Звіт до Лабораторної роботи №1

З дисципліни Кросплатформене програмування

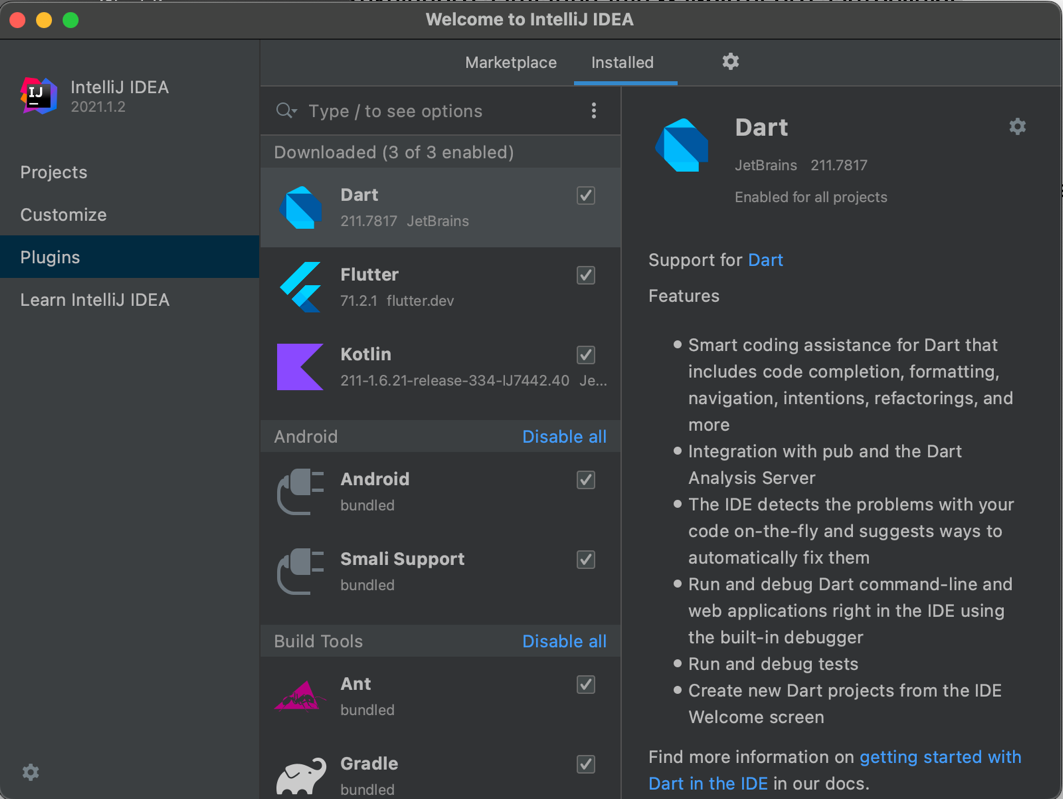
Виконав студент групи ПП-31  
Щербина Андрій Віталійович

Тема : Розгортання середовища для розробки крос-платформного застосунку, використовуючи Flutter та DART. Створення технічного завдання та архітектури.

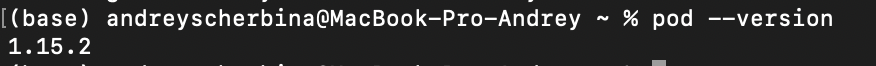
Мета : Метою лабораторної роботи є отримання практичних навичок з розгортання потрібних інструментів для майбутньої розробки та створення технічного завдання для майбутнього застосунку.

Хід роботи

На початку виконання лабораторної роботи необхідно визначитися з середою розробки та платформою на якій буде розроблятися майбутня програма. Я працюю у операційній системі MacOS та для виконання завдання використовував IntelliJ IDEA.  
Після встановлення програмного забезпечення потрібно увійти в IDE та натиснути на кнопку Plugins.



Шукаємо та завантажуємо необхідні плагіни, а саме – мову програмування Dart та фреймворк Flutter, також для розробки під MacOS та IOS нам знадобиться менеджер залежностей CocoaPods.



Звичайно потрібно оновити сторонні програми, такі як Xcode та Android Studio для встановлення додаткових залежностей.   
Не обов'язково встановлювати всі три IDE. Можна обрати одне середовище розробки відповідно до ваших потреб і налаштувати «PATH».

Також можна спробувати зібрати проект під веб, щоб переконатися в правильності налаштувань. Якщо все працює належним чином, код також має запускатися на інших платформах.  
Створення технічного завдання та архітектури:  
Основні аспекти ТЗ:  
1. Опис проекту:

* Ідея застосунку полягає в створенні інтерактивного інструмента для навчання і засвоєння мовних навичок за допомогою перекладача Google Translate.
* Мета застосунку - надати користувачам можливість вчити та практикувати іноземні мови шляхом взаємодії з текстом, аудіо та візуальними матеріалами.

Основні завдання:

* Надання доступу до перекладу тексту з різних мов на основі API Google Translate.
* Забезпечення можливості зберігання улюблених перекладів користувача.
* Створення зручного інтерфейсу для взаємодії з застосунком та використання його функцій для навчання.

2. Цільова аудиторія:

* Кінцеві користувачі: люди, які бажають вивчити нову мову або покращити свої мовні навички.
* Потреби та очікування: простий інтерфейс, широкий вибір мов для перекладу, можливість зберігати улюблені переклади, доступ до аудіо-та візуальних матеріалів для покращення вивчення мови.

3. Функціональні вимоги:

* Переклад тексту з однієї мови на іншу за допомогою Google Translate API.
* Збереження улюблених перекладів користувача.

4. Нефункціональні вимоги:

* Продуктивність: швидкий доступ до перекладів та інших функцій застосунку.
* Безпека: захист особистих даних користувачів та безпечність зв'язку з Google Translate API.
* Масштабованість: можливість розширення функціоналу та підтримки нових мов або функцій у майбутньому.
* Інтероперабельність: сумісність з різними операційними системами та пристроями.

5. Дизайн інтерфейсу:

* Простий та зрозумілий інтерфейс.
* Зручний доступ до основних функцій.
* Використання візуальних елементів для полегшення вивчання мови та перекладу.

6. Платформи та пристрої:

* Адаптація під операційні системи Android та IOS, MacOs та Web.
* Підтримка смартфонів та планшетів.

7. Інтеграції:

* Інтеграція з Google Translate API для перекладу тексту.

8. Тестування:

* Юніт-тести для перевірки роботи окремих компонентів.
* Інтеграційне тестування для перевірки взаємодії між компонентами.
* Тестування продуктивності та безпеки.

9. Вимоги до розгортання:

* Доступність для завантаження з App Store та Google Play Store.
* Встановлення та оновлення застосунку через магазини додатків.

10. Бюджет та терміни:

* Орієнтований бюджет: відповідно до розміру команди розробників та обсягу робіт.
* Очікувані терміни: залежить від складності функціоналу та обсягу робіт, але зазвичай від кількох місяців до півроку для повноцінного релізу.

Основні аспекти архітектури:

1. Використання шаблону Repository:

Для ефективного відокремлення джерела даних від бізнес-логіки додатку, ми використовуємо патерн Repository. Це сприяє полегшенню тестування та розширенню функціональності.

2. Використання Dependency Injection:

Для зручності управління залежностями та забезпечення їх доступності в класах Cubit, ми використовуємо Dependency Injection. Цей підхід робить код більш масштабованим та легким для тестування.

3. Використання BLoC та Cubit:

Управління станом додатку здійснюється за допомогою BLoC та Cubit. Це дозволяє ефективно реалізувати реактивну логіку та зручно керувати станом додатку.

4. Модульність та розділення відповідальностей:

Код розділений на окремі модулі, що дозволяє зберігати логічно пов'язаний код окремо. Наприклад, логіка перекладача розділена від презентаційного шару.

5. Використання MultiProvider та MultiBlocProvider:

Для забезпечення залежностей різних частин додатку використовуються MultiProvider та MultiBlocProvider. Це дозволяє ефективно керувати залежностями та станом додатку.

6. Використання констант для конфіденційних даних:

Для забезпечення безпеки та зручності управління конфіденційними даними використовується константа для зберігання ключа доступу до API Google Translate.

7. Сувора типізованість:

Використання суворо типізованих залежностей та станів допомагає уникнути помилок та забезпечує більшу стабільність додатку.

Висновки: Після виконання лабораторної роботи №1, я отримав практичні навички з розгортання потрібних інструментів для майбутньої розробки та створив технічне завдання для майбутнього застосунку.