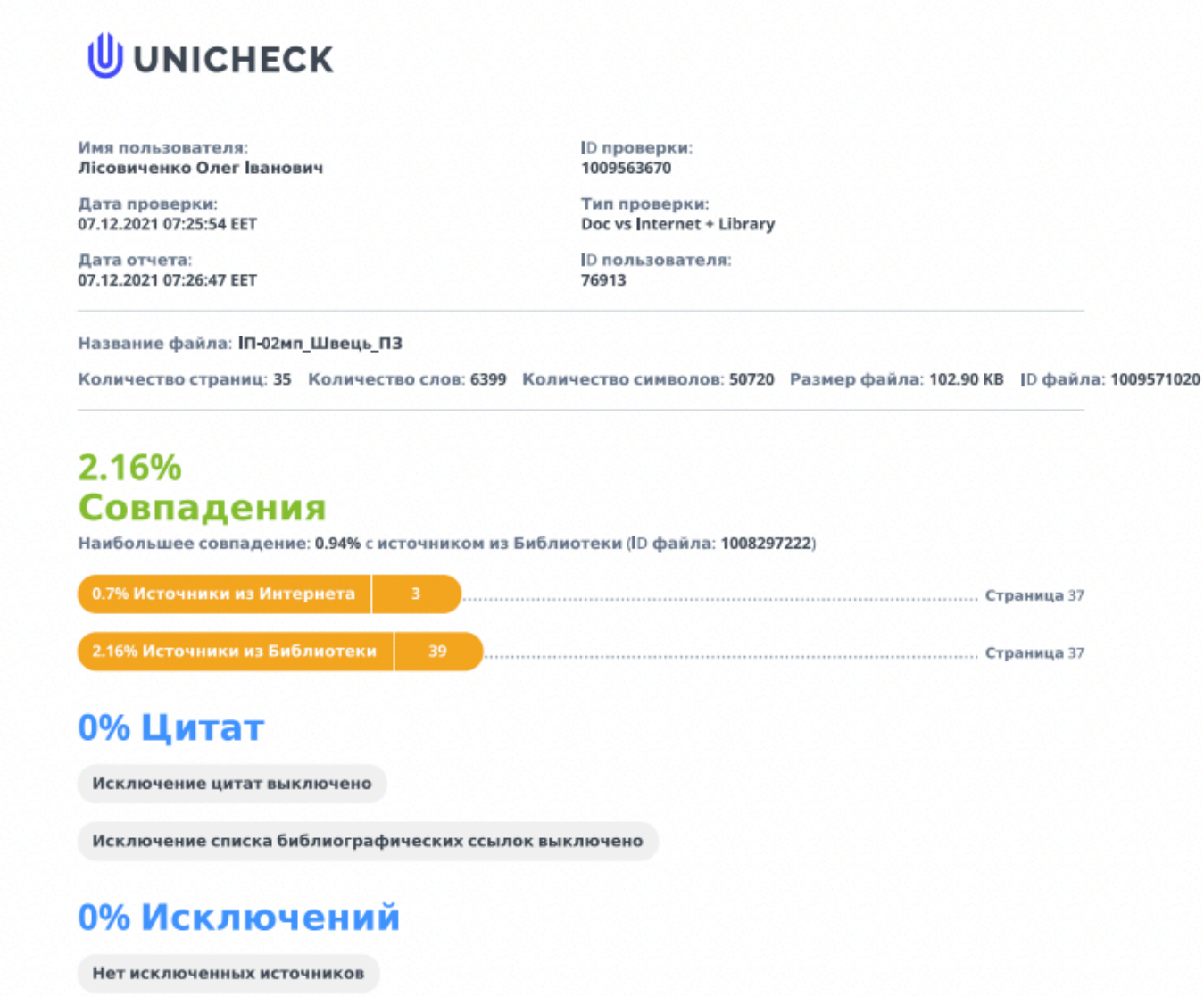
```
      }
      @Override
      public void exitRule(ParseTreeListener listener) {
         if ( listener instanceof EDSLListener)
((EDSLListener)listener).exitPath(this);
      }
      @Override
      public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
         if ( visitor instanceof EDSLVisitor) return ((EDSLVisitor<? extends
T>)visitor).visitPath(this);
         else return visitor.visitChildren(this);
      }
   }

   public final PathContext path() throws RecognitionException {
      PathContext _localctx = new PathContext(_ctx, getState());
      enterRule(_localctx, 16, RULE_path);
      int _la;
      try {
         enterOuterAlt(_localctx, 1);
         {
         setState(84);
         _la = _input.LA(1);
         if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << DAO) | (1L <<
CRUD) | (1L << KOIN) | (1L << DATABASE) | (1L << DATABASE_PACKAGE))) != 0)) ) {
         _errHandler.recoverInline(this);
         }
         else {
            if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
            _errHandler.reportMatch(this);
            consume();
         }
         setState(85);
         match(T__10);
         setState(86);
         match(PATH_LITERAL);
         }
      }
      catch (RecognitionException re) {
         _localctx.exception = re;
         _errHandler.reportError(this, re);
         _errHandler.recover(this, re);
      }
      finally {
         exitRule();
      }
      return _localctx;
   }

   public static final String _serializedATN =
      "\3\u608b\ua72a\u8133\ub9ed\u417c\u3be7\u7786\u5964\3\37[\4\2\t\2\4\3\t"+
      "\3\4\4\t\4\4\5\t\5\4\6\t\6\4\7\t\7\4\b\t\b\4\t\t\t\4\n\t\n\3\2\3\2\3\3"+
      "\3\3\3\3\3\3\3\3\3\3\4\3\4\6\4\37\n\4\r\4\16\4 \3\4\3\4\3\5\3\5\3\5"+
      "\6\5(\n\5\r\5\16\5)\3\5\3\5\3\6\5\6/\n\6\3\6\3\6\3\6\3\6\3\6\3\6\3\6\5\6\67"+
      "\n\6\3\6\3\6\3\6\3\6\3\6\5\6=\n\6\3\6\3\6\3\7\3\7\7\7\7C\n\7\f\7\16\7F\13\7"+
      "\3\7\3\7\3\b\3\b\3\b\3\b\3\t\3\t\7\tP\n\t\f\t\16\tS\13\t\3\t\3\t\3\n\3"+
      "\n\3\n\3\n\3\n\2\2\13\2\4\6\b\n\f\16\20\22\2\6\3\2\17\21\3\2\22\24\3\2"+
      "\27\30\3\2\22\26\2X\2\24\3\2\2\2\4\26\3\2\2\2\6\34\3\2\2\2b$\3\2\2\2"+
      "\n\66\3\2\2\2f@\3\2\2\2\16I\3\2\2\2\20M\3\2\2\2\22V\3\2\2\2\24\25\5\4"+
      "\3\2\25\3\3\2\2\2\26\27\7\3\2\2\27\30\5\6\4\2\30\31\5\f\7\2\31\32\5\20"+
      "\t\2\32\33\7\4\2\2\33\5\3\2\2\2\34\36\7\5\2\2\35\37\5\b\5\2\36\35\3\2"+
      "\2\37 \3\2\2\2 \36\3\2\2\2 !\3\2\2\2!\"\3\2\2\2\"#\7\4\2\2#\7\3\2\2"+
```

```
        "\2$%\7\33\2\2%\'\7\6\2\2&(\5\n\6\2\'&\3\2\2\2()\3\2\2\2)\'\3\2\2\2)*\3"+
        "\2\2\2*+\3\2\2\2+,\7\4\2\2,\t\3\2\2\2-/\7\31\2\2.-\3\2\2\2./\3\2\2\2/"+
        "\67\3\2\2\2\60\61\7\32\2\2\61\62\7\7\2\2\62\63\7\33\2\2\63\64\7\b\2\2"+
        "\64\65\7\33\2\2\65\67\7\t\2\2\66.\3\2\2\2\66\60\3\2\2\2\678\3\2\2\289"+
        "\7\33\2\29:\7\n\2\2:<\t\2\2\2;=\7\36\2\2<;\3\2\2\2<=\3\2\2\2=>\3\2\2\2"+
        ">?\7\13\2\2?\13\3\2\2\2@D\7\f\2\2AC\5\16\b\2BA\3\2\2\2CF\3\2\2\2DB\3\2"+
        "\2DE\3\2\2\2EG\3\2\2\2FD\3\2\2\2GH\7\4\2\2H\r\3\2\2\2IJ\t\3\2\2JK\7"+
        "\r\2\2KL\t\4\2\2L\17\3\2\2\2MQ\7\16\2\2NP\5\22\n\2ON\3\2\2\2PS\3\2\2\2"+
        "QO\3\2\2\2QR\3\2\2\2RT\3\2\2\2SQ\3\2\2\2TU\7\4\2\2U\21\3\2\2\2VW\t\5\2"+
        "\2WX\7\r\2\2XY\7\34\2\2Y\23\3\2\2\2\t ).\66<DQ";
    public static final ATN _ATN =
        new ATNDeserializer().deserialize(_serializedATN.toCharArray());
    static {
        _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
        for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
            _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
        }
    }
}
```

ДОДАТОК В РЕЗУЛЬТАТИ ПЕРЕВІРКИ РОБОТИ НА СПІВПАДІННЯ



UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009563670

Дата проверки:
07.12.2021 07:25:54 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
07.12.2021 07:26:47 EET

ID пользователя:
76913

Название файла: ІП-02мп_Швець_ПЗ

Количество страниц: 35 Количество слов: 6399 Количество символов: 50720 Размер файла: 102.90 KB ID файла: 1009571020

2.16%
Совпадения

Наибольшее совпадение: 0.94% с источником из Библиотеки (ID файла: 1008297222)

0.7% Источники из Интернета | 3 ..... Страница 37

2.16% Источники из Библиотеки | 39 ..... Страница 37

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников