КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА РОЗШИРЕННЯ PYCHARM IDE ДЛЯ ПІДТРИМКИ QML

Виконав студент 4-го курсу
Дмитро СУКОВАНЧЕНКО

_____
(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Костянтин ЖЕРЕБ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

_____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол №11
Завідувач кафедри
Олександр ПРОВОТАР

_____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 50 сторінок, 29 ілюстрації, 10 використаних джерел, 2 додатки

QML, ПАРСЕР, СИНТАКСИЧНИЙ АНАЛІЗ, SDK, PYCHARM, РОЗШИРЕННЯ IDE

Об’єктом роботи є побудова розширення для роботи з PyCharm IDE.

Метою кваліфікаційної роботи є створення власного засобу для підтримки QML з інтегруванням в середовище розробки, а також доповненням необхідним функціоналом, а саме підсвітка кольором тексту, в залежності від контексту, аналіз помилок, та інше.

Середовище розробки: IntelliJ Platform SDK — інструмент для створення розширень для платформи Jetbrains. Grammar-Kit — плагін для побудови лексеру та парсеру. Intellij IDEA 2023.1 — IDE в якому проводиться робота та відладка отриманих результатів. Мова програмування — Java.

Результати роботи: створено розширення для роботи з QML файлами, з необхідним фунціоналом, а також можливостю доповнення. Сформовано план для майбутнього удосконалення розробки.

Сфера застосування може бути будь-яким проектом, що розробляється мовою QML, а саме, мобільні застосунки, застосунки для персонального комп’ютера чи розробки для вбудованих систем з інтерфейсом користувача.

Плани подальшого розвитку розробленого додатку, включають доопрацювання деяких аспектів, а також можливе додавання чи видозміна існуючого функціоналу.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AST — Abstract Syntax Tree, абстрактне синтаксичне дерево яке має утворитись в результаті парсингу мови;

GUI — Graphical User Interface, візуальний інтерфейс, що дозволяє користувачам взаємодіяти з програмами за допомогою графічних елементів;

IDE — Integrated development environment, комплексне програмне рішення для розробки програмного забезпечення;

IntelliJ IDEA — IDE для розробки на мові Java від компанії JetBrains;

PSI Program Structure Interface, інтерфейс структури програми, є рівнем у платформі IntelliJ, відповідальним за розбір файлів і створення синтаксичної та семантичної моделі коду, яка забезпечує багато функцій платформи;

PyCharm — IDE для розробки на мові Python від компанії JetBrains;

Qt — кросплатформовий фреймворк для розробки GUI;

QML — Qt Modeling Language, мова для опису графічного інтерфейсу від компанії Qt;

SDK — Software development kit, комплект розробницьких інструментів, утиліт та документації;

UI — User Interface, те, як користувач візуально бачить продукт;

UX — User Experience, функціональні можливості для взаємодії користувача;

ПЗ — Програмне забезпечення.

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. Сучасний світ розробки інформаційних технологій майже неможливий без використання IDE (Integrated development environment), які значно спрощують процес створення програмного коду. Одним із найбільш популярних та функціональних IDE для розробки на Python є PyCharm. Його зручність, швидкість і потужність інструментів дозволяють програмістам ефективно реалізовувати свої проекти. В той же час, мова програмування QML дедалі більше набирає популярності у розробницькому середовищі, особливо при створенні графічних інтерфейсів для звичайних та мобільних додатків. Мова дозволяє створювати візуально привабливі, гнучкі та високопродуктивні інтерфейси, але підтримка QML в PyCharm на даний момент є обмеженою.

Актуальність роботи та підстави для її виконання. Актуальність теми полягає в створенні зручного розширення яке буде на рівні вже існуючих рішень та допоможе з розробкою на Python різних застосунків. Реалізація такого розширення забезпечить можливість комфортної і продуктивної роботи з QML безпосередньо в PyCharm, що зробить розробку на цій мові більш доступною для широкого кола програмістів, особливо тих, хто вже звик до роботи в PyCharm. Крім того, тема вимагає дослідження таких аспектів, як розробка розширень для IDE, робота з IntelliJ Platform Plugin SDK (Software development kit) та адаптація в PyCharm мови QML для роботи вдосконалення процесу розробки.

З урахуванням швидкого розвитку технологій та збільшення складності програмного забезпечення, інтеграція QML в PyCharm стане важливим кроком у забезпеченні продуктивності та ефективності роботи розробників. Отримане розширення зможе слугувати вихідним пунктом для подальшого

вдосконалення, а також стимулюватиме розвиток та адаптацію інших мов програмування в PyCharm.

Вивчення та реалізація даної теми є важливим та актуальним вкладом у розвиток програмної інженерії та технологій розробки програмного забезпечення.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення власного засобу для підтримки QML з інтегруванням в середовище розробки, а також доповненням необхідним функціоналом, а саме підсвітка кольором тексту, в залежності від контексту, аналіз помилок, та інше. Для досягнення цієї мети поставлено такі завдання:

— проаналізувати сучасні методи роботи з QML в PyCharm та інших IDE, включаючи наявні розширення та плагіни;

— визначити ключові вимоги до нового розширення, зокрема функціональність, сумісність та продуктивність;

— розробити архітектуру нового розширення, включаючи визначення основних компонентів та їх взаємодію;

— реалізувати розширення згідно з визначеною архітектурою та вимогами;

— провести тестування розширення для виявлення та виправлення помилок та недоліків;

— провести оцінку ефективності розширення та його впливу на продуктивність розробки.

Об’єкт, методи й засоби розроблення. Об’єктом роботи є побудова розширення для роботи з PyCharm IDE. Предметом дослідження є процес розробки розширення PyCharm для підтримки програмування на QML.

Можливі сфери застосування. Розроблений плагін для підтримки QML в PyCharm може мати широкий спектр застосувань в різних сферах, зокрема: розробка десктопних застосунків, мобільна розробка, розробка вбудованих систем та ігрова розробка. Загалом, може знайти своє

застосування в будь-якій сфері, де використовується QML для створення UI(User Interface) та UX (User Experience) дизайну.

РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ

1.1 Введення в IDE та їх роль в програмуванні

Програмне забезпечення, яке надає програмістам всеосяжне середовище для розробки програмного забезпечення називають IDE (Integrated development environment). Воно об'єднує різні інструменти, необхідні для розробки ПЗ (Програмне забезпечення), включаючи редактор коду, інструменти для побудови та відлагоджувач, в одному інтерфейсі. Використання IDE може поліпшити продуктивність розробника, спростивши процес написання, тестування та інше.

Різні середовища розробки можна поділяються на деякі категорії залежно від мови програмування, яку вони підтримують, платформи, на якій вони працюють, та особливостей, які вони надають, розміщення, де саме можна працювати із середовищем, а також моделлю розповсюдження. Нижче наведено кілька основних прикладів:

— за мовою програмування: часто оптимізуються під конкретну мову програмування. Наприклад, PyCharm використовується для Python, IntelliJ IDEA для Java, а Visual Studio для C#, VB.NET і C++. Деякі IDE, як-от Eclipse або NetBeans, можуть підтримувати багато мов за допомогою плагінів;

— за платформою: деякі IDE розроблені спеціально для розробки під певні платформи. Наприклад, Android Studio для розробки Android-додатків, Xcode для iOS/MacOS аплікацій;

— за особливостями: деякі IDE надають спеціалізовані інструменти або можливості, такі як підтримка веб-розробки, мобільної розробки, розробки вбудованих систем або наукового обчислення. Наприклад, RStudio надає спеціалізовані інструменти для розробки в R, мови, що широко використовується в наукових та статистичних обчисленнях;

— за розміщенням: IDE можуть бути розміщені локально на комп'ютері розробника або в облаку. Хмарні IDE, такі як Repl.it, Google Colaboratory або Cloud9, надають доступ до середовища розробки через веб-браузер, що дозволяє розробникам працювати з будь-якого комп'ютера з підключенням до Інтернету;

— за моделлю розповсюдження: можуть бути як безкоштовними, так і платними, або ж мати комбіновану модель розповсюдження. Безкоштовні IDE, такі як Eclipse або Visual Studio Code, надають масу функціональності безкоштовно, в той час як платні IDE, наприклад, IntelliJ IDEA або PyCharm Professional, часто пропонують додаткові преміум-функції або кращу підтримку. Комбіновані моделі, якими також є JetBrains, надають базову версію безкоштовно, але мають платну версію з додатковими можливостями.

Ці категорії не є взаємовиключними, і багато IDE можуть відноситися до декількох категорій одночасно. Наприклад, PyCharm є IDE, оптимізованим під Python (за мовою програмування), працює на декількох платформах, включаючи Windows, Linux та MacOS (за платформою), має додаткові інструменти для веб-розробки та наукового обчислення (за особливостями), і в той же час доступний в якості локального застосунку.

Зосередимось саме на PyCharm, бо він предназначен для мови Python, однією з найпопулярнішою мов програмування на сьогодні. Згідно статистики саме цю мову найчастіше шукають початківці (Рисунок 1.1)

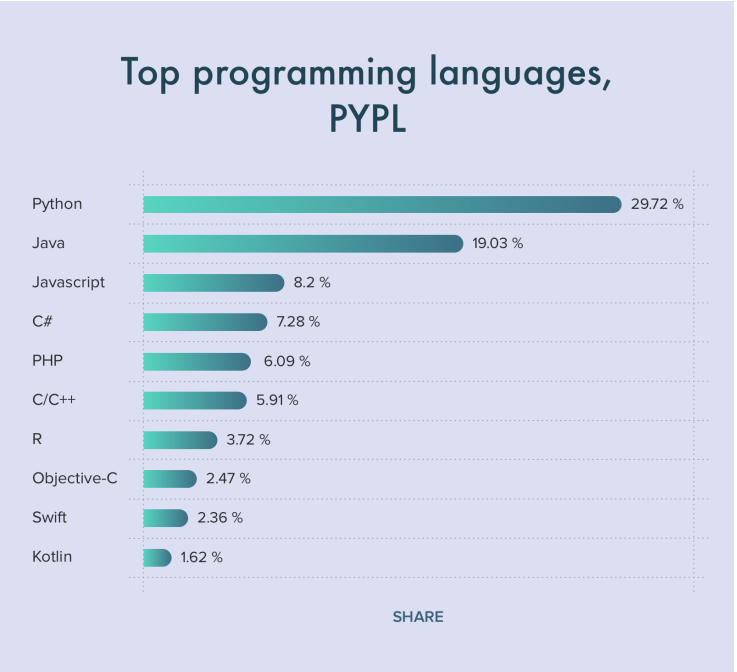

Рисунок 1.1 — Популярність мов програмування [1]

PyCharm це потужне середовище розробки для мови Python. PyCharm надає розробникам широкий набір інструментів, необхідних для ефективної роботи, а саме:

— редактор коду: IDE містить вбудований редактор коду, який підтримує синтаксичне виділення, автоматичне вирівнювання та інші функції, які спрощують процес написання коду. Редактор також включає функцію автозавершення коду, яка пропонує можливі варіанти на основі контексту, що допомагає зекономити час та зменшує ймовірність помилок;

— відлагоджувач (debugger): має потужний відлагоджувач, який дозволяє перевіряти та налагоджувати свій код. Надає можливість ставити точки зупинки, переглядати значення змінних під час виконання та керувати процесом виконання;

— інтеграція з системами керуванням версіями: PyCharm інтегрується з популярними системами керуванням версіями, включаючи

Git, SVN, Mercurial, тощо. Ця інтеграція дає змогу виконувати основні операції з керуванням версіями безпосередньо з IDE;

— підтримка веб-розробки: PyCharm підтримує веб-розробку з використанням Python-фреймворків, таких як Django і Flask. Він також включає вбудовані інструменти для роботи з HTML, CSS і JavaScript;

— інтегроване тестування: PyCharm надає широкі можливості для тестування коду на Python. Воно підтримує різні фреймворки для тестування, включаючи unittest, pytest та doctest, і дозволяє запускати тести безпосередньо з IDE;

— інтеграція з базами даних: PyCharm має вбудовані інструменти для роботи з базами даних. Він підтримує популярні системи управління базами даних, включаючи MySQL, PostgreSQL, SQLite і інші.

Узагальнюючи, PyCharm — це всебічний інструмент, який значно полегшує процес розробки на Python. Його різноманітні функції підтримують всі етапи розробки ПЗ, включаючи написання коду, тестування, відлагодження, контроль версій і взаємодію з базами даних, що робить його незамінним помічником для програмістів, що працюють з Python.

1.2 Огляд мови програмування QML та її особливостей

QML (Qt Modeling Language) – це мова декларативного програмування для дизайну UI, розроблена компанією Qt [2]. Вона використовується для описування ієрархії об'єктів та їх взаємодії, і є основою для QtQuick, графічного фреймворку для розробки мобільних та вбудованих додатків. Мовою базується на JSON з додатковими функціями, як-от вбудовані типи

даних, підтримка наслідування та зв'язування даних. Однією з ключових особливостей QML є її здатність легко інтегруватися з JavaScript і C++, а також Python, що дає можливість розробникам використовувати найкращі характеристики цих мов при створенні додатків.

Завдяки декларативному підході в мові розробники можуть описувати, ЩО повинно бути зроблено, а не ЯК це потрібно робитись. Це дозволяє зосередитися на високорівневому дизайні інтерфейсу користувача без необхідності занурюватися в деталі низькорівневого програмування графіки. Загальну структуру базових класів Qt можна подивитись нижче (Рисунок 1.2). Для QML використовуються саме QtQuick елементи, якими можна задовольнити базові потреби в UI/UX дизайні.

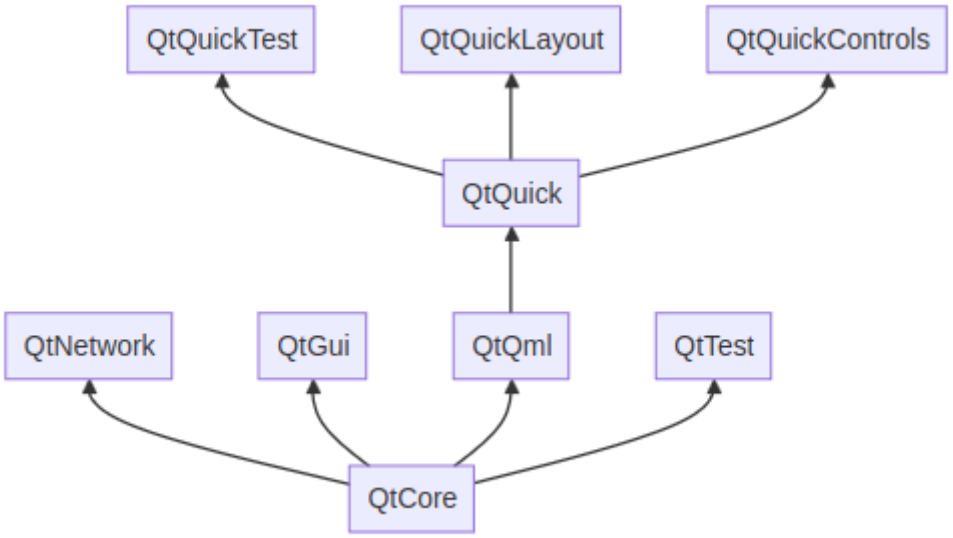

Рисунок 1.2 — Схема наслідування основних елементів Qt [2]

Загальною схемою під час розробки інтерфейсів користувача є відокремлення представлення даних від візуалізації. Це дає змогу відображати одні й ті самі дані різними способами залежно від того, яке завдання виконує користувач. Наприклад, телефонна книга може бути організована як вертикальний список текстових записів або як сітка

зображень контактів. В обох випадках дані ідентичні — телефонна книга, але різна візуалізація. Цей поділ зазвичай називають патерном модель-представлення. У ньому дані називаються моделлю, тоді як візуалізація обробляється представленням (Рисунок 1.3 б).

У QML модель і представлення об’єднані делегатом. Обов'язки розподілені таким чином: модель надає дані. Для кожного елемента даних може бути кілька значень. У наведеному прикладі з телефоною книгою кожен запис має ім’я, зображення та номер. Дані впорядковуються в поданні, у якому кожен елемент візуалізується за допомогою делегата. Завдання представлення полягає в тому, щоб упорядкувати делегати, у той час як кожен делегат показує значення кожного елемент моделі для користувача. Це означає, що делегат знає про вміст моделі та про те, як її візуалізувати. Подання знає про концепцію делегатів і про те, як їх розмістити. Модель знає лише про дані, які вона представляє.

Простий приклад QML коду, а також згаданий патерн, зображено нижче (Рисунок 1.3 а)

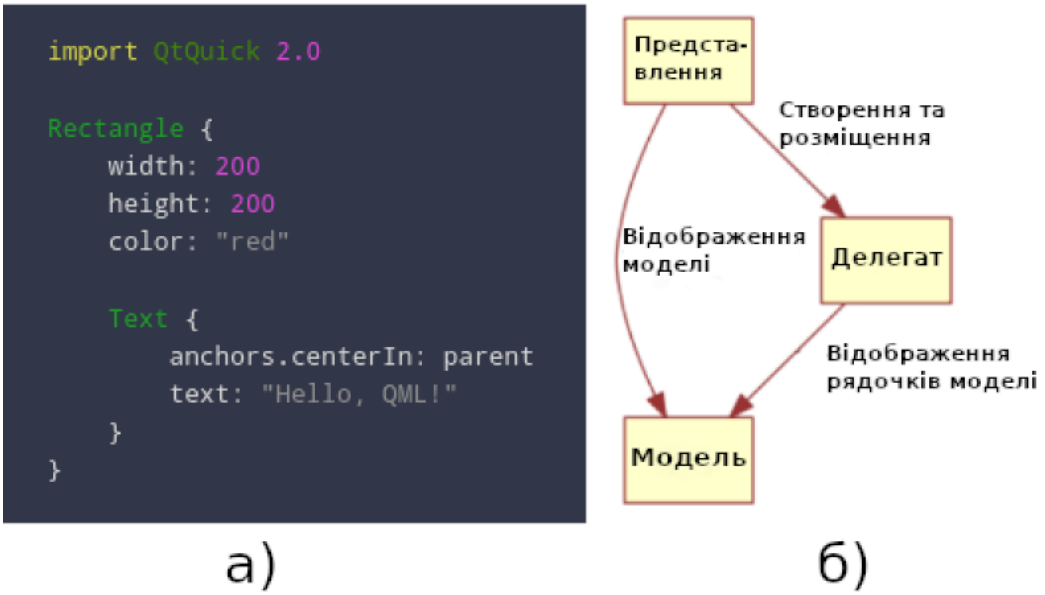

Рисунок 1.3 — Основи роботи з QML: а — приклад коду; б — паттерн модель-представлення-делегат (Model-View-Delegate)

Мова QML включає в себе наступні концепції:

— елементи: елементи в QML це базові блоки для створення QML-додатків. Вони включають прості компоненти, такі як Rectangle (для створення прямокутників), Text (для відображення тексту) та Image (для відображення зображень), а також складніші компоненти, такі як ListView (для створення списків елементів) або Flickable (для створення гортання елементів). Є також ті, що контролюють анімацію, інтерполяцію та інші аспекти динаміки інтерфейсу;

— властивості: властивості дозволяють контролювати характеристики та поведінку елементів. Наприклад, властивість color в елемента Rectangle контролює колір прямокутника, властивість text в елемента Text контролює відображуваний текст, та властивість source в елемента Image визначає зображення, що відображається. Властивості можна зв'язувати, що означає, що зміна однієї властивості автоматично викликає зміну іншої;

— сигнали і слоти: механізм, який дозволяє об'єктам спілкуватися між собою в QML. Об'єкт відправляє сигнал, коли відбувається певна подія, а інший об'єкт може реагувати на цей сигнал, використовуючи слот. Наприклад, кнопка може відправити сигнал, коли її натискають, а слот може бути використаний для реагування на цей сигнал, наприклад, змінюючи колір кнопки;

— анімація: QML надає багато можливостей для створення анімацій. Анімація може бути додана до будь-якої властивості, що може змінюватись в часі. Це може бути, наприклад, розмір, положення, колір або прозорість елемента. QML має потужні механізми для контролю над тим, як відбувається анімація, включаючи інтерполяцію, ефекти затухання, затримки, послідовної та паралельної анімації;

— інтеграція з JavaScript, C++ та Python: мова має підтримку JavaScript, що дозволяє виконувати динамічні обчислення та маніпуляції з даними безпосередньо в QML-коді. Це може включати такі речі, як обробка дій користувача, виконання математичних обчислень або робота з даними JSON. Щодо C++ та Python, QML може інтегруватися через спеціальний API, що дозволяє коду експонувати об'єкти та функції до QML та викликати QML-функції з іншого місця;

— зв'язування даних: одна з сильних сторін мови це зв'язування даних. Коли властивість об'єкта QML зв'язується з виразом, вираз обчислюється автоматично, кожен раз, коли змінюються вхідні дані. Це означає, що вам не потрібно явно оновлювати властивості, коли змінюється їхній залежний стан - замість цього, система автоматично оновлює властивості за вас.

Загалом, QML призначено для створення невеликих до середніх, швидкодіючих, графічно вимогливих програм. Вона пропонує багато привабливих можливостей для розробників, що хочуть створити ефективні, реактивні та красиві інтерфейси користувача.

1.3 Аналіз існуючих рішення для роботи з QML

Розробка інтерфейсів користувача за допомогою QML вже підтримується декількома IDE та текстовими редакторами. Давайте розглянемо деякі з них:

— Qt Creator: основний редактор для розробки на QML, наданий компанією, яка власне створила QML. IDE має вбудовану підтримку QML з підсвічуванням синтаксису, автодоповненням, візуальним редактором

інтерфейсу та іншими інструментами. Проте із недоліків в даному випадку можна виділити відсутність підтримки python та інших корисних плагінів, які можна було б імпортувати в популярних редакторах. Qt Creator підлаштовний для розробки з C++ кодом, яка є більш природною для нього. Дизайн застосунку можна побачити нижче (Рисунок 1.4);

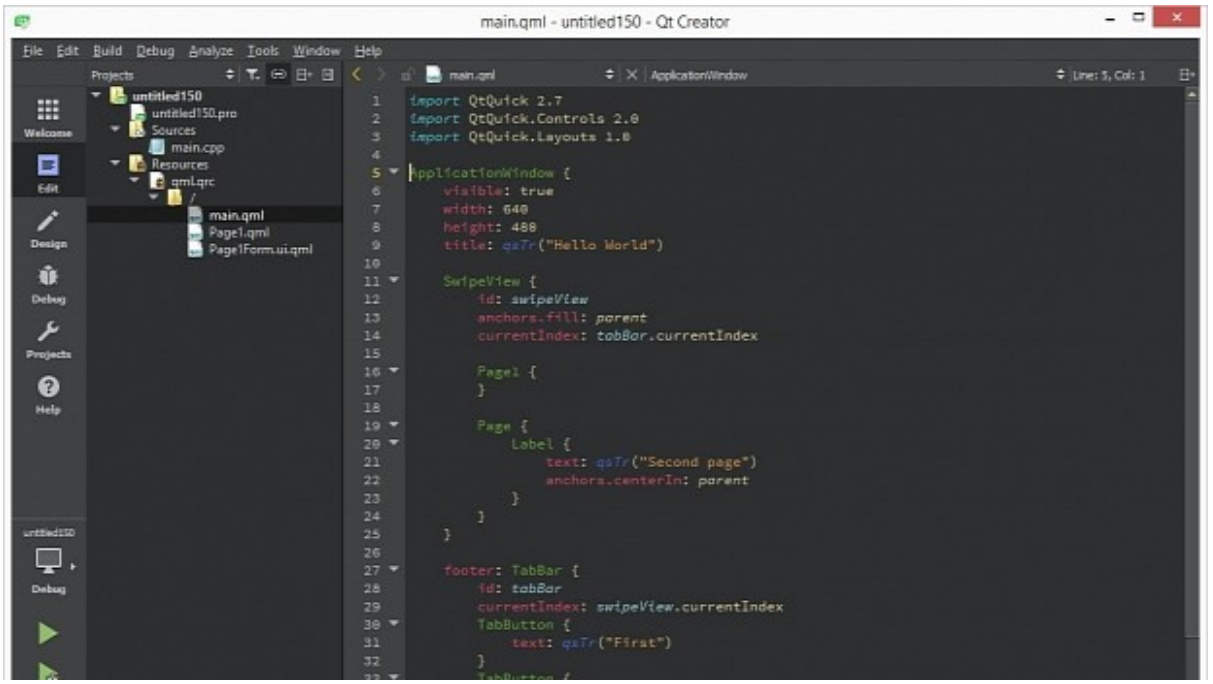

Рисунок 1.4 — Вигляд Qt Creator

— Visual Studio Code з розширенням QML: Visual Studio Code має декілька розширень для підтримки QML, включаючи підсвічування синтаксису, автодоповнення та перегляд документації. IDE досить популярна, та презентує себе більше як набір різних плагінів для універсальної роботи з текстом, аніж щось конкретного. Оцінки [3] розширення для QML в магазині розширень VS Code змішані, з деякими користувачами, які скаржаться на недостатню підтримку автозаповнення і відсутність деяких ключових функцій IDE. Вигляд фрагменту коду можна подивитись на зображені нижче (Рисунок 1.5);

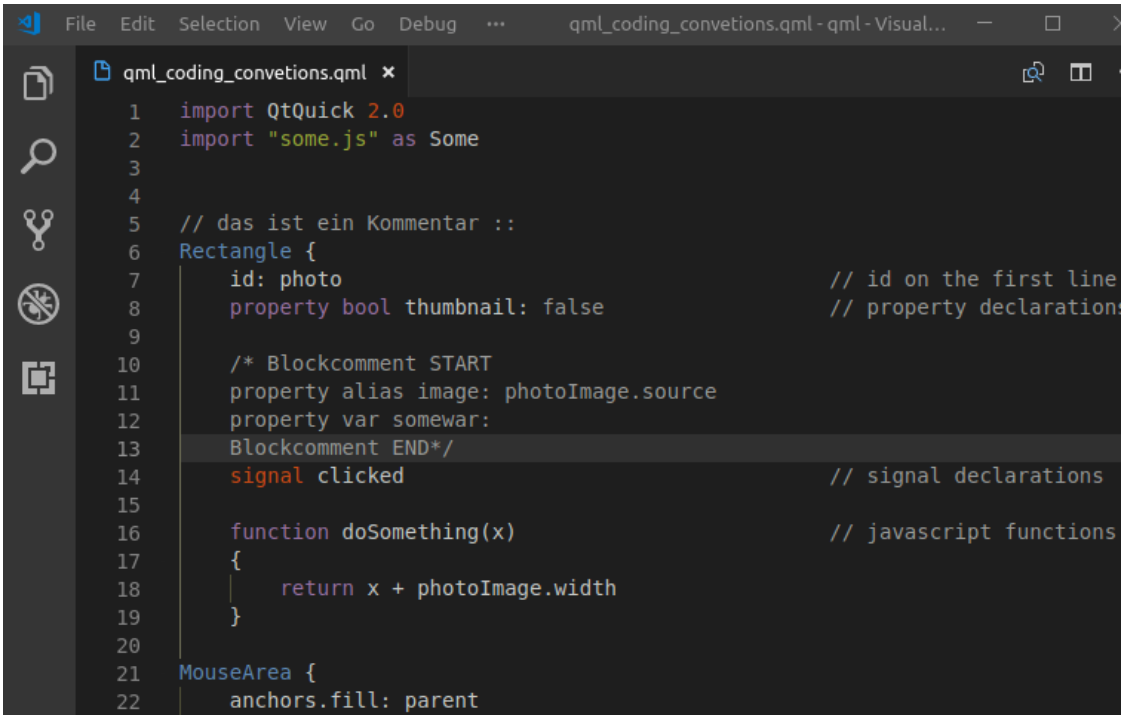

Рисунок 1.5 — Вигляд Visual Studio Code з розширенням QML

— Sublime Text з плагіном QML: Sublime Text також має плагіни для підтримки QML, які надають підсвічування синтаксису і автодоповнення. Sublime завжди славився своєю швидкістю і мінімалістичним дизайном. Його пакет QML було впроваджено давно і отримало позитивні відгуки, але, як і з розширенням QML VS Code, користувачі вказували на відсутність деяких ключових функцій IDE;

— QmlEditor плагін для PyCharm: вже існує розширення для платформи JetBrains, яке надає підтримку QML, та поки воно єдине. Крім того воно платне та відгуки користувачів вказують на ряд проблем, які обмежують його ефективність та загальну корисність, маючи рейтинг 2 зірочок з 5 згідно з [5]. Основні проблеми, вказані користувачами, включають:

а) обмеженість автодоповнення: користувачі скаржаться, що функція автодоповнення коду не працює так, як вони очікували, часто не вдається вгадати використовувані вирази або їх частини;

б) проблеми зі стабільністю та продуктивністю: деякі користувачі зазначають, що плагін може спричиняти збої IDE або уповільнити його роботу;

в) недостатній синтаксичний аналіз та підсвічування: користувачі вказують на проблеми з розумінням QML синтаксису плагіном, що призводить до неточного або відсутнього підсвічування коду.

Отже, розробка власного розширення для PyCharm має ряд переваг. Інтеграція з середовищем PyCharm дозволить розробникам, які вже використовують його для Python, працювати з QML без перемикання на інше середовище, що покращить їхню продуктивність. Доповнення буде сумісність з вже існуючими інструментами, таких як налагоджувач, профілер, система контролю версій, інструменти для рефакторингу та інші, які будуть доступні і для роботи з QML.

Створення власного розширення дозволяє адаптувати PyCharm для специфічних потреб розробки на QML. Також публікація цього розширення може сприяти зростанню спільноти розробників QML, надаючи їм нові інструменти для роботи. Це може стимулювати розвиток QML як мови в комбінації з Python, оскільки більша підтримка та доступність можуть привернути більше розробників. Власна розробка розширення дозволяє робити код відкритим для інших розробників, що також може сприяти збільшенню кількості людей, які можуть запропонувати вдосконалення розширення, що забезпечує його постійне покращення та адаптацію до нових вимог і технологій.

Крім того, розробка власного розширення PyCharm для підтримки QML є не тільки важливим кроком для покращення середовища розробки для даної мови, але також може мати позитивний вплив на спільноту розробників, забезпечуючи їм нові можливості та інструменти для ефективної роботи. Важливим моментом для розробки є врахування вже існуючих відгуків та запропонувати вирішення відомих проблем. Це може включати покращення функціональності автодоповнення, покращення стабільності та продуктивності, а також покращення синтаксичного аналізу та підсвічування коду.

РОЗДІЛ 2 АНАЛІЗ INTELLIJ PLATFORM PLUGIN SDK

2.1 Архітектура та компоненти SDK

IntelliJ Platform Plugin SDK [6] служить основою для створення розширень для всієї сім'ї продуктів JetBrains, включаючи PyCharm. Цей SDK містить різні компоненти, які допомагають розробникам швидко і ефективно створювати нові розширення.

В основі SDK лежить потужна база IDE, що містить багато сервісів та компонентів, що спрощують розробку розширень. Ці компоненти вже включають редактор коду, систему керування проектами, інтеграцію з системами керуванням версіями і багато інших інструментів, які можуть використовуватися розробниками плагінів.

Один з ключових компонентів це його API для розширення. Воно надає розробникам можливість розширювати та налаштовувати функціональність IDE, адаптуючи його до специфічних потреб користувачів. API охоплює широкий спектр функцій, включаючи зміну коду, його виконання, налагодження, тестування, інтеграцію з іншими сервісами та інструментами, і багато іншого. Крім того, SDK містить бібліотеки для роботи з графічними інтерфейсами, роботи з даними, мережевими операціями, і багато іншого. Це означає, що розробники плагінів можуть використовувати ці бібліотеки для створення більш продуктивних, ефективних та користувацьких плагінів, що можуть охопити будь які побажання.

За замовчуванням SDK налаштований на певну архитектуру проекту при створенні, зображену в додатку Б. До неї входить build.gradle.kts, який є файлом конфігурації, використовуваним Gradle, системою автоматизації збірки. Цей файл містить інструкції, які описують, як зібрати і тестувати проект. Файл build.gradle.kts може включати залежності, які необхідні для проекту, і вказувати, які завдання необхідно виконати для збірки.

META-INF/plugin.xml — це манифест плагіну (приклад на Рисунку 2.1), який описує основну інформацію про плагін, включаючи його назву, версію, опис, вимоги до сумісності та так далі. Він також містить список розширень, які надає плагін, і точки розширення, які використовує плагін. Розширення — це ключовий механізм, який дозволяє плагінам розширювати або модифікувати функціональність IntelliJ Platform. Кожне розширення відноситься до певної точки розширення, яка визначає, який тип функціональності може бути розширено або змінено.

Java-файли, які підключаються до розширень, використовуються для реалізації цієї додаткової або зміненої функціональності. Наприклад, якщо плагін хоче розширити функціональність автодоповнення коду, можна створити Java-клас, який реалізує відповідний інтерфейс, і підключити цей клас до відповідної точки розширення в plugin.xml. Коли IntelliJ Platform завантажує плагін, вона читає plugin.xml, визначає, які розширення надає плагін, і динамічно підключає відповідні Java-класи до системи. Ця архітектура дозволяє плагінам розширювати та модифікувати практично будь-який аспект IntelliJ Platform без зміни основного коду платформи.

Grammar-Kit / resources / META-INF / plugin-java.xml

gregsh i18n: initial 6feca3e · 6 months ago History

Code Blame 28 lines (26 loc) · 1.67 KB Raw

```xml
<?xml version="1.0" encoding="utf-8"?>
<idea-plugin>
  <resource-bundle>messages.GrammarKitBundle</resource-bundle>
  <extensions defaultExtensionNs="com.intellij">
    <projectService serviceInterface="org.intellij.grammar.java.JavaHelper" serviceImplementation="org.intellij.grammar.java.JavaHelper$PsiHelper"/>
    <referencesSearch implementation="org.intellij.jflex.psi.impl.JFlexStateUsageSearcher"/>
    <regExpLanguageHost forClass="org.intellij.grammar.psi.impl.BnfStringImpl" implementationClass="org.intellij.grammar.psi.impl.BnfStringRegexHost"/>

    <multiHostInjector implementation="org.intellij.jflex.psi.impl.JFlexJavaCodeInjector"/>
    <lang.elementManipulator forClass="org.intellij.jflex.psi.impl.JFlexJavaCodeImpl"
                             implementationClass="org.intellij.jflex.psi.impl.JFlexJavaCodeManipulator"/>

    <notificationGroup id="grammarkit.parser.generator.log" displayType="NONE" key="notification.group.parser.generator.log"/>
  </extensions>
  <actions>
    <group id="grammar.group">
      <action id="grammar.Generate" class="org.intellij.grammar.actions.GenerateAction">
        <keyboard-shortcut keymap="$default" first-keystroke="control shift G"/>
      </action>
      <action id="grammar.Run.JFlex" class="org.intellij.grammar.actions.BnfRunJFlexAction"
              use-shortcut-of="grammar.Generate">
      </action>
      <action id="grammar.Generate.ParserUtil" class="org.intellij.grammar.actions.BnfGenerateParserUtilAction"/>
      <action id="grammar.Generate.JFlexLexer" class="org.intellij.grammar.actions.BnfGenerateLexerAction"/>
      <add-to-group group-id="grammar.file.group" anchor="first"/>
    </group>
  </actions>
</idea-plugin>
```

Рисунок 2.1 — Приклад манифесту плагіна

У загальному вигляді, архітектура IntelliJ Platform Plugin SDK відображає філософію JetBrains щодо забезпечення потужних, гнучких та користувацьких інструментів для розробки.

2.2 Можливості для розширення IDE

IntelliJ Platform Plugin SDK надає комплексне середовище для розробки плагінів, що націлене на максимальне зручність і продуктивність розробників. Це середовище базується на декількох важливих принципах, серед яких вирішення типових задач, гнучкість налаштувань, інтеграція з іншими інструментами та висока продуктивність.

Однією з ключових особливостей IntelliJ Platform Plugin SDK є використання системи PSI (Program Structure Interface). PSI — це абстрактний інтерфейс, який надає доступ до структури програмного коду. Його використовують для аналізу коду, рефакторингу, навігації, виконання коду і багатьох інших задач, пов'язаних з обробкою коду.

PSI дозволяє аналізувати та модифікувати код на різних рівнях, від індивідуальних лексем до високорівневих структур, таких як класи, методи і так далі. Це полегшує створення розширень, які впливають на поведінку IDE при роботі з кодом, дозволяючи розробникам керувати такими аспектами, як форматування коду, автоматичне доповнення, підсвічування синтаксису, і т.д.

Більше того, PSI працює з різними мовами програмування, що робить IntelliJ Platform Plugin SDK універсальним інструментом для розробки плагінів. Це означає, що розробники можуть використовувати цей SDK для створення плагінів, які працюють з різними мовами програмування, не потребуючи писати окремий код для кожної мови. У PSI також входять структурні елементи UI, приклад яких можна подивитись на рис. 2.2

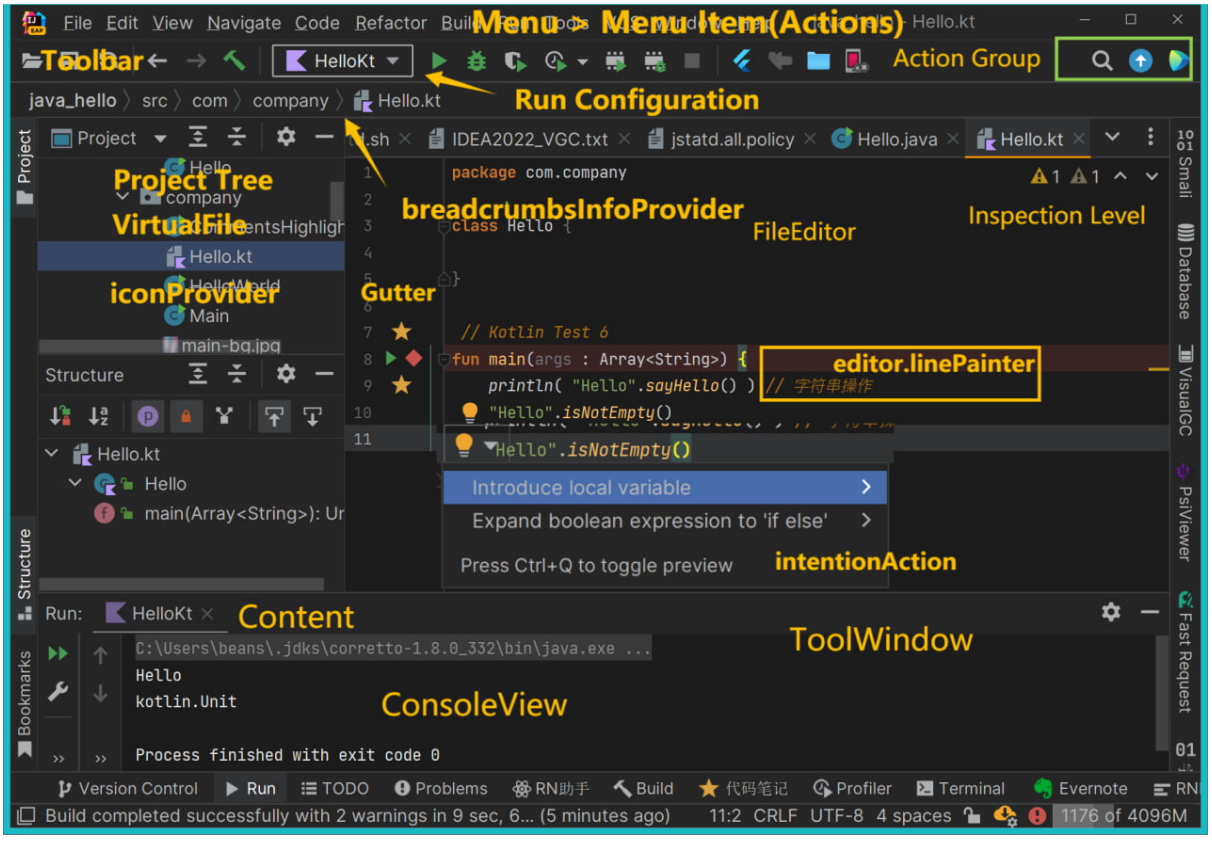

Рисунок 2.2 — Структурні елементи PSI

Таким чином, IntelliJ Platform Plugin SDK, використовуючи систему PSI, дає розробникам гнучкий і потужний інструмент для створення розширень, які можуть значно покращити продуктивність розробників і користувацький досвід при роботі з IDE.

2.3 Використання SDK для створення мовних розширень

Grammar-Kit — це плагін для IDE від JetBrains, призначений для підтримки створення власних мов програмування. Він включає інструменти для генерації лексерів і парсерів на основі визначень граматики мови в форматах .lex і .bnf.

У контексті розробки плагіну для PyCharm, Grammar-Kit відіграє важливу роль, оскільки дозволяє визначити граматику QML та генерувати відповідний лексер і парсер. Лексер, або лексичний аналізатор, є першим кроком у процесі аналізу коду: він розбиває вхідний текст на послідовність "лексем", або токенів, кожен з яких представляє окрему синтаксичну одиницю (наприклад, ідентифікатор, ключове слово, оператор тощо).

Парсер, або синтаксичний аналізатор, наступний за лексером. Він бере послідовність токенів, створених лексером, та використовує граматику мови, щоб побудувати "дерево розбору" — структуроване представлення коду, яке відображає його синтаксичну структуру. Дерево розбору можна потім використовувати для різних цілей, включаючи аналіз коду, рефакторинг, підсвічування синтаксису, автодоповнення та інше.

Grammar-Kit — це потужний інструмент, але він не є єдиним можливим рішенням для створення лексерів і парсерів. Інші інструменти, такі як ANTLR або Bison, також можуть використовуватися для цієї мети. Проте, є декілька причин, через які Grammar-Kit є привабливим вибором

для розробки плагіну для PyCharm. По-перше, Grammar-Kit було спеціально розроблено для інтеграції з IDE від JetBrains, включаючи PyCharm. Це означає, що він має глибоку інтеграцію з редактором коду і іншими компонентами IDE, що дозволяє створювати більш розширені і зручні для користувача розширення. По-друге, Grammar-Kit підтримує формати .lex і .bnf, що є широко використовуваними стандартами для опису граматики мов програмування. Це означає, що він може легко працювати з вже існуючими визначеннями граматики, і що розробники, які знайомі з цими форматами, зможуть легко працювати з Grammar-Kit. По-третє, Grammar-Kit включає ряд вбудованих інструментів, що спрощують процес розробки. Наприклад, він включає в себе генератор коду, що автоматично створює Java-класи для роботи з лексемами і деревами розбору, а також інструменти для перевірки граматики і візуалізації дерев розбору.

Таким чином, хоча існують інші інструменти, які можна використовувати для створення лексерів і парсерів, Grammar-Kit пропонує унікальну комбінацію потужності, гнучкості і інтеграції з IDE від JetBrains, що робить його ідеальним вибором для розробки плагіну для PyCharm.

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РОЗРОБКА РОЗШИРЕННЯ

3.1 Визначення функціональних вимог до розширення

З практичної точки зору розроблене розширення має задовольнити базові вимоги користувача, який повинен мати змогу зробити за допомогою розширення те чого він очікує. Для розширення PyCharm IDE з підтримкою QML передбачені наступні функціональні вимоги:

а) підсвічування синтаксису, розширення повинно надавати синтаксичне підсвічування для коду QML, що полегшує читання та розуміння коду;

б) автодоповнення, плагін повинен підтримувати автодоповнення коду для QML, допомагаючи користувачам швидше та точніше писати код;

в) пошук і навігація, повинна бути можливість пошуку та навігації по коду QML, що полегшить розробку великих проектів;

г) діагностика і виправлення помилок, повинно бути можливість мати засоби для виявлення та виправлення синтаксичних та семантичних помилок в коді QML;

ґ) підтримка рефакторингу, необхідна підтримка рефакторинг коду QML, допомагаючи користувачам підтримувати чистоту та якість свого коду;

д) підтримка різних версій QML, розширення повинно підтримувати різні версії QML, дозволяючи користувачам використовувати найновіші функції мови.

Ці вимоги допомагають забезпечити, що розширення буде корисним та ефективним інструментом для розробників, які працюють з QML в PyCharm.

3.2 Проектування архітектури розширення

Проектування архітектури розширення — це критичний етап у розробці. Перед впровадженням будь-якого коду, необхідно ретельно продумати структуру та процеси, які будуть відбуватися в розширенні. Саме на цьому етапі виписуються основні модулі, визначаються їх взаємодії та залежності.

Загалом, архітектура розширення залежить від специфічних потреб та мети розширення. У нашому випадку, основою розширення є побудова AST (Abstract Syntax Tree) з тексту QML коду, розфарбування синтаксису, автозавершення коду та перевірка синтаксичних та семантичних помилок. На цьому етапі було прийнято рішення використовувати Grammar-Kit для побудови лексера та парсера, а також Platform SDK IntelliJ IDEA для взаємодії з IDE.

Для кожного з модулів було розроблено відповідні вимоги, визначено його основні відповідальності та можливі взаємодії з іншими модулями:

— модуль src/main/java/com/qmlplugin/language буде містити усі необхідні класи та файли що використовуються для автоматичної генерації коду (а саме Qml.bnf, Qml.flex для лексера та парсера відповідно);

— модуль src/main/java/com/qmlplugin/language/psi буде визначати додаткову імплементацію класів, що використовуватимуться для автоматично генерируемого парсеру;

— модуль src/main/resources визначатиме різного роду конфігурації (файли з розширенням .xml для вказання кольорових схем за замовчуванням а також необхідні параметри при зборці) та статичні файли (такі як картинки що використовуються для задання розширення);

— модуль src/test буде повторять структуру src/main, але визначаючи як саме повинні відбуватися тести різних частин коду.

Проектування архітектури розширення — це не статичний процес, він продовжуватиметься протягом всього циклу розробки, оскільки кожен новий етап може внести зміни у вимоги та потреби розширення. Цей етап допомагає забезпечити стабільність, масштабованість та легкість удосконалення розширення в майбутньому, а також дозволяє зосередитись на якості коду та ефективності виконання. Остаточно побудовану архітектуру можна побачити на рис. 3.1.

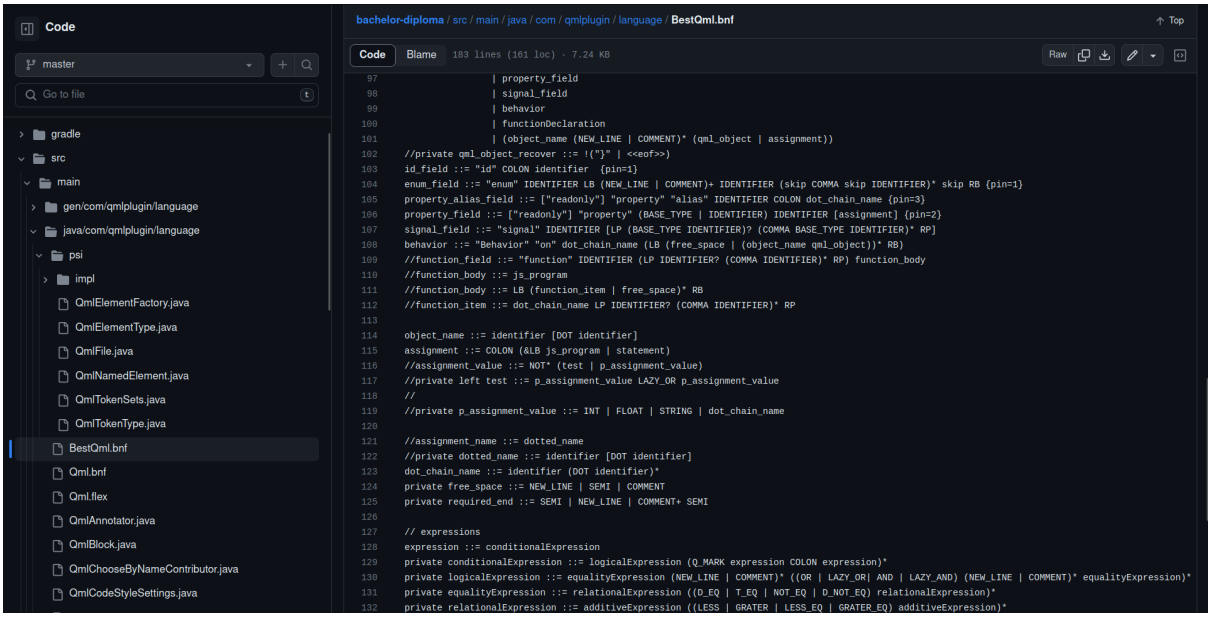

Рисунок 3.1 — Остаточна структура проекту та вигляд граматики BNF [8]

3.3 Розробка розширення, використовуючи SDK

Основною метою цього етапу проекту було використання SDK для розробки розширення. Почав відповідно з інсталяції SDK IntelliJ IDEA та

встановлення необхідного середовища розробки. Використовуючи IntelliJ IDEA як основне середовище розробки, я мав змогу легко перейти від концепції до коду. Завдяки чудовим інструментам, таким як вбудований редактор коду, менеджер проектів і налагоджувач, я міг ефективно організувати та управляти своїм проектом.

Після встановлення необхідного середовища розробки, було начато процес розробки розширення. Процес включав створення нового проекту розширення IntelliJ, перегляд вимог до функціональності розширення, імплиментація архітектури програмного забезпечення, написання коду, тестування та виправлення помилок. Щодо реалізації функціональності, було важливо забезпечити правильну підтримку синтаксису QML, включаючи розпізнавання ключових слів, розпізнавання типів даних і контроль синтаксису. Якщо користувач ввів некоректний синтаксис, розширення повинно було показати відповідне повідомлення про помилку. Крім того, розширення повинно було надавати корисні поради та рекомендації для поліпшення коду. В результаті було отримано наступний конфігураційний файл з вказанням що саме змінюється в IDE при імпорті розширення (Рисунок 3.2).

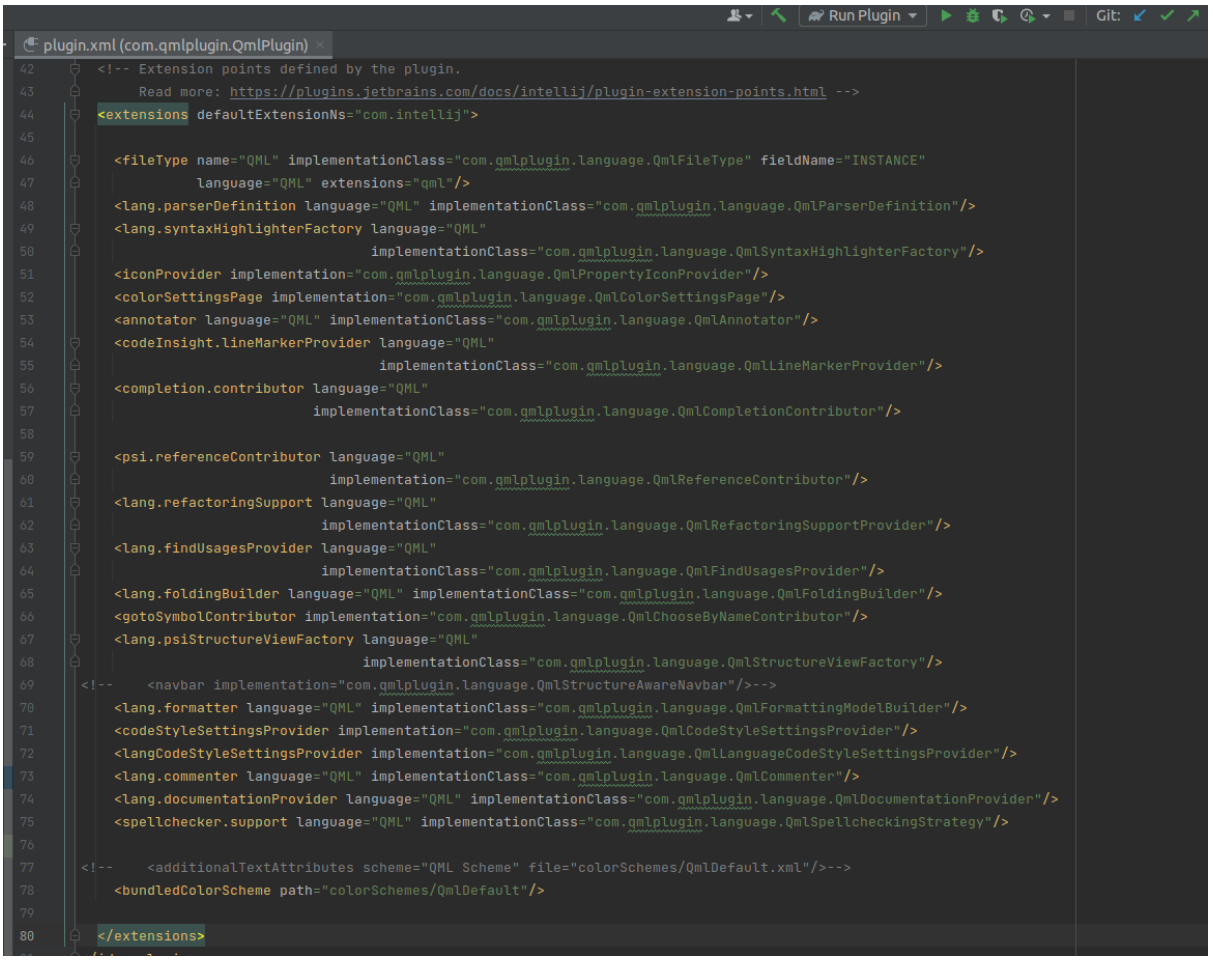

Рисунок 3.2 — Основний конфігураційний файл розширення

Процес розробки також включав створення юніт-тестів для перевірки коректності роботи розширення, детальніше в підрозділі 3.4. Це було важливою частиною процесу, оскільки це допомогло виявити та виправити помилки на ранніх стадіях розробки.

Крім того, зі складнощів що виникали, була необхідність позбутися неоднозначності парсингу, а також вирішення проблем з рекурсивним визначенням граматики мови.

Неоднозначність у парсингу виникає, коли є більше ніж один можливий спосіб розбити вхідний текст на лексеми або інтерпретувати ці лексеми відповідно до граматики мови. Це може призвести до створення двох або

більше синтаксичних дерев для одного й того ж вхідного тексту. Наприклад, вираз "a / b / c" може бути інтерпретований як "(a / b) / c" або "a / (b / c)", що приводить до різних результатів [9].

Неоднозначність може бути вирішена різними способами. Одним з них є переписування граматики таким чином, щоб змінити пріоритет операторів, що вирішує проблему вищезгаданого прикладу. Також використовують введення нових нетермінальних символів або перестановка правил [10]. Інший підхід полягає в використанні технології аналізу, що може обробляти неоднозначність. Наприклад, GLR-парсери (Generalized Left-to-right Rightmost derivation) можуть обробляти граматики з неоднозначностями, створюючи декілька синтаксичних дерев за потреби. Проте, найкращий спосіб уникнути неоднозначності полягає у відповідному проектуванні граматики з самого початку. Це дозволить уникнути проблем, пов'язаних з неоднозначністю, і зробить процес парсингу більш простим і ефективним.

Ліва рекурсія в граматиці виникає, коли нетермінальний символ в першому правилі виводу вказує на себе самого. Наприклад, якщо у нас є правило виводу A ::= Aα | β, де A є нетермінальним символом, α та β є строками термінальних і/або нетермінальних символів, то це правило вважається ліворекурсивним. Це може створити проблеми для деяких алгоритмів синтаксичного аналізу, наприклад, для рекурсивного спуску, оскільки це може призвести до безкінечного циклу.

Розглянемо для прикладу частину, що відповідає за логічні та математичні вирази (Рисунок 3.3).

```
// expressions
expression ::= conditionalExpression
private conditionalExpression ::= logicalExpression (Q_MARK expression COLON expression)*
private logicalExpression ::= equalityExpression (NEW_LINE | COMMENT)* ((OR | LAZY_OR| AND | LAZY_AND) (NEW_LINE | COMMENT)* equalityExpression)*
private equalityExpression ::= relationalExpression ((D_EQ | T_EQ | NOT_EQ | D_NOT_EQ) relationalExpression)*
private relationalExpression ::= additiveExpression ((LESS | GRATER | LESS_EQ | GRATER_EQ) additiveExpression)*
private additiveExpression ::= multiplicativeExpression ((PLUS | MINUS) multiplicativeExpression)*
private multiplicativeExpression ::= unaryExpression  ((MULT | DIV | PERCENT) unaryExpression)*
private unaryExpression ::= (NEW_LINE | COMMENT | NOT | MINUS)* primaryExpression
private primaryExpression ::= "null" | literal | dot_chain_name | LP expression RP
private literal ::= STRING | FLOAT | INT
```

Рисунок 3.3 — Граматика логічних та математичних виразів

На прикладі вище добре продемонстрированный метод нисходящего парсингу. Низхідний парсинг — це метод синтаксичного аналізу, при якому дерево виводу генерується з кореня до листя. Тобто, аналіз починається з початкового символу граматики (кореня) і продовжується до термінальних символів (листя). Низхідний парсинг може бути реалізований за допомогою методу рекурсивного спуску, де для кожного нетермінального символу в граматиці пишеться підпрограма або функція, яка намагається розпізнати ліві частини продукцій цього символу.

Однак, як я вже згадав, цей метод не працює з граматиками, що містять ліву рекурсію, тому усунення лівої рекурсії — важливий етап підготовки граматики для низхідного парсингу.

Щоб позбутись лівої рекурсії, можна перетворити ліворекурентну граматику на граматику, що не містить лівої рекурсії. Для цього замість безпосереднього вказівки на себе у першому правилі виводу, використовується індиректний метод через додатковий нетермінал. Наприклад, ліворекурсивне правило A ::= Aα | β можна перетворити на два нові правила: A ::= βA' та A' ::= αA' | ε, де ε вказує на пусту строку.

Для семантичного аналізу використовується механізм анотацій (Рисунок 3.4), тобто створюється додатковий прошарок з вказанням що щось має неправильний вигляд, або ж може бути оптимізовано.

```
@Override
public void annotate(@NotNull final PsiElement element, @NotNull AnnotationHolder holder) {
    if (element.getText().equals("import") || element.getText().equals("as")) {
        var range = element.getTextRange();
        if (element.getParent() instanceof QmlImportItem) {
            holder.newSilentAnnotation(HighlightSeverity.INFORMATION)
                    .range(range).textAttributes(QmlSyntaxHighlighter.IMPORT).create();
        } else {
            holder.newAnnotation(HighlightSeverity.ERROR, message: "Unexpected import")
                    .range(range)
                    .textAttributes(QmlSyntaxHighlighter.BAD_CHARACTER)
                    .highlightType(ProblemHighlightType.GENERIC_ERROR_OR_WARNING)
                    .create();
        }
        return;
    }
}
```

Рисунок 3.4 — Анотація для відображення помилки імпортів

3.4 Тестування розширення і виправлення помилок

Тестування розширення і виправлення помилок — це важливий крок у розробці плагінів IntelliJ Platform. Розробка без відповідного тестування може призвести до створення продукту, який не працює так, як очікувалось, або, що гірше, може заважати коректній роботі інших компонентів IDE. Тому детальне тестування та виправлення помилок є невід'ємною частиною процесу розробки. При тестуванні варто в повній мірі перевірити код на наявність як синтаксичних, так і семантичних помилок. Це включає в себе перевірку коду на відповідність QML синтаксису, а також перевірку логіки роботи розширення.

Не менш важливим етапом є перевірка на різних версіях PyCharm та інших продуктів на базі IntelliJ Platform. Це допоможе забезпечити сумісність розширення з різними продуктами та версіями. Для забезпечення якості коду та пошуку помилок використовувати техніка юніт-тестування, яка допомагає переконатися, що окремі частини

розширення працюють правильно. В даному випадку перевірка що дані розбиваються на правильні

Після виявлення помилок вони повинні бути виправлені, а потім процес тестування слід повторити. Це важливо для підтвердження, що виправлення не спричинили нових проблем, що насправді зустрічається досить часто. Варто також зазначити, що робота над виправленням помилок ніколи не закінчується. Навіть після релізу розширення, важливо продовжувати моніторинг звітів про помилки та відгуків користувачів, щоб швидко реагувати на виявлені проблеми.

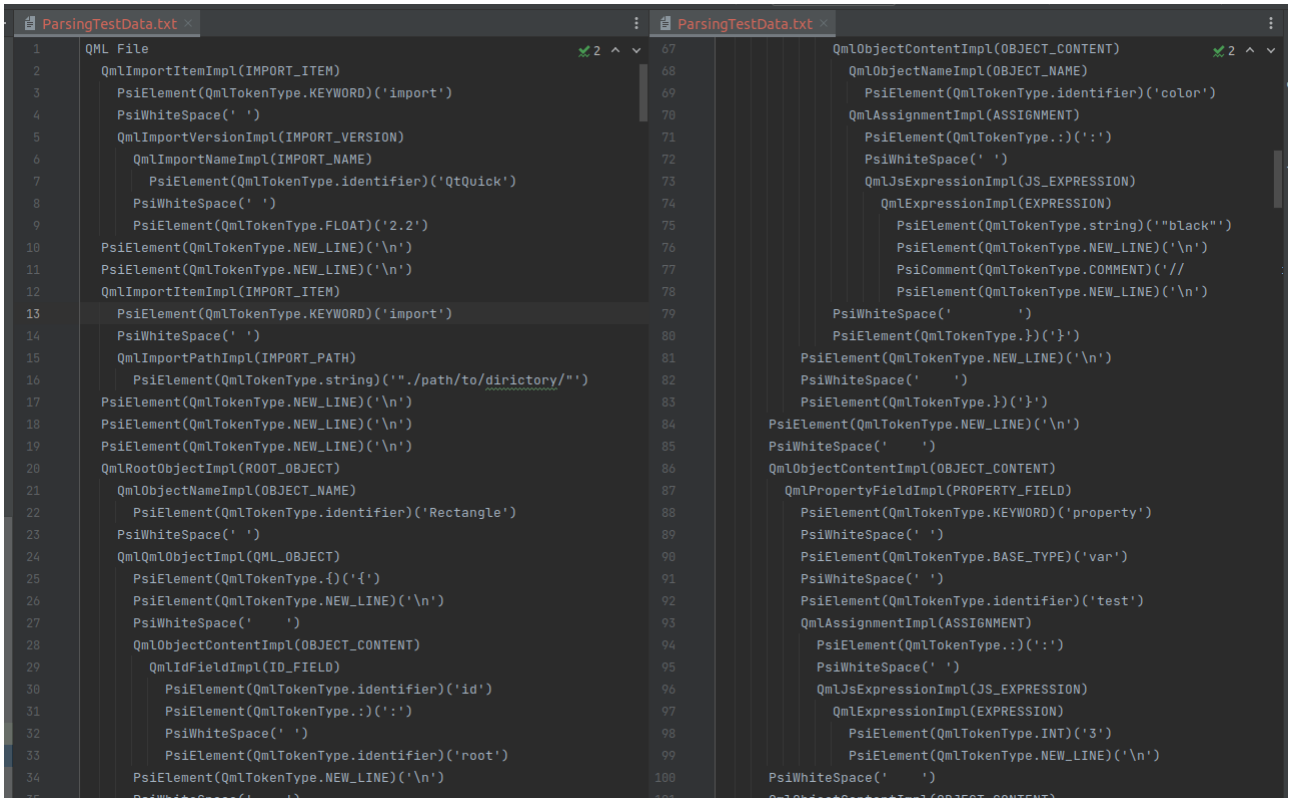

Рисунок 3.5 — Представлення структури AST для тестування

Тестування парсингу в плагіні являє собою перевірку роботи лексера та парсера. Ці дві компоненти відповідають за аналіз вхідного коду та створення AST (рис 3.5), яке потім використовується для семантичного аналізу, підсвічування синтаксису, форматування, а також інших функцій

IDE. Недоліки в парсингу можуть призвести до некоректного розуміння коду IDE та наслідкових помилок.

Аспекти, які було враховано при тестуванні:

— Перевірка лексера: лексер перетворює вхідний потік символів на послідовність токенів. Тести лексера зазвичай включають перевірку правильності визначення токенів та їх меж;

— Перевірка парсера: парсер використовує токени, згенеровані лексером, для створення AST. Тести парсера перевіряють, чи правильно парсер відтворює структуру та семантику вхідного коду;

— Перевірка обробки помилок: важливо перевірити, як парсер обробляє синтаксичні помилки. Це означає, що парсер повинен відмовлятися від некоректного вхідного коду, але водночас продовжувати розбір коду після місця помилки;

— Профілювання та оптимізація: на великих файлах або проектах процес парсингу може займати значний час, тому важливо перевірити продуктивність парсера та лексера.

Для тестування парсингу використовую юніт-тести, створюворюючи тестові випадки з різними вхідними даними та роблю перевірку, чи відповідає дерево розбору, створене лексером та парсером, очікуваному. Крім того, додані тестові випадки, які навпаки включають синтаксичні помилки, і перевіряють, чи правильно парсер обробляє ці помилки.

Можливі сценарії тестування включають, але не обмежуються:

— Простий код без синтаксичних помилок;

— Код що має синтаксичними помилками;

— Код, що містить коментарі, пробіли та інші допоміжні символи;

— Випадки одразу з багатьма різними конструкціями мови;

— Великі файли або проекти для тестування продуктивності.

За допомогою цих тестів можна забезпечити, що парсер працює правильно, забезпечуючи коректну підтримку мови в IDE.

РОЗДІЛ 4 ДЕМОНСТРАЦІЯ ТА ОЦІНКА РОЗШИРЕННЯ

4.1 Встановлення та демонстрація роботи розширення

Розширення, яке було розроблено в ході цього проекту, призначене для полегшення роботи з QML в середовищі Jetbrains. Щоб зрозуміти, як воно працює на практиці, необхідно встановити його (Додаток А) та зробити огляд можливостей. Наступні рисунки (рис. 4.1 — 4.8) демонструють основні елементи роботи з розширенням. Вони допоможуть зрозуміти, як плагін може полегшити розробку QML-проектів, надаючи синтаксичний аналіз, автодоповнення коду, визначення символів та інші корисні функції.

Перш за все це розуміння речей які одразу будуть помітні після встановлення. А саме: зміна іконки для .qml файлів та кольоровий вигляд вмісту файлу (Рисунок 4.1).

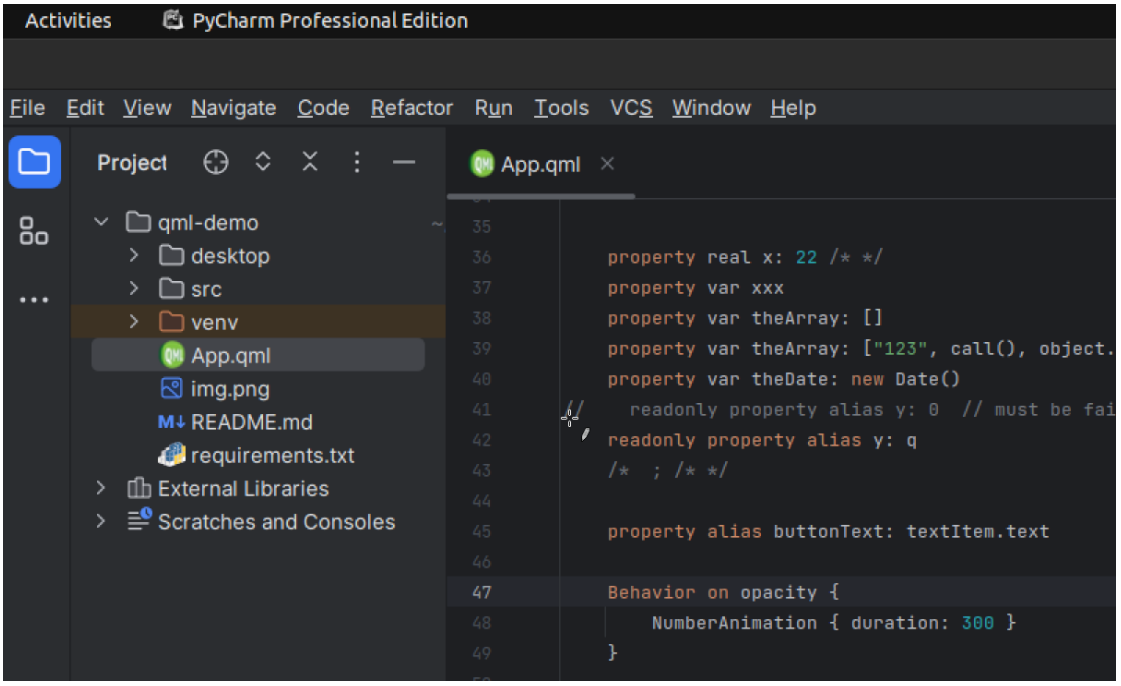

Рисунок 4.1 — Зміна іконки та виділення кольором тексту

Також було додані семантичні перевірки, наприклад при імпорт (Рисунок 4.2).

Рисунок 4.2 — Семантична перевірка імпорту

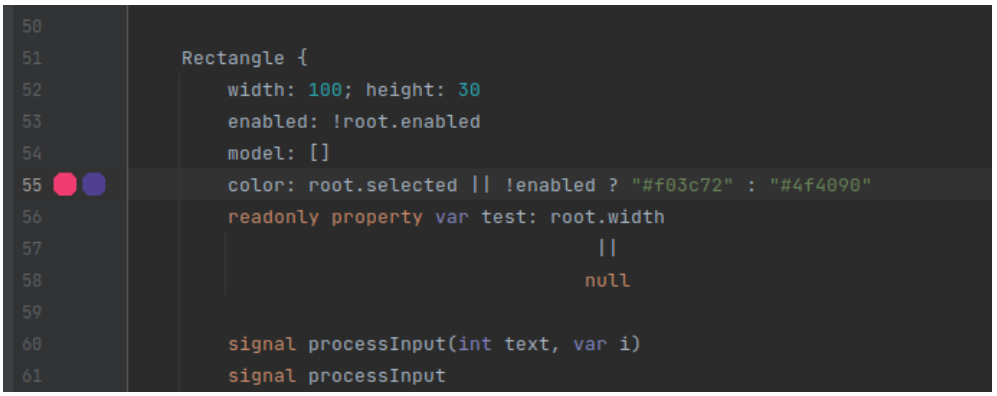

Рисунок 4.3 — Допоміжне вказання кольору з хекс значення

Звичайно що було додано автодоповнення коду, наприклад, при вказанні назви об'єкту що створюємо (Рисунок 4.4).

Рисунок 4.4 — Автодоповнення

Були інтегровані й інші чудові функції що пропонує JetBrains, такі як перевірка орфографічних помилок (Рисунок 4.5), коментування комбінацією

“Ctrl + /” (Рисунок 4.6) та вікна налаштувань внутрішніх параметрів (Рисунок 4.7 та Рисунок 4.8).

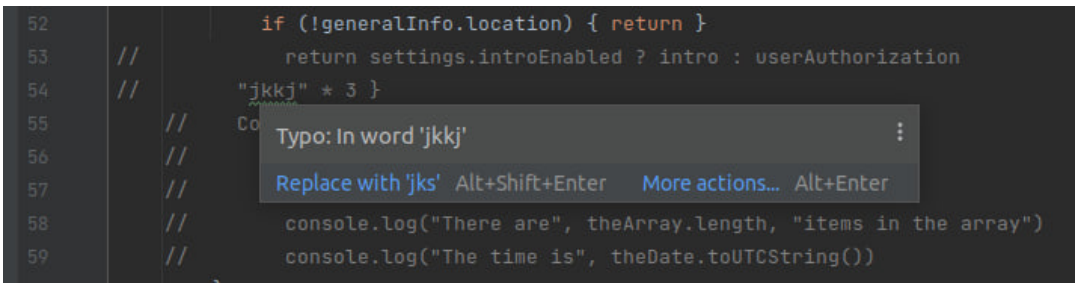

Рисунок 4.5 — Пошук та виправлення орфографічних помилок

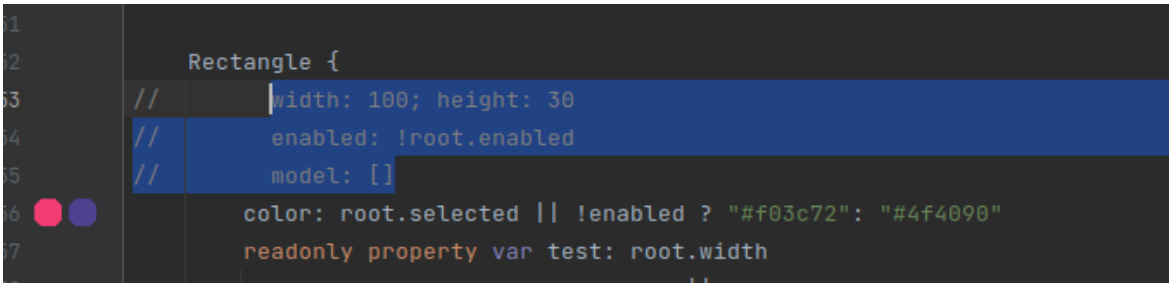

Рисунок 4.6 — Коментування рядків

Інші приклади можна подивитись в Додатку А.

Ефективність розширення можна визначити шляхом оцінки його впливу на продуктивність розробників. Розширення значно полегшує розробку на QML, автоматизуючи низку рутинних завдань та надаючи набір інструментів для аналізу коду. Це не тільки економить час, але й зменшує ризик помилок.

Щодо корисності, розширення вносить вагомий вклад в роботу з проектами, заснованими на QML, враховуючи усі його особливості та специфіку мови. Таким чином, розширення є ефективним та корисним інструментом для розробки на QML в середовищі Jetbrains.

ВИСНОВКИ

В ході виконання цієї роботи було створено власний засіб для підтримки QML в IDE, з підтримкою підсвітки кольором тексту, відображенням кольорів за шістнадцятковим значенням, автодоповнення в залежності від контексту, можливості коментування та форматування блоків коду, а також інший функціонал. Проведено дослідження можливостей створення розширення для PyCharm IDE на основі IntelliJ Platform Plugin SDK. Також проаналізовано ключові компоненти та архітектуру SDK та детально вивчено основні етапи створення плагіна, починаючи від формулювання функціональних вимог та завершуючи розробкою розширення.

Розробка досягла своєї основної мети — створити розширення, яке виконує зазначені завдання та функції. Кожен поставлений етап було успішно виконано, пройдено всі необхідні етапи від проектування до тестування та налагодження. Результати дослідження показали, що розроблене розширення є цілком практичним і може бути корисним для розробників програмного забезпечення. Воно вдосконалює процес написання коду, забезпечуючи більшу продуктивність і зручність, а також поліпшує розуміння коду завдяки візуалізації абстрактного синтаксичного дерева.

Щодо використання розширення в майбутньому, вважаю доцільно продовжити його поліпшення, включаючи додавання нових функцій та покращення існуючих. З виходом оновлень у IntelliJ Platform Plugin, також слід інтегрувати у власне рішення. Можна також розглянути можливість адаптації розширення для інших IDE, що розширить його застосування. Окрім того, важливо прислухатися до відгуків користувачів і включити їх в

подальші плани розвитку розширення. Потенційно у майбутнє, є безліч можливостей для розвитку та покращення розширення. При виході з метою зробити життя розробників простішим і продуктивнішим, можна додати більше інтуїтивно зрозумілих та корисних функцій, які спростять написання коду та його розуміння. Це може включати більш сучасні методи візуалізації коду, покращення UI/UX частин та навіть інтеграцію з іншими інструментами розробки.

Одним з основних планів на майбутнє є адаптація розширення до інших IntelliJ платформ. На мою думку, у даної розробки є потенціал стати універсальним інструментом для розробників що працюють з QML, і варто розширити його діапазон застосування, щоб досягти більшої аудиторії. На додаток до іншого, в планах провести більше досліджень у сфері оптимізації коду і вирішення типових проблем програмування синтаксичних аналізаторів, щоб зробити розширення ще більш корисним та ефективним. Наявний значний потенціал у використанні штучного інтелекту та машинного навчання для автоматизації багатьох рутинних задач, пов'язаних з написанням коду, або ж навіть його аналізу. Всі ці плани та ідеї мають на меті зробити розширення не просто інструментом, а незамінним помічником для кожного розробника QML. Завдяки чому зможу зробити значний внесок у світ програмування.

Підводячи підсумки роботи хочу сказати, що розроблене мною розширення є сумісним з найбільш поширеними операційними системами (Linux, Windows, MacOS) та популярными продуктами платформи JetBrains, а саме: PyCharm, IntelliJ IDEA та CLion. Це позитивно впливає на його потенційну аудиторію, адже розробникам, які використовують різноманітні інтегровані середовища розробки JetBrains, відкрито доступ до зручного інструменту для роботи з QML. Це розширення полегшить процес розробки, оптимізувати роботу з кодом та покращити якість кінцевого продукту. Маючи

можливість працювати в різних середовищах, розробники отримують гнучкість та зручність використання, що спонукає їх активно застосовувати це розширення у своїй щоденній роботі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Most Popular Programming Languages for 2022 (2023) [Електронний ресурс] – Режим доступу до ресурсу: https://balifuren-com.ngontinh24.com/article/most-popular-programming-languages-for-2022.

2. Jürgen B. Qt 6 QML [Електронний ресурс] / B. Jürgen, L. Cyril, T. Johan – Режим доступу до ресурсу: https://www.qt.io/hubfs/_website/QML%20Book/qt6book-with-frontpage.pdf

3. The QML Reference [Електронний ресурс] – Режим доступу до ресурсу: https://doc.qt.io/qt-5/qmlreference.html.

4. QML syntax highlighling & snippets for VSCode [Електронний ресурс] – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=cutetee.qml.

5. QmlEditor for PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://plugins.jetbrains.com/plugin/14434-qmleditor

6. About IntelliJ Platform SDK [Електронний ресурс] – Режим доступу до ресурсу: https://plugins.jetbrains.com/docs/intellij/about.html

7. Grammar-Kit HOWTO [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/JetBrains/Grammar-Kit/blob/master/HOWTO.md

8. Context free languages, context free grammars, and BNF [Електронний ресурс] – Режим доступу до ресурсу: https://zoo.cs.yale.edu/classes/cs201/topics/topic-cfls.pdf

9. CSE 425S: Programming Systems and Languages Dr. Gruia-Catalin Roman / Washington University in St. Louis / Fall 2008 [Електронний ресурс] – Режим доступу до ресурсу: https://classes.engineering.wustl.edu/2008/fall/cse425s/resources/bnf/

10. Aho A. Compilers: Principles, Techniques, and Tools, 2nd ed. / Aho A., Lem M., Sethi R., Ullman J. — Addison Wesley, 2007.

ДОДАТОК А
Інструкція з встановлення та використання розширення

1. Необхідно завантажити розширення, наприклад, за посиланням https://github.com/sNoDliD/bachelor-diploma/blob/master/QmlPlugin-1.0.0.jar, воно представлено у .jar файлі, який можна легко імпортувати у IDE.
2. Відкрити IDE від JetBrains, знайти в налаштуваннях вкладку Plugins, або ж двічі натиснути на Shift щоб відкрилось меню пошуку, де можна буде зробити пошук, дивіться рис. А.1

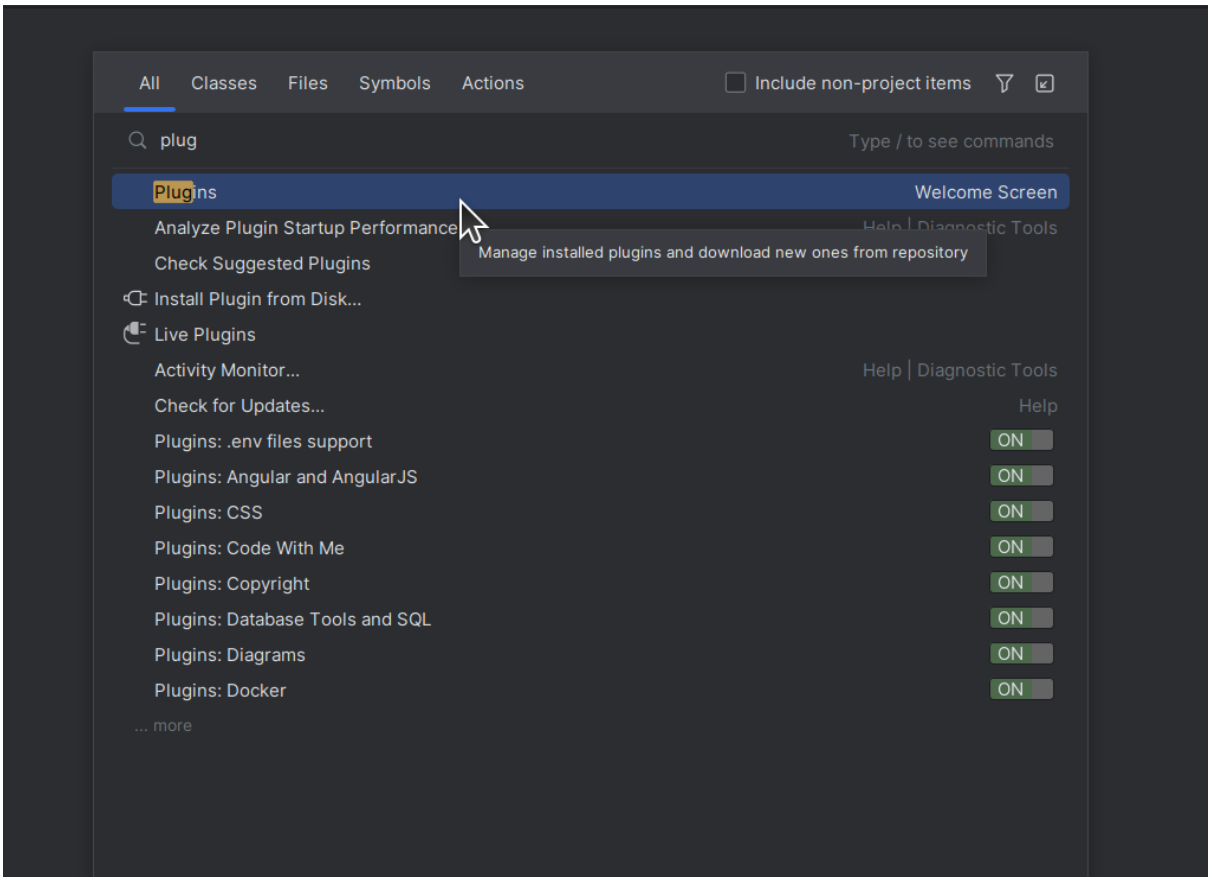

Рисунок А.1 — Пошук меню плагінів

3. Зайти в налаштування (іконка з шестернею) та обрати пункт “Встановити Плагін з Диску” (Рисунок А.2)

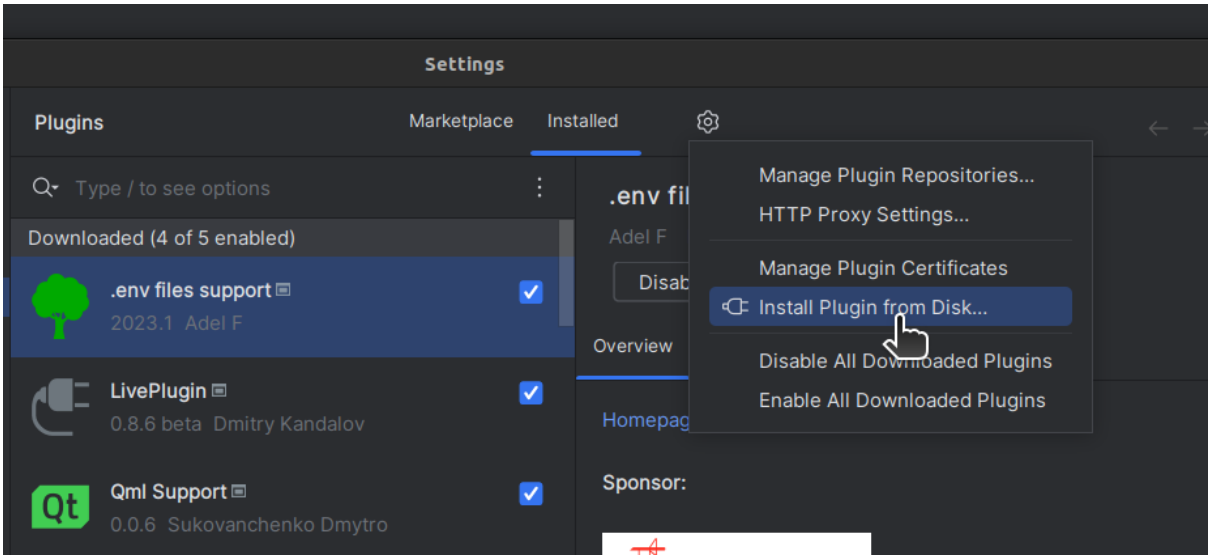

Рисунок А.2 — Відкриття окна імпорту розширення

4. Далі необхідно обрати шлях до завантаженого раніше файлу (Рисунок А.3)

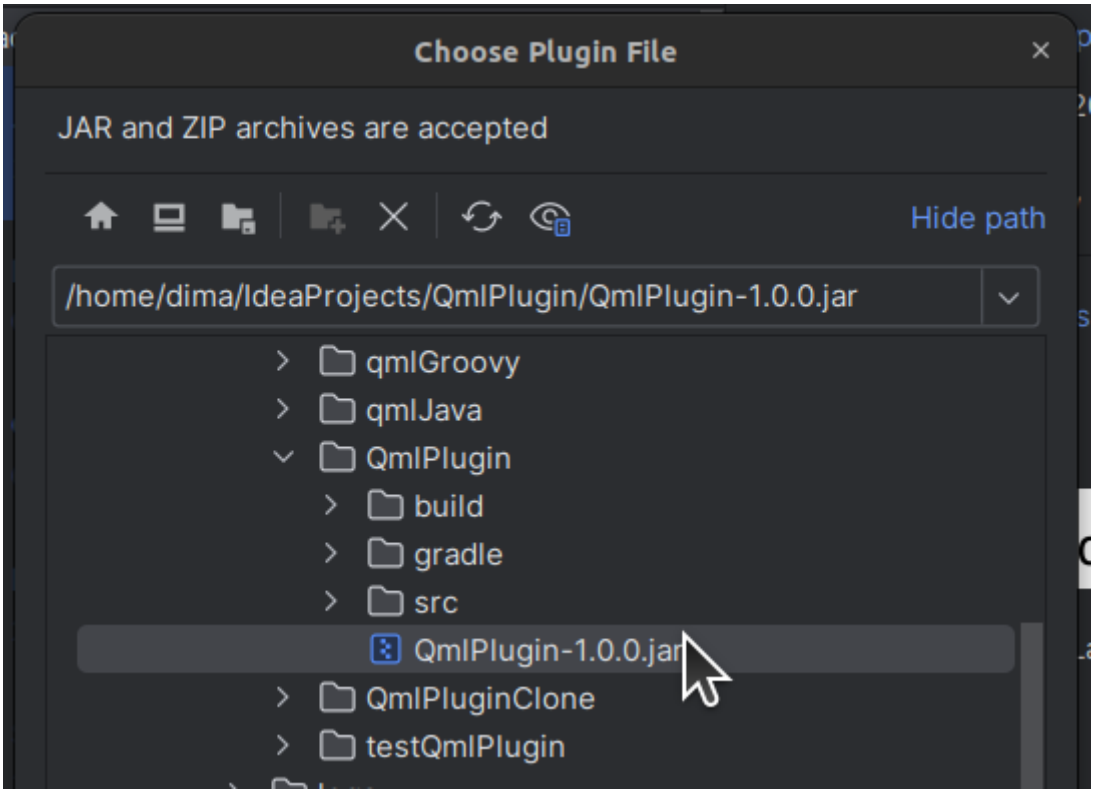

Рисунок А.3 — Обираємо файл з розширенням

5. Після чого починається процедура встановлення, IDE запропонує перезавантажити себе щоб успішно усе завершити (Рисунок А.4)

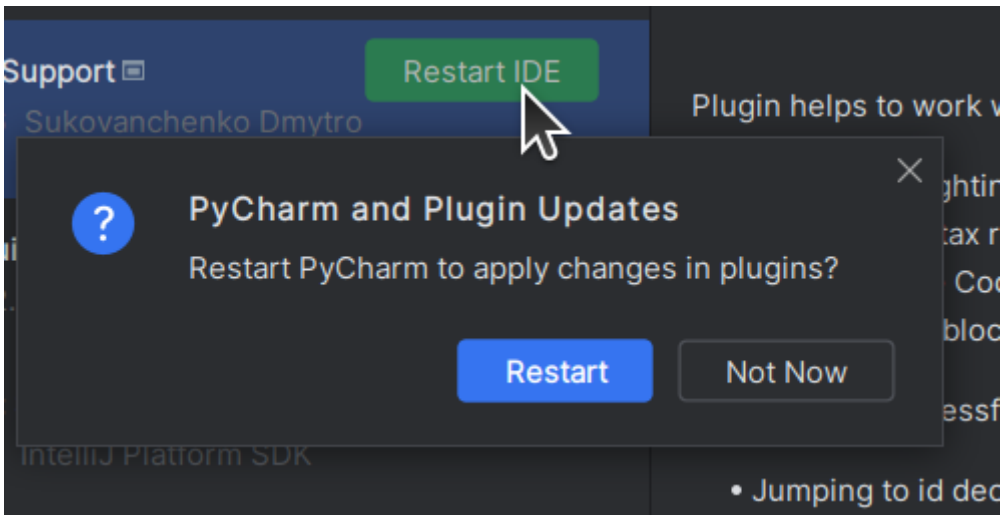

Рисунок А.4 — Пропозиція перезавантажити IDE

6. Коли застосунок перезавантажиться появиться інформаційне повідомлення, що буде свідчити про вдале встановлення (Рисунок А.5)

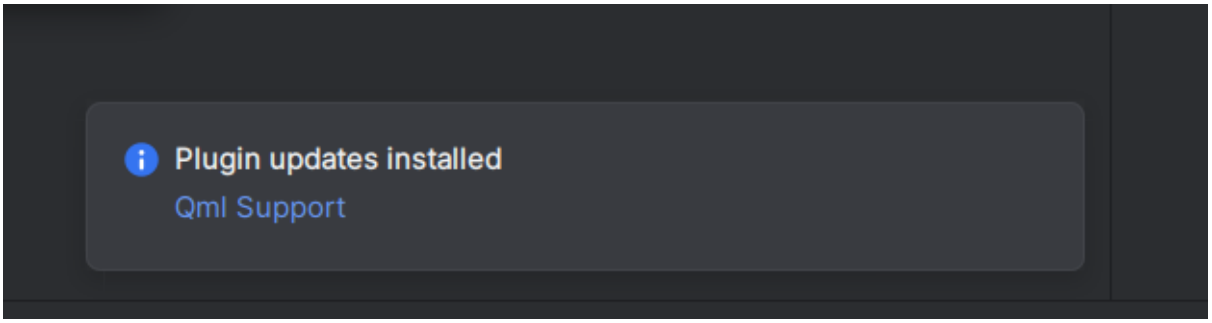

Рисунок А.5 — Повідомлення про успішне додавання розширення

7. Якщо в проекті вже були присутні файли з розширенням .qml, то можливо доведеться змінити їм тип за замовчуванням (Рисунок А.6). Для нових проектів буде підтримка сходу.

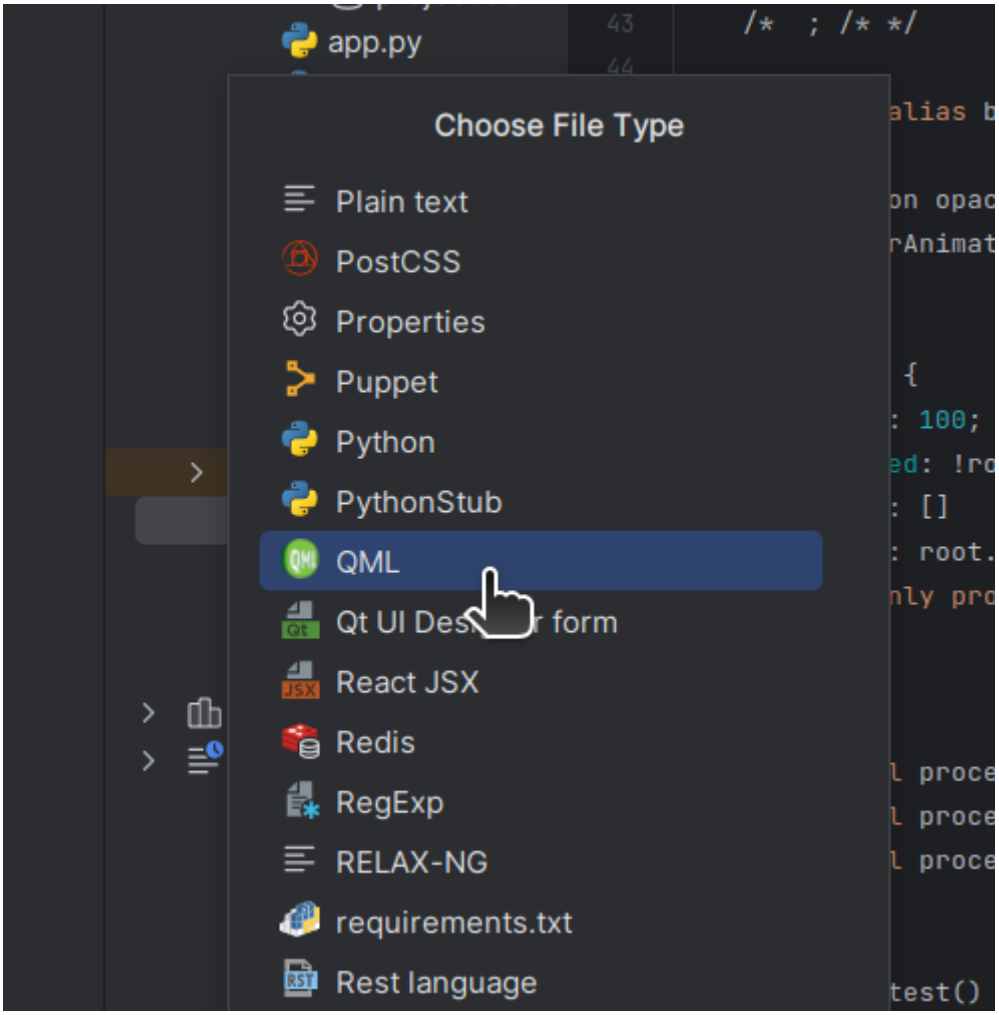

Рисунок А.6 — Зміна типу файла, що було присвоєно раніше

8. Готово, тепер IDE підтримує синтаксис мови QML, та інші додаткові функції для роботи з ним. Приклад коду з підсвіткою можна переглянути на рисунку А.7.

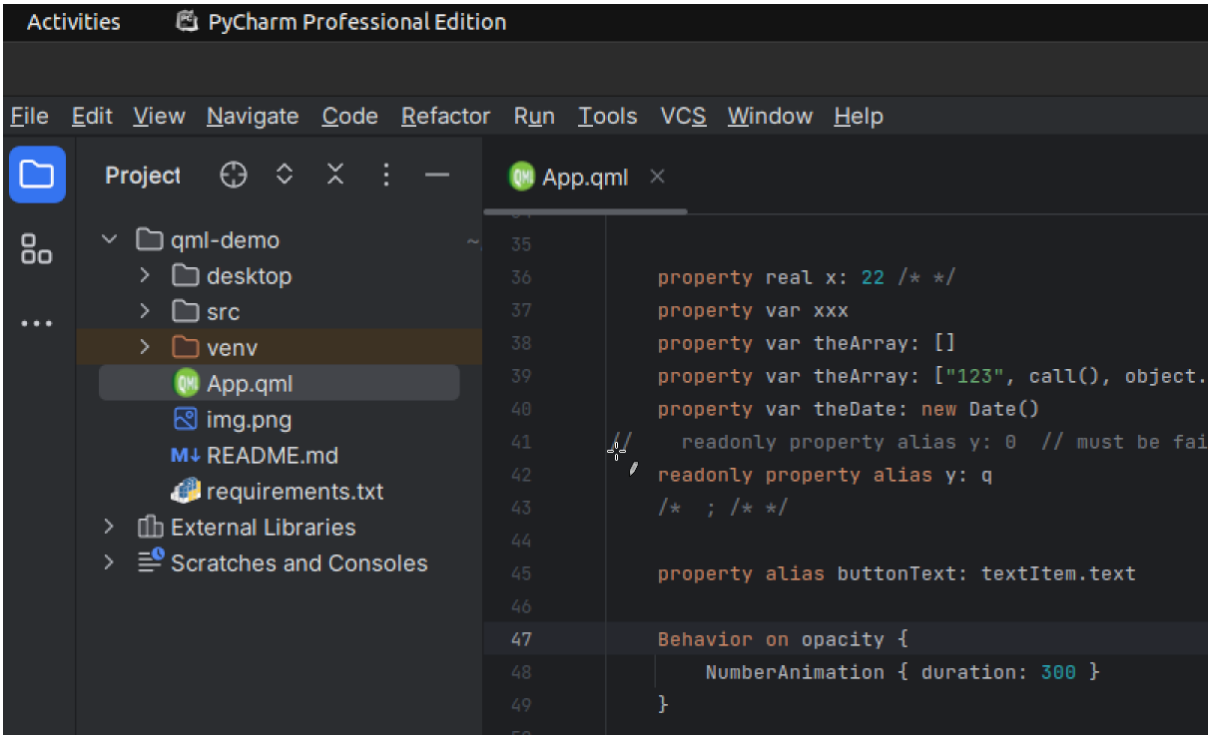

Рисунок А.7 — Вигляд підсвітки коду та зміненої іконки

9. Крім того, можна зробити налаштування розширення, а саме: можливе форматування (Рисунок А.8) та підсвітка синтаксису (Рисунок А.9)

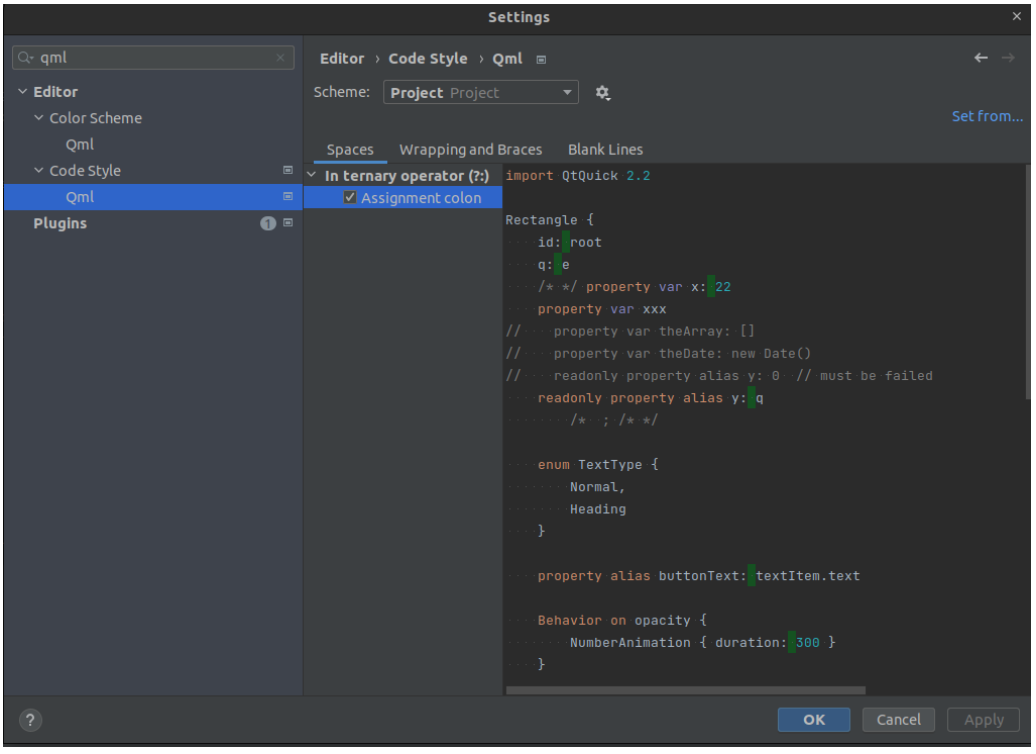

Рисунок А.8 — Налаштування форматування коду

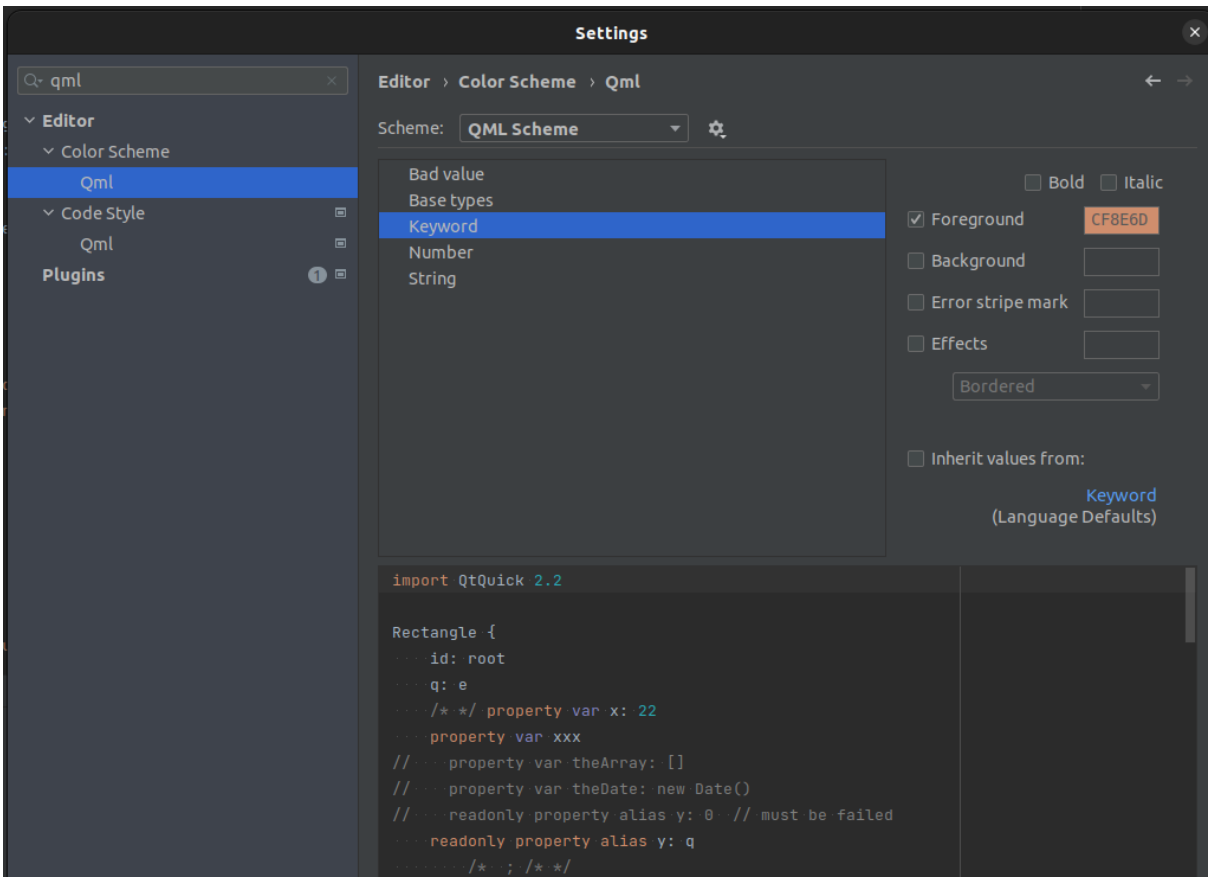

Рисунок А.9 — Налаштування кольорової схеми розширення

ДОДАТОК Б
Створення проекту з використанням IntelliJ Platform SDK

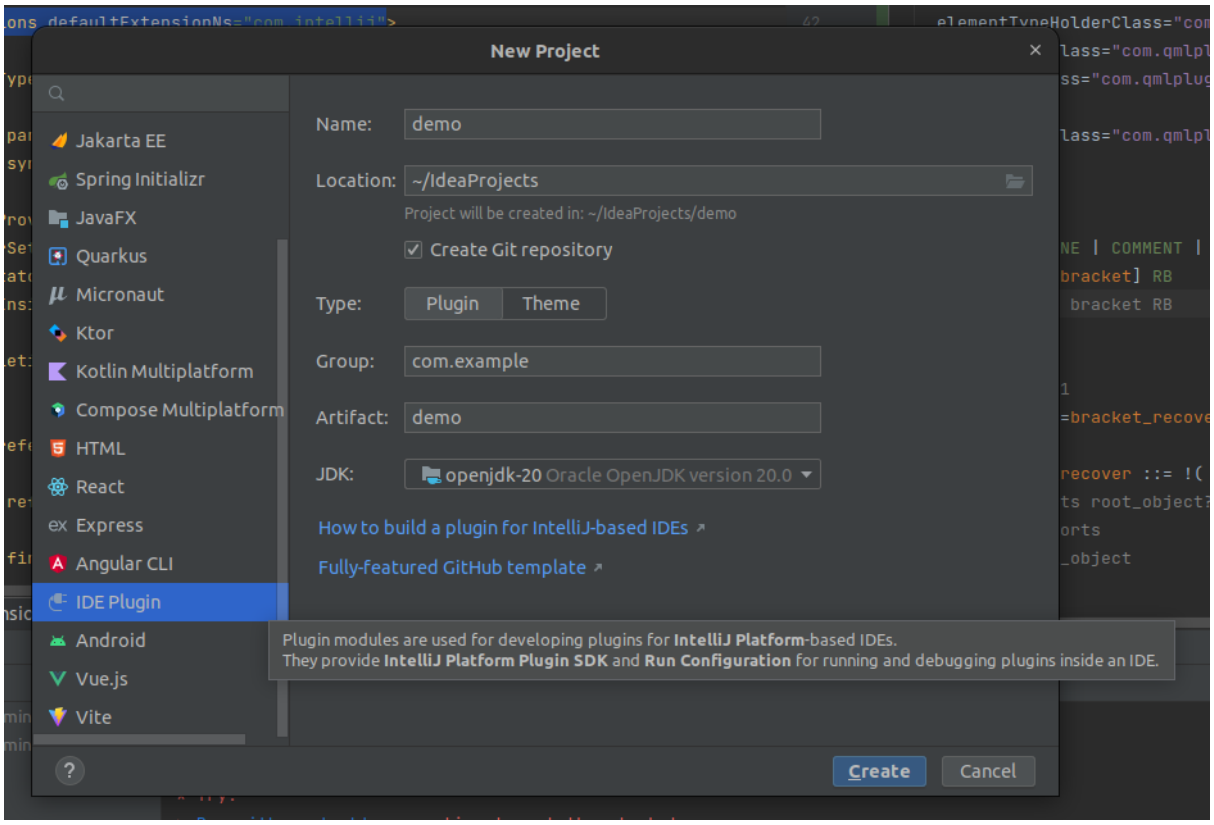

Рисунок Б.1 — Вибір варіанту створення плагіну

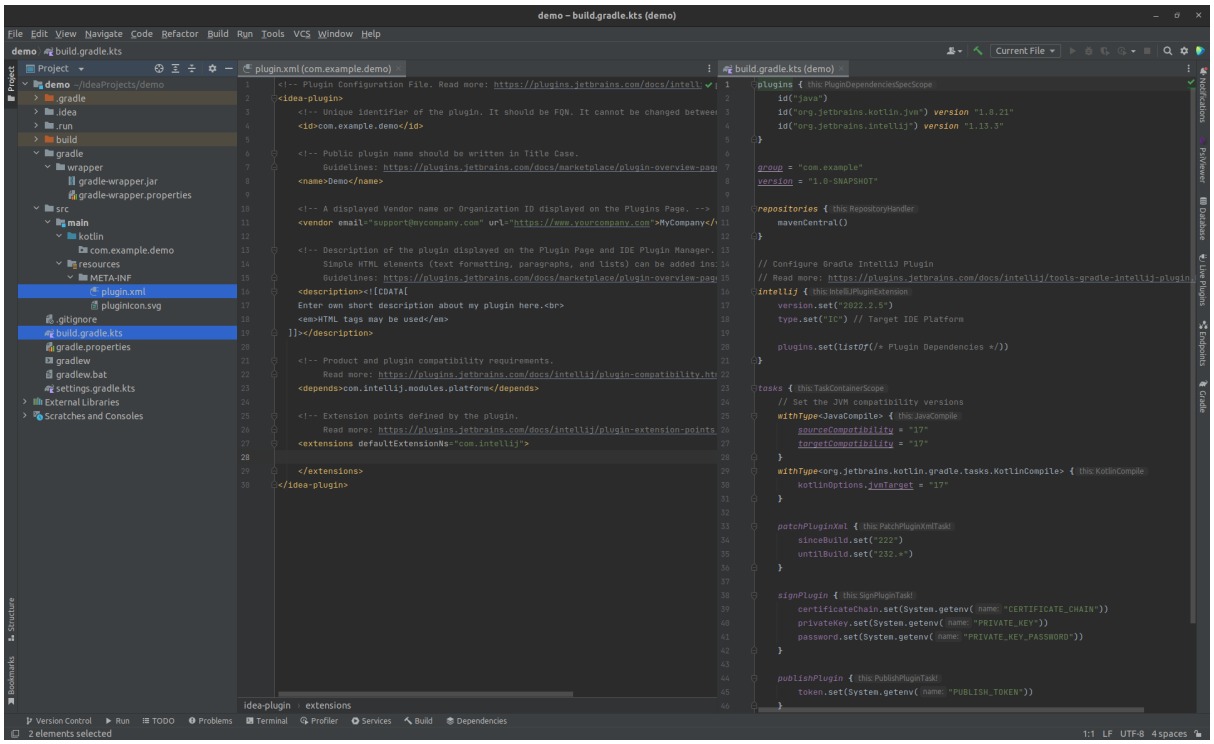

Рисунок Б.2 — Створений проект за необхідним шаблоном