МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ МОБІЛЬНОГО ДОДАТКУ «ROZCLOUD»

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу
Спеціальності 122 Комп'ютерні
науки та інформаційні технології
Освітньо-професійної програми
«Комп'ютерні
науки та інформаційні технології»
першого (бакалаврського) рівня освіти
Орлов Володимир Андрійович
Керівники кандидат технічних наук,
доцент Осипова Наталія Володимирівна,
доктор педагогічних наук, кандидат
фізико-математичних наук, професор
Співаковський Олександр
Володимирович
Рецензент кандидат педагогічних наук,
доцент Таточенко Володимир Іванович

Херсон – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
BLOC	Business Logic Component
NoSql	Not only Structured Query Language
MVC	Model View Controller
MVP	Model View Presenter
MVVM	Model View ViewModel
UI	User Interface

ВСТУП

Інформаційні технології вже давно стали невід'ємною частиною сучасного світу. Саме вони визначають подальший економічній та суспільний розвиток людства. Не виключенням є система навчання. Проте, якщо на заході вже досягли певних успіхів в цифровій трансформації, ми тільки починаємо шлях у цьому напрямку.

Дотримуючись тенденції діджиталізації, ми вирішили розробити власний продукт, що зможе полегшити життя студентам, викладачам та іншим співробітникам вищих навчальних закладів. Більшість університетських on-line сервісів є абсолютно різними ресурсами. Це не завжди зручно, адже на кожному з них потрібно пройти процедуру реєстрації, і якщо знадобиться інформація з конкретного ресурсу – кожен раз доведеться переходи з одного на інший.

Саме тому, наша мета полягає в створенні цілої системи, яка стане універсальним інструментом для всіх учасників освітнього процесу, і буде розташована в інформаційному просторі. Розклад занять, налагоджена комунікація між студентами та викладачами, розповсюдження навчальних матеріалів, новини факультету чи університету, та багато іншого буде зосереджено в одній централізованій системі. Це великий проект, що структурно поділяється на:

- **Клієнт-частину** – мобільний додаток для абітурієнтів студентів, викладачів та випускників. За допомогою додатку користувач може переглядати свій розклад, своєчасно дізнаватися про зміни в ньому, отримувати доступ до навчальних матеріалів, дізнаватися про новини свого вишу та багато іншого;

- **Адмін-частину** – веб-додаток, в якому працівники кафедри/ факультету/університету зможуть легко складати розклад, швидко та своєчасно доносити інформацію до студентів та викладачів, ділитися новинам кафедри/факультету/університету, та інше;
- **Сервер-частину** – база даних та методи її передачі для зв'язку між клієнт/адмін-частинами. В базі даних буде зберігатись архів інформації за всю роботу системи. Наддасть змогу швидко знаходити всю інформацію щодо навчального процесу, та робити глибокий аналіз методики навчання.

Розробка такого масштабного сервісу займає багато часу та зусиль. Тому доцільніше описати всі вимоги до функціоналу сервісу та, визначивши необхідний мінімум для його запуску, розбити продукт на модулі, які будуть поступово впроваджуватись доповнюючи цей сервіс. Це дозволить досягнути при проектуванні архітектури та розробці усіх частин максимальної гнучкості та здатності до легкого масштабування.

В цій дипломній роботі описані вимоги клієнт-частини сервісу, проєктування та розроблення мобільного додатку.

Актуальність теми. Проєктування та розроблення мобільного додатку для освітніх навчальних закладів є актуальним у часи діджиталізації та всесвітньої цифрової трансформацією.

Мета дослідження: проєктування та розроблення мобільного додатку «Rozcloud».

Завдання дослідження:

- Зробити аналіз аналогів.
- Визначити вимоги до функціоналу кінцевої версії сервісу.
- Скласти діаграми прецедентів виходячи з вимог продукту;

- Розглянути сучасні підходи й технології розробки мобільних додатків;
- Спроєктувати архітектуру мобільного додатку;
- Розробити мобільний додаток.
- Провести тестування мобільного додатку;

Об'єкт дослідження: архітектура мобільного додатку.

Предмет дослідження: технології проєктування та розроблення мобільних додатків.

Практичне значення полягає в представленні студентам та викладачам вищих навчальних закладів потрібної інформації в гарному та зручному вигляді.

Структура дослідження. Дипломна робота складається зі вступу, списку умовних скорочень, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1
АНАЛІЗ АНАЛОГІВ ТА ВИЗНАЧЕННЯ ВИМОГ

1.1. Аналіз аналогів

Розробка мобільних додатків – це складний технологічний процес, що вимагає ретельного планування та виконання. Перед початком написання вимог продукту, проєктуванням та процесом розробки необхідно провести аналіз аналогів. Наш продукт не є виключенням. Для проведення аналізу були обрані додатки, що найбільше схожі с задумом нашого сервісу, мають найбільшу кількість відгуків та широкий функціонал:

- «Расписание занятий МКР» https://play.google.com/store/apps/details?id=com.mkr.shedule_app
- «Расписание занятий для школы и вузов – Weeklie» https://play.google.com/store/apps/details?id=com.numen.timetable
- «СтудЖурнал - Расписание занятий» https://play.google.com/store/apps/details?id=com.romansytnyk.studentstudio

Після проведення аналізу були виявлені основні недоліки, на які було звернено увагу:

«Расписание занятий МКР»:

- Відсутність авторизації – користувач може лише переглядати розклад у режимі гостю.
- Дані студентів та викладачів у відкритому доступі.
- При перегляді розкладу інформація не зовсім доцільно розташована, через що погано сприймається користувачем.
- Дуже обмежені налаштування додатку.
- Наявні невеликі несправності та баги.

«Расписание занятий для школы и вузов – Weeklie»:

- Розклад заповнюється користувачем.
- Майже відсутні налаштування.
- Лише один режим перегляду розкладу.
- Відсутня авторизація та персоналізація.
- Функціонал обмежений лише переглядом розкладу та заповненням домашнього завдання.
- Наявність реклами та преміум-версії.

«СтудЖурнал - Расписание занятий»:

- Розклад заповнюється користувачем.
- Відсутність автоматизації.
- Швидкість роботи програми.
- Відсутня авторизація та персоналізація.
- Функціонал обмежений лише переглядом розкладу та заповненням домашнього завдання.

Порівняльна характеристика аналогів говорить сама за себе. На даний момент не існує такого сервісу, яким в майбутньому стане “Rozcloud”.

1.2. Визначення вимог продукту

Вимоги до програмного забезпечення - сукупність тверджень щодо атрибутів, властивостей або якостей програмної системи, яка підлягає реалізації.

Ми визначили ряд головних вимог до продукту:

- Є декілька типів користувачів: студент, викладач та співробітник.
- Користувачі мають різний функціонал в залежності від типу свого облікового запису.
- Є декілька структур з ієрархічним функціоналом: кафедра, факультет, університет.
- Структури мають різний функціонал в залежності від типу.
- Структура об’єднує під собою «співробітників».
- Реєстрація користувачів відбувається користувачами з вищої ланки ієрархії: співробітник університету –> співробітник факультету –> співробітник кафедри –> викладач/студент.
- Складання розкладу відбувається в адмін-частині співробітниками.
- Користувач авторизується за наданим йому логіном та паролем.
- Студент/викладач може переглядати свій розклад за вибраним тижнем/місяцем.

На основі вимог до продукту в цілому та аналізу аналогів були виведені основні вимоги до клієнт-частини:

- Клієнт-частина повинна бути реалізована в якості мобільного додатку.
- Мобільний додаток повинен буди кросплатформним (IOS та ANDROID).
- Масштабованість – можливість додавання нових модулів з новим функціоналом.
- Гнучкість – через додавання нових модулів в наступних версіях додаток не повинен перероблюватись.
- Можливість авторизації для викладача та студента.
- Можливість переглядати персоналізований розклад в залежності від типу користувача та його належності до тих чи інших структур.

- Можливість спілкування між користувачем в залежності від налаштувань конфіденційності.
- Перегляд архівів.
- Можливість проходити опитування.
- Отримання повідомлень під час змін у розкладі.
- Можливість перегляду новинної стрічки.
- Локальне кешування даних (розклад, тощо).

Через обширний функціонал “Rozcloud” було вирішено розробляти додаток поступово: Alpha -> Beta -> Release версії.

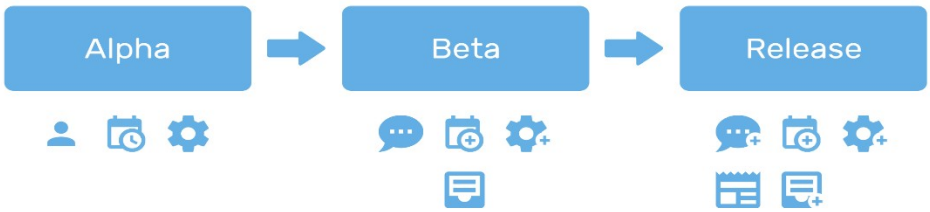


Рис. 1 Версії продукту “Rozcloud”

У Дипломній роботі йдеться про реалізацію Alpha версії продукту.

1.3. Створення діаграми прецедентів

Діаграми прецедентів застосовуються для моделювання виду системи з точки зору прецедентів (або варіантів використання). Найчастіше це передбачає моделювання контексту системи, підсистеми

або класу або моделювання вимог, що пред'являються до поведінки зазначених елементів.

У дипломній роботі діаграми прецедентів використовуються я якості моделювання Alpha версії додатку. Користувач спочатку повинен увійти в систему під наданим йому логіном та паролем, тільки після цього він зможе використовувати додаток у повній мірі. На діаграмі спроєктовано дію користувача у клієнт-частині сервісу, що описано на рисунку 1.1.

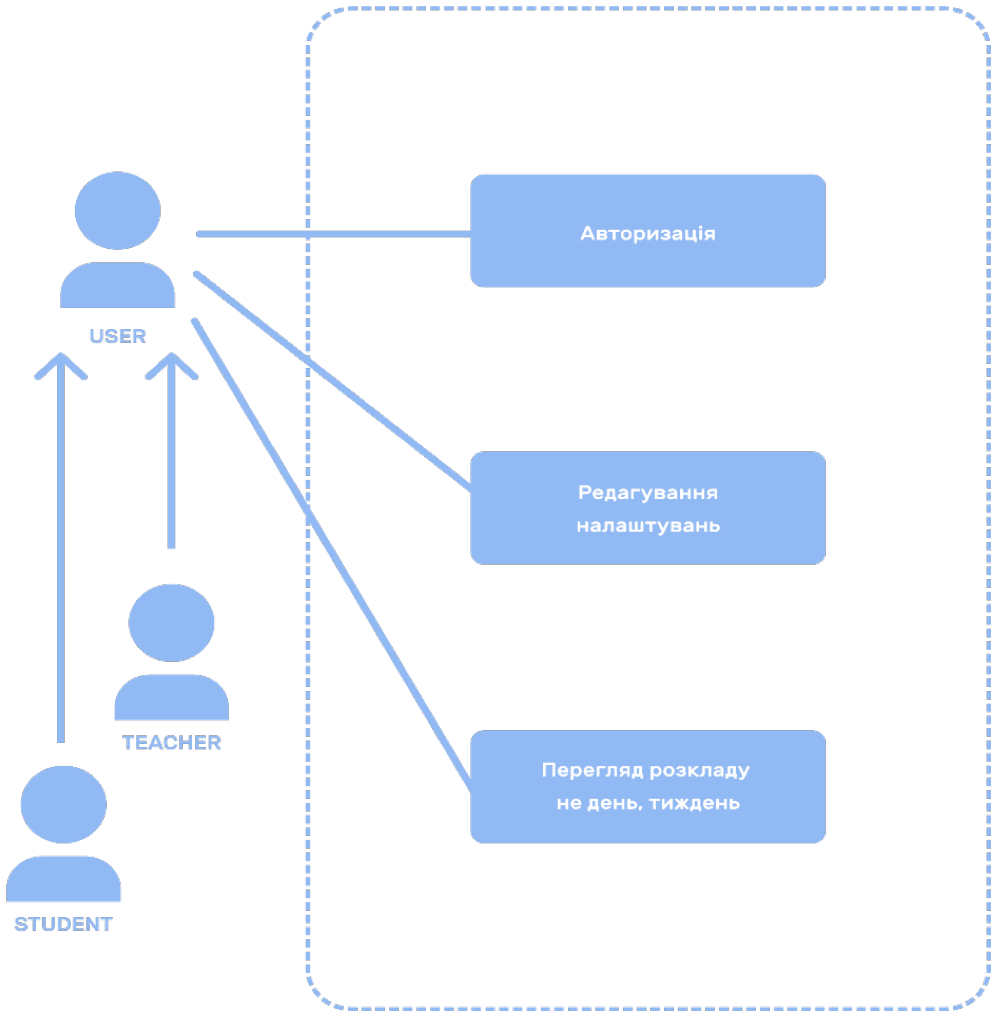


Рис. 1.1 Діаграма прецедентів

У Alpha версії додатку усі користувачі мають майже однаковий функціонал. Однак, у майбутніх версіях в залежності від типу

користувача (студент / викладач / співробітник) він буде змінюватись. Також з`явиться новий тип користувача – абітурієнт.

РОЗДІЛ 2
СУЧАСНІ ПІДХОДИ Й ТЕХНОЛОГІЇ РОЗРОБКИ

2.1. Аналіз предметної області

В основу сервісу закладена дворівнева архітектура клієнт-серверної взаємодії. Принцип роботи дворівневої архітектури взаємодії клієнт-сервер полягає в тому, що обробка запиту відбувається на одній машині (сервері) без використання сторонніх ресурсів.

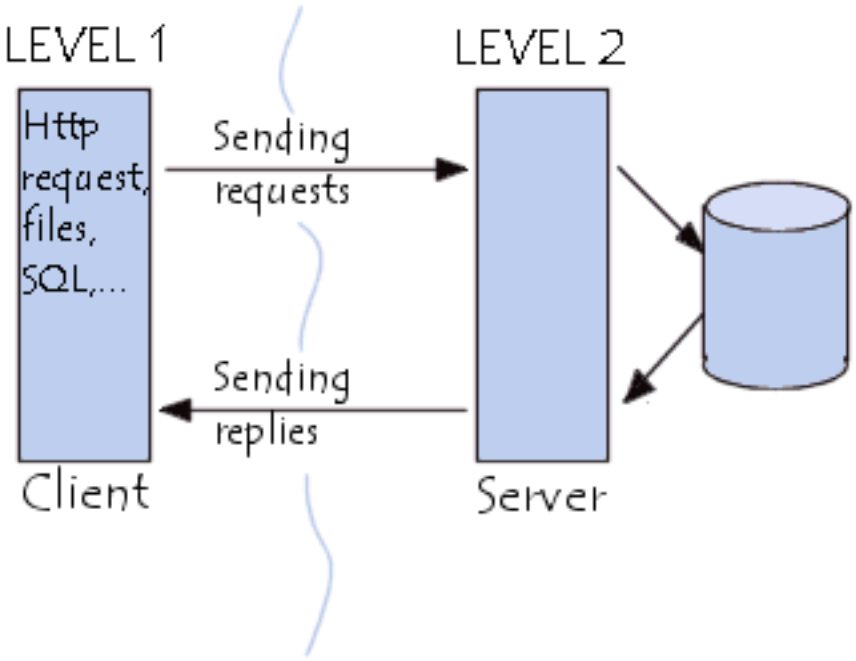


Рис. 2 Клієнт-серверна архітектура

Клієнт і сервер взаємодію один з одним в мережі Інтернет або в будь-який інший комп'ютерної мережі за допомогою різних мережевих протоколів, наприклад, IP протокол, HTTP протокол, FTP та інші [31]. Протоколів насправді дуже багато і кожен протокол дозволяє надавати ту чи іншу послугу. В нашому сервісі взаємодія між клієнтом та сервером відбувається завдяки HTTP.

Взаємодія клієнта і сервера завжди починається саме з клієнта. Сервер ж вирішує, чи давати відповідь клієнту, і якщо так - то яку саме. На стороні клієнта же йде обробка отриманих даних, і відображення їх користувачу. Це дуже здорово, адже за рахунок того, що всі важкі операції, такі як взаємодія з базою даних, виконуються на стороні сервера, ми істотно знижуємо вимога до машин клієнтів.

Що стосується платформ для реалізації клієнтської частини - їх декілька. Клієнтську частину можливо реалізувати як в якості веб-додатку, так і в якості комп'ютерної програми або мобільного додатка. У сучасному світі дуже важко уявити людину без смартфона. Він дуже щільно влився в наше життя, і знаходиться з нами завжди і всюди. Саме цей факт і став ключовим у виборі платформи для клієнтської частини.

Як вже говорилося, усі тяжкі операції відбуваються на стороні сервера, що дозволяє з легкістю використовувати телефон в якості клієнта. Хоча, новинки в світі смартфонів вже зараз дозволяють

виконувати ресурсовитратні операції, наприклад як відеомонтаж чи обробка фотографій.

Ця архітектура дозволяє розподілити навантаження і розділити функціонал між клієнтською частиною і серверної.

2.2. Технології розробки мобільних додатків

В останні роки тенденція розробки мобільних додатків значно змінилася. З випуском нових технологій, фреймворків і бібліотек стало можливим приступати до розробки з мінімальними знаннями в області програмування, навіть з інших напрямків, наприклад таких як веб.

У якості вимоги до клієнт-частини сервісу є пункт, який зобов'язує розробити саме кросплатформний додаток. Так що доведеться відмовитися від звичної “рідної” розробки, і зробити вибір серед численних кросплатформних фреймворків. Лідируючі інструменти в даному напрямку це: ReactNative, Flutter, Ionic та Xamarin. Ми зробили порівняльну таблиці цих фреймворків (таблиця 1 та 2).

	React Native	**Flutter**
Мова	JavaScript + React	Dart
Тип додатку	Кросплатформний	Кросплатформний
Реліз	2015	2017
Розробка	Facebook	Google

Спільнота	Дуже велика	Невелика
Платформи	Android, iOS, UWP	Android,iOS, Google, Fuchsia
Open Source	Так	Так
UI	Компоненти Native + Declarative UI	Вбудовані віджети
Продуктивність	Достатньо висока	Дуже висока

Таблиця 1

Таблиця 2

	Ionic	**Xamarin**
Мова	JavaScript, HTML + CSS	C# + .NET
Тип додатку	Кросплатформний, гібрідний	Кросплатформний
Реліз	2013	2011
Розробка	Drifty Co.	Microsoft
Спільнота	Велика	Велика
Платформи	Android, iOS, WEB	Android, iOS, UWP
Open Source	Так	Так
UI	HTML, CSS, віджети	HTML, CSS
Продуктивність	Середня	Достатньо висока

Більш ретельно ми вирішили зупиниться на розгляді React Native та Flutter.

Підтримка та спільнота.

Реліз у React Native відбувся раніше, ніж у Flutter. Велика спільноті, велика кількість бібліотек.

У Flutter спільнота менша, проте активно розвивається.

Основні елементи.

Віджети у React Native зручні при створенні компонентів користувацького інтерфейсу. Вони адаптуються під різний розмір екрану без додаткових зусиль з боку розробника.

Flutter ближче до “рідної” розробки, й використовує компоненти, дуже схожі на нативні.

Робота з нативними функціями.

У React Native використовують сторонні бібліотеки для доступу к основним функціям пристрою. Існують складнощі зі зміною бібліотек.

Flutter має прямий доступ до основних функцій пристрою, не потрібно використовувати додаткові бібліотеки.

Відображення.

React Native використовує Java Script Bridge для передачі даних, тому працює повільніше.

Flutter через те, що має свій графічний рушій, має високу продуктивність (60 кадрів на секунду).

Написання коду.

React Native використовує Java Script, є можливість розділити стилі та виконуваний код на різні класи, створити одну таблицю стилів для усіх елементів і застосувати для кожної частини додатку

Flutter розмітка UI невідокремлена від коду й пишеться у тому ж класі, де й знаходиться елемент.

Популярність.

React Native має більшу кількість готових рішень, та має вищу популярність.

Flutter молодий, проте достатньо популярний.

Висновок.

Після ретельного вивчення сучасних технологій ми вирішили зупиниться на Flutter. Він використовує один і той же код для всіх платформ. На ньому легко створювати красиві інтерфейси. Навідмінну від React Native, Flutter одразу компілюються в машинний код, йому не потрібен зв'язок між JavaScript і машинним кодом [3]. Flutter перевершує

конкурентів і демонструє найвищу продуктивність завдяки сучасній мові Dart і власним рушієм рендерингу

2.3. Архітектура мобільних додатків

Існує досить багато підходів для побудови складних систем з хорошою архітектурою. Незважаючи на невеликі відмінності цих підходів, у них багато спільного. Зрозуміло, вони усі задають способи розбиття програми на окремі модулі. При цьому в кожній системі як мінімум є модулі, що містять бізнес-логіку програми, і модулі для відображення даних. І кожен підхід в підсумку дозволяє побудувати систему, яка задовольняє наступні принципи:

- Архітектура в жодному разі не повинна бути залежна від фреймворків.
- Система повинна бути протестована, навіть без використання UI слою.

На рисунку 2.1 зображено приклад незалежної архітектури.

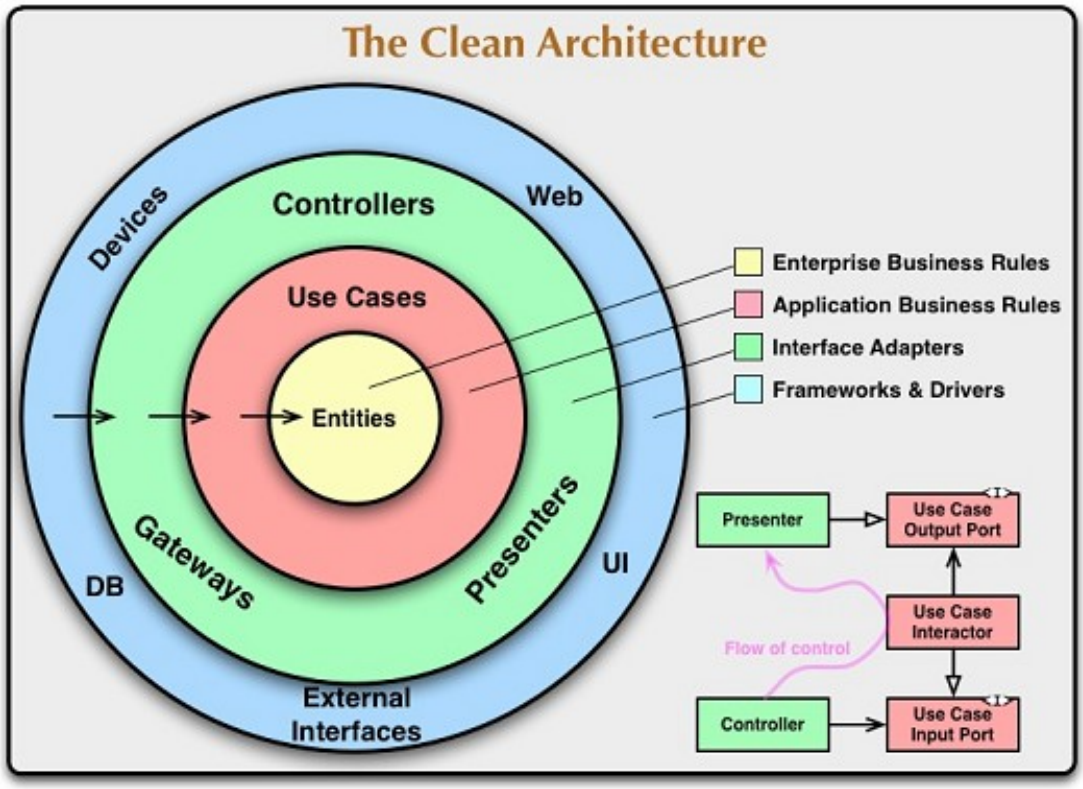


Рис. 2.1 Приклад незалежної архітектури

Така система складається з бізнес-об'єктів, об'єктів для керування даними і бізнес-логікою. Такий поділ системи на частини є досить логічним. Головне правило “чистої” архітектури – внутрішня частина нічого не повинна знати про зовнішню [25].

Саме це правило дозволяє архітектурі бути незалежною. Виходячи з концепту незалежної архітектури, оптимізуємо її для мобільного додатку.

Одна з головних ідей такої архітектури є можливість її тестування. В основному, тестування додатку відбувається у частинах бізнес-логіки та бізнес-правил, які визначають сутність самого додатку. Тому ці частини повинні бути незалежні один від одного.

Для того щоб отримати самостійність цих частин, ми розіб'ємо структуру додатку на 3 ключові слої [20]:

- Data Layer
- Domain Layer

- Presentation Layer

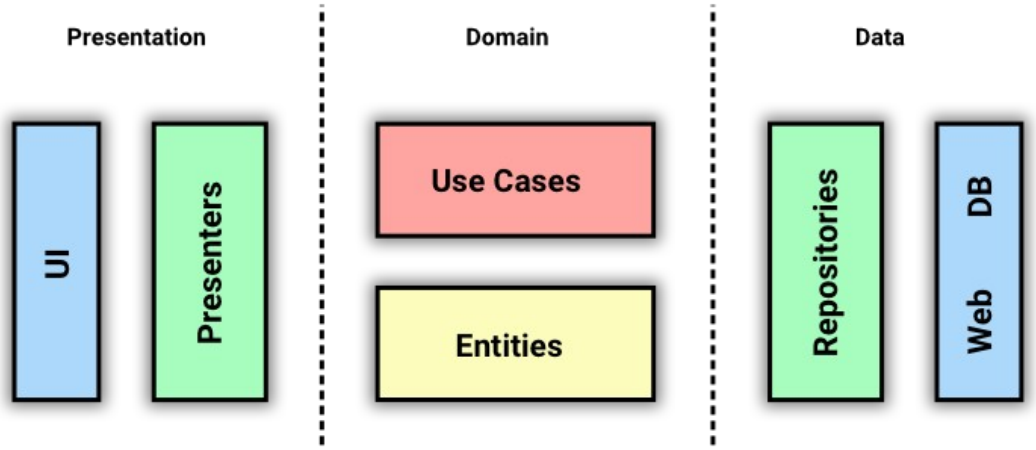


Рис 2.2 Clean Architecture від Цехаса

Роздивимось кожну частину структури окремо:

Data Layer

Дана частина відповідає в першу чергу за отримання даних з різних джерел та їх кешування.

Він реалізується за рахунок патерна Repository, і його загальну схему можна представити таким чином:

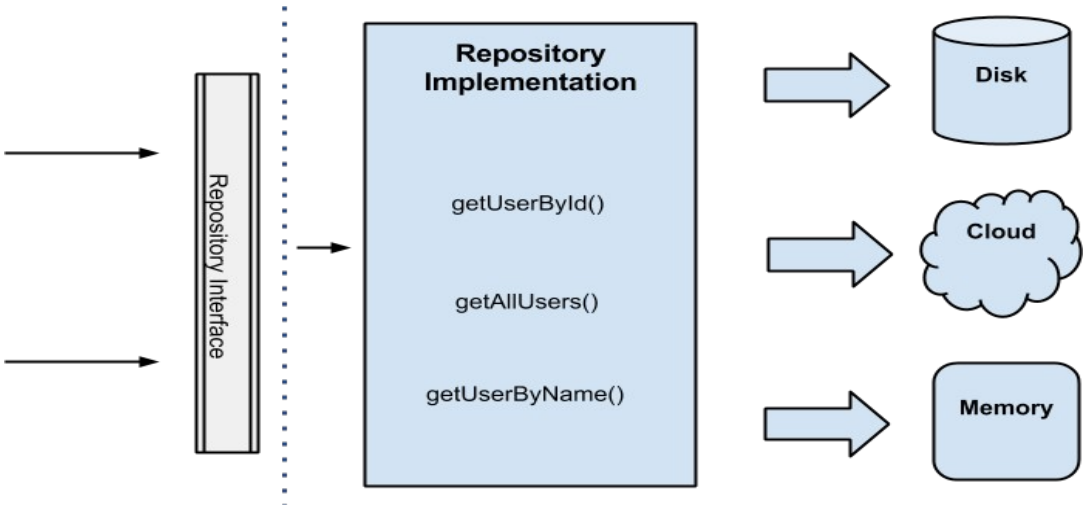


Рис.2.3 Взаємодія Data Layer частини

Наприклад, коли необхідно отримати ім’я якогось користувача, реалізація сховища перевіряє наявність цього користувача в локальному сховищі пристрою, при відсутності збереженого користувача вона відправляє запит на сервер за допомогою HTTP і отримує відповідь, який зберігається в локальне сховище і повертається.

Або, наприклад, репозиторій може завжди звертатися до сервера, а вже в разі помилки повертати збережений результат.

Плюс такого підходу в тому, що інші частини структури нічого не знають про те звідки беруться данні. Data Layer залишається єдиним джерелом інформації, і це дуже зручно, адже якщо нам доведеться брати інформацію з іншого ресурсу – нам доведеться змінювати лише цю частину проекту [22].

Domain Layer

У цій частині структури міститься, як не дивно, бізнес-логіка програми. Він є певним об'єднанням частин сценаріїв взаємодії і бізнес-логіки в оригінальній "чистій" архітектурі. Саме до цієї частини звертається частина Presentation для виконання запитів і отримання даних.

Presentation Layer

В першу чергу мобільний додаток – це взаємодія з користувачем. Тому нам потрібен спеціальна частина, яка буде відповідати за логіку відображення даних на екрані, за взаємодію з користувачем і за інші процеси, пов'язані з UI. Ця частина не повинна містити логіку додатку, не пов'язану з UI. Presentation layer допомагає організувати взаємодію бізнес-логіки з частинною даних. Він може бути реалізований з використанням будь-якого патерну, наприклад, MVC, MVP, MVVM. В нашому додатку через особливості Flutter ми будемо використовувати BloC.

Детальніше про цей патерн ми розповімо у 3 розділі дипломної роботи.

При використанні даної архітектури слід не забувати про один з головних правил – правил залежності (Dependency Rule).

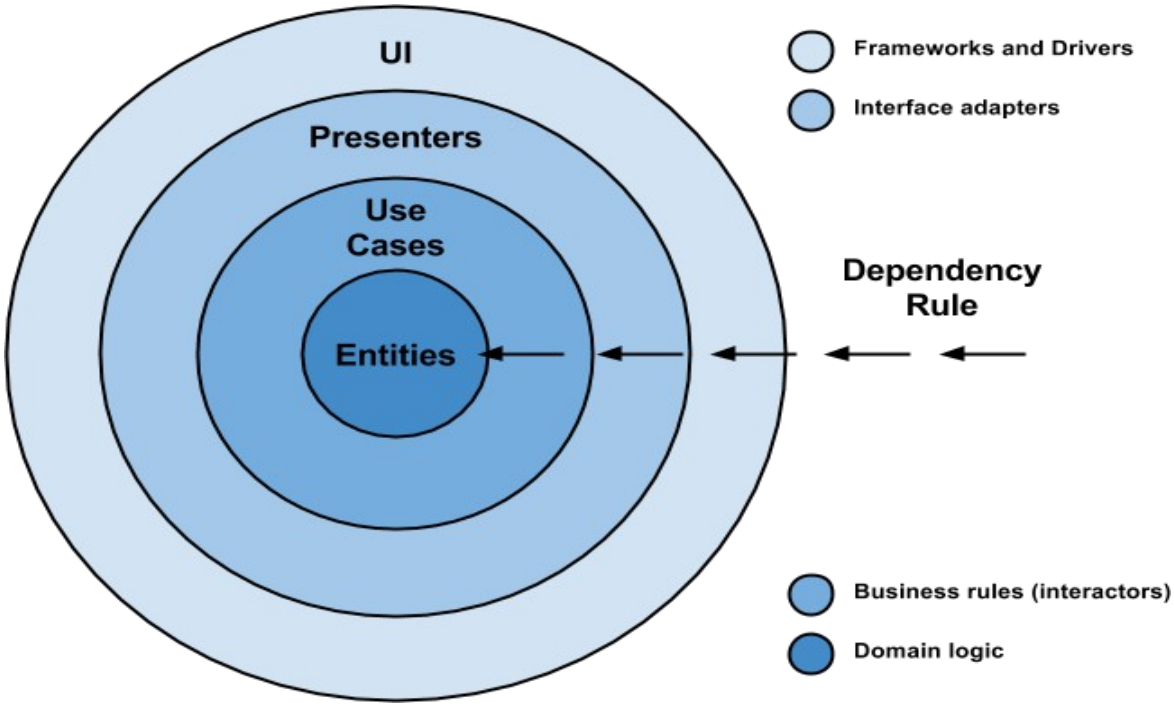


Рис. 2.4 Правила залежності

Dependency Rule говорить нам, що внутрішні частини не повинні залежати від зовнішніх. Тобто наша бізнес-логіка і логіка програми не повинні залежати від презентерів, UI, баз даних тощо. Тепер ми розуміємо, як розбити додаток на модулі і зробити ці модулі відносно незалежними один від одного.

РОЗДІЛ 3
ПРОЄКТУВАННЯ, РОЗРОБЛЕННЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Проєктування архітектури додатку

Проєктування є дуже важливим етапом створення мобільного додатку, адже від якості роботи залежить його подальше розширення. Ми максимально дотримувались вимог Clean Architecture. Розглянемо взаємодію слоїв архітектури між собою на високому рівні [23].

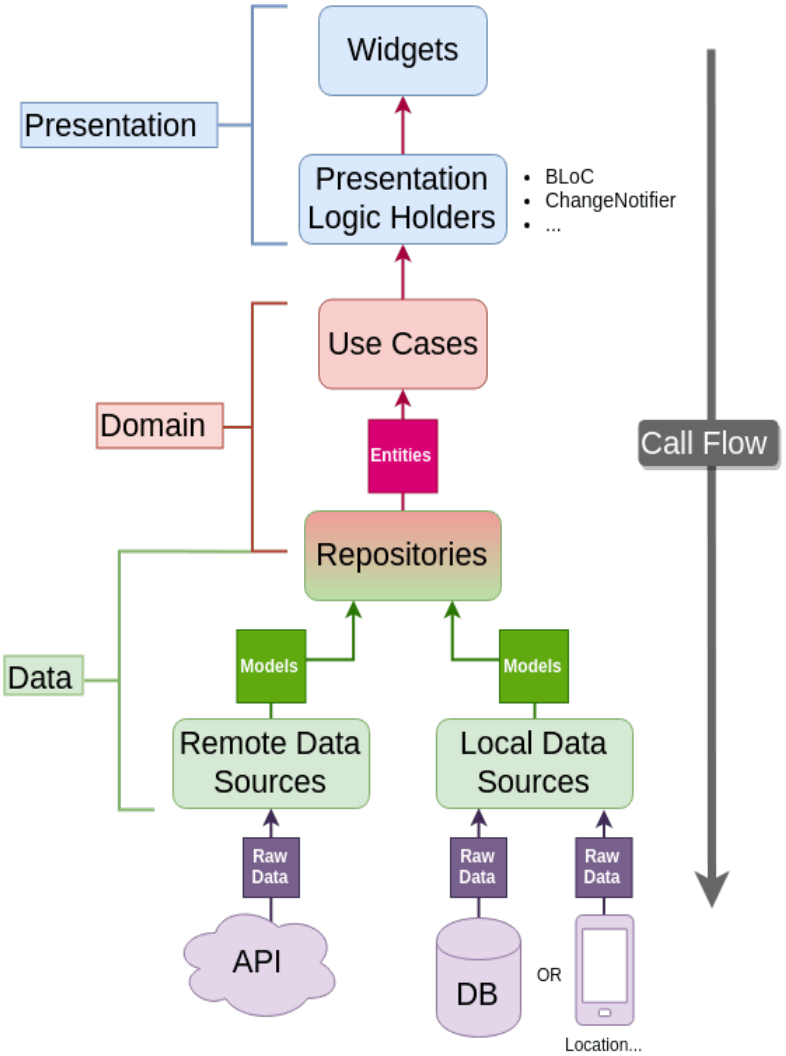


Рис 3. Взаємодія слоїв архітектури

Відповідно до стандартів обраної архітектури, ми вирішили поділити функціонал на окремі невеликі модулі, фактично тим самим відмовившись від одного великого моноліту. Кожен окремий модуль має структуру, що складається з 3 слоїв.

Розглянемо можливий сценарій взаємодії користувача та системи.

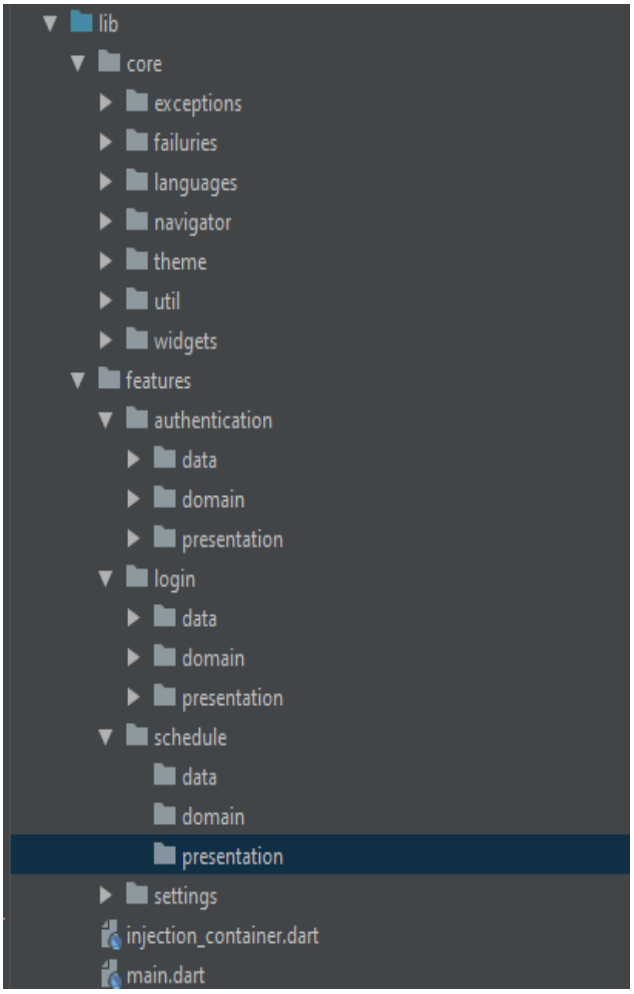


Рис 3.1 Структура модулів

Користувач натискає на кнопку, після чого йде обробка її натискання (Presentation Layer). Після обробки відбувається звернення до сховища через use case (Domain Layer). Сховище ж вирішує, до якого типу даних (локальних чи серверних) звертатися, щоб дати відповідь. У майбутньому таке розбиття дозволить нам модульно додавати функціонал додатку.

У наступних підрозділах ми ретельно розглянемо модуль “Login”.

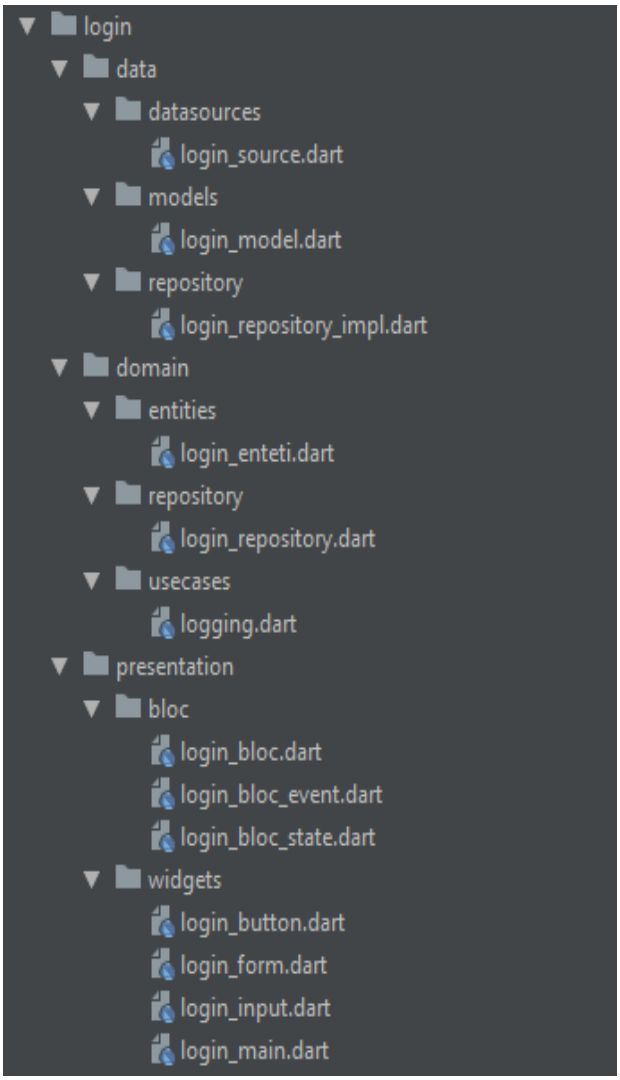


Рис 3.2 Структура Login модулю

3.2. Взаємодія користувача з інтерфейсом

Основна ідея UI Flutter полягає в тому, що ви будуєте свій користувальницький інтерфейс з віджетів. Віджети описують, як повинні виглядати елементи інтерфейсу, враховує їх поточну конфігурацію і стан. Коли стан віджету змінюється, віджет перебудовує свій опис, яке фреймворк надалі відрізняє від попереднього опису, щоб

визначити мінімальні зміни стану, які необхідно зробити [6]. Разом усі віджети утворюють дерево.

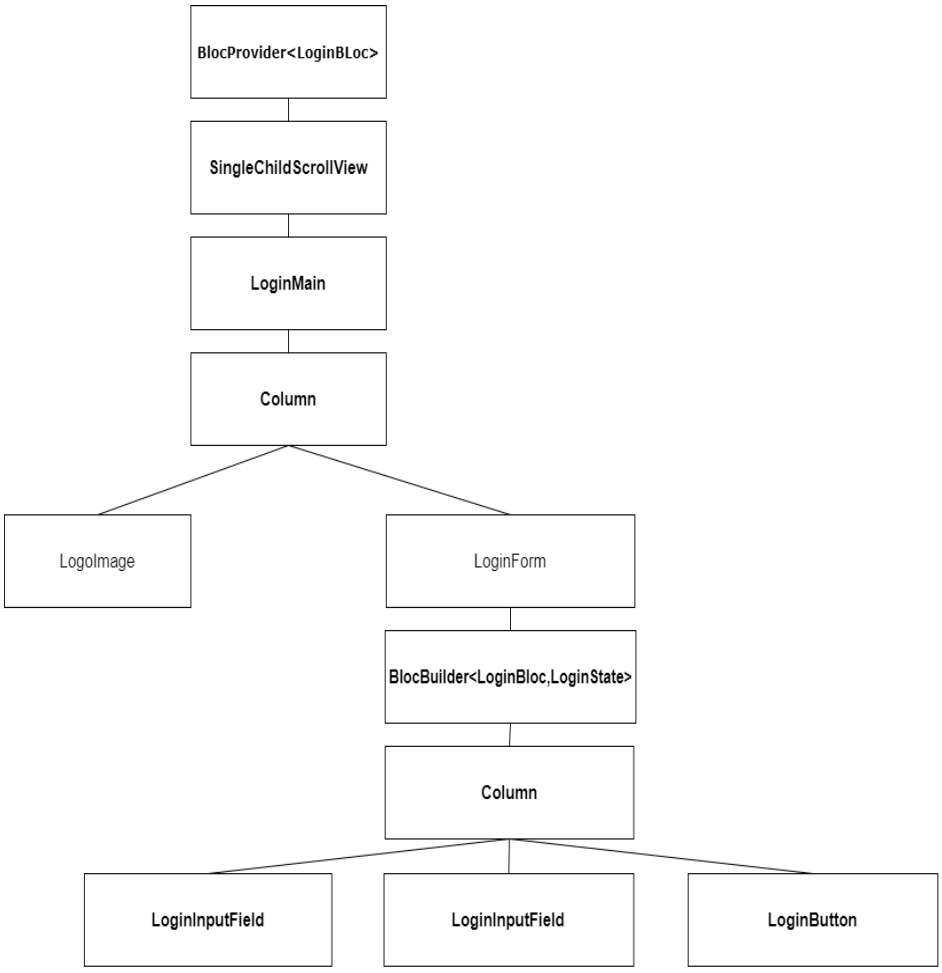


Рис 3.3 Піддерево віджетів Login екрану

Насправді, піддерево Login є набагато більшим, на малюнку 3.3 зображені лише найголовніші его віджети. Для оновлення інтерфейсу та взаємодії з Data частиною ми використали BloC.

BLoC розшифровується як Business Logic Components. Ідея BLoC полягає в тому, що все в додатку має бути представлене у якості потоку подій (Events) та станів (States) [12]. Віджети при взаємодії посилають Event, та очікують на відповідь у вигляді State, після чого оновлюють вигляд інтерфейсу [11]. (Рис 3.4)

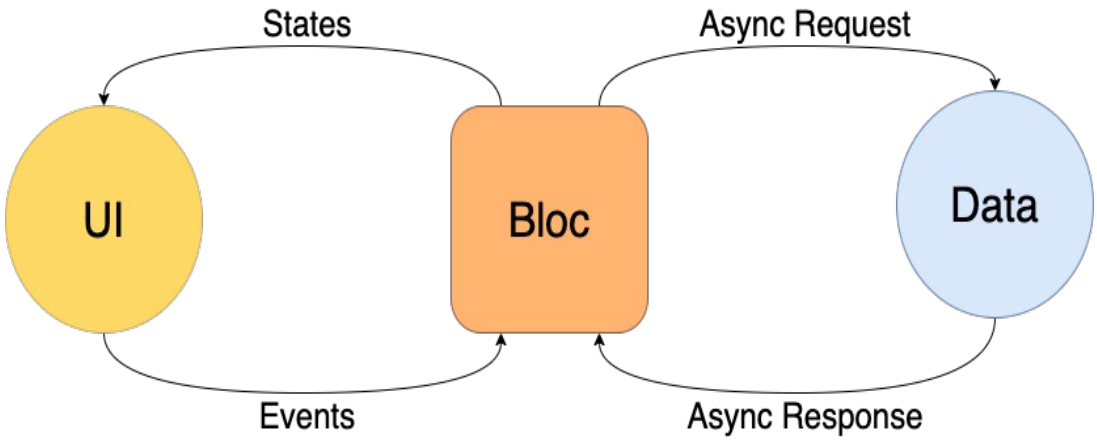
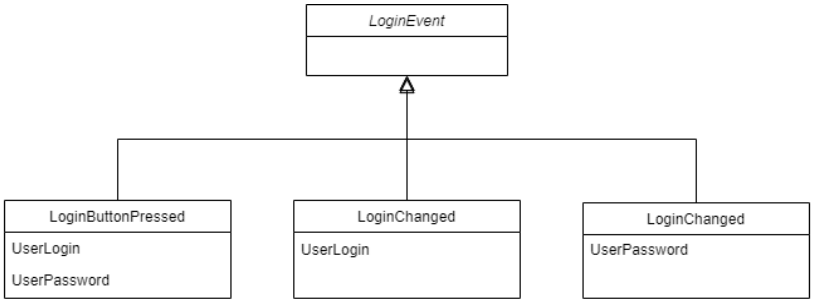


Рис 3.4 Bloc Architecture

Модуль Login складаються з наступних подій та станів.



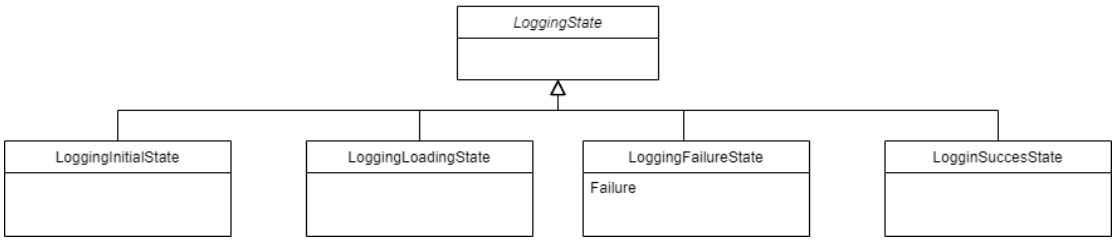


Рис 3.5 Events та States Login модулю

При натисканні кнопки Login ми відправляємо LoginButtonPressed подію у LoginBloc. У якості UserLogin та UserPassword ми беремо введені користувачем поля з LoginForm.

Після цього спочатку проходить перевірка даних, аби вони не були порожні. Якщо перевірка не проходить ми повертаємо LoggingFailureState з параметром Failure (в залежності від помилки).

Якщо перевірка пройшла успішно, ми повертаємо LoggingLoadingState, завдяки якому на екрані з`являється індикатор прогресу (аби користувач не подумав, що додаток перестав працювати).

У цей час LoginBloc звертається до Data слою, який вже буде робити запити до серверу, та очікує на відповідь.

Якщо авторизація пройшла успішно, мі кешуємо дані в локальне сховище, та відправляємо LogginSuccessState. Інакше, ми повертаємо LoggingFailureState з Failure, в якому указана помилка.

Кожен раз, коли буде повертатися State, дерево віджетів буде оновлюватись залежно від типу State.

Рис 3.6 UI екрану Login

3.3. Взаємодія з сервером

На щастя, Dart і Flutter надають інструменти для отримання даних з інтернету. Приклад тому є HTTP-package.

Відправлення запитів на сервер досить складна операція, тому їх потрібно робити асинхронно. Dart дозволяє нам працювати асинхронно, і надає async, await та future для таких типів операцій. Async та await – це ключові слова, що дозволяють Dart зрозуміти, що наступна операція повинна бути зроблена асинхронно. Future (з маленькою f) – це об`єкт класу Future, який використовуються при асинхронних операціях. Future має два стани: незавершений та завершений.

Незавершений - коли ви викликаєте асинхронну функцію. Цей future чекає, коли асинхронна робота функції закінчиться або поверне помилку.

Завершений - якщо асинхронна операція успішна, future завершується значенням. В іншому випадку future завершується помилкою.

Було вирішено використовувати public token, який додавався в Header запитів. Тим самим ми могли визначити, що запит приходить безпосередньо з мобільного додатка та ігнорувати інші.

Перед тим, як робити запит на сервер , ми проведемо процес шифрування над логіном і паролем, що раніше було передано в LoginBloc. Після цього, ми маємо додати зашифровані дані до GET запиту, і вже потім відправити на сервер.

У цей час повертається перший стан future, і ми очікуємо на відповідь сервера у якості JSON.

У разі вірних даних, авторизація буде успішною, та користувач у відповідь отримає особистий token, який ми збережемо локально. LoginBloc згенерує LoggingSuccessState, який переправить нас на

головну сторінку. Усі наступні запити будуть проводиться саме по цьому ключу, адже він є особистим ідентифікатором користувача.

У разі помилки, користувач отримає код помилки, та LoginBloc згенерує LoggingFailureState з даними о помилці. В обох випадках future вже буде завершеним.

На час написання Alpha версії було вирішено, що у один й той же момент користувач може бути авторизований лише на одному пристрої. Під час кожної успішної авторизації, сервер буде генерувати новий token.

Якщо користувач зробить запит на сервер з вже не дійсним ключем, сервер поверне помилку, під час якої усі локальні данні додатку будуть стерті з пристрою.

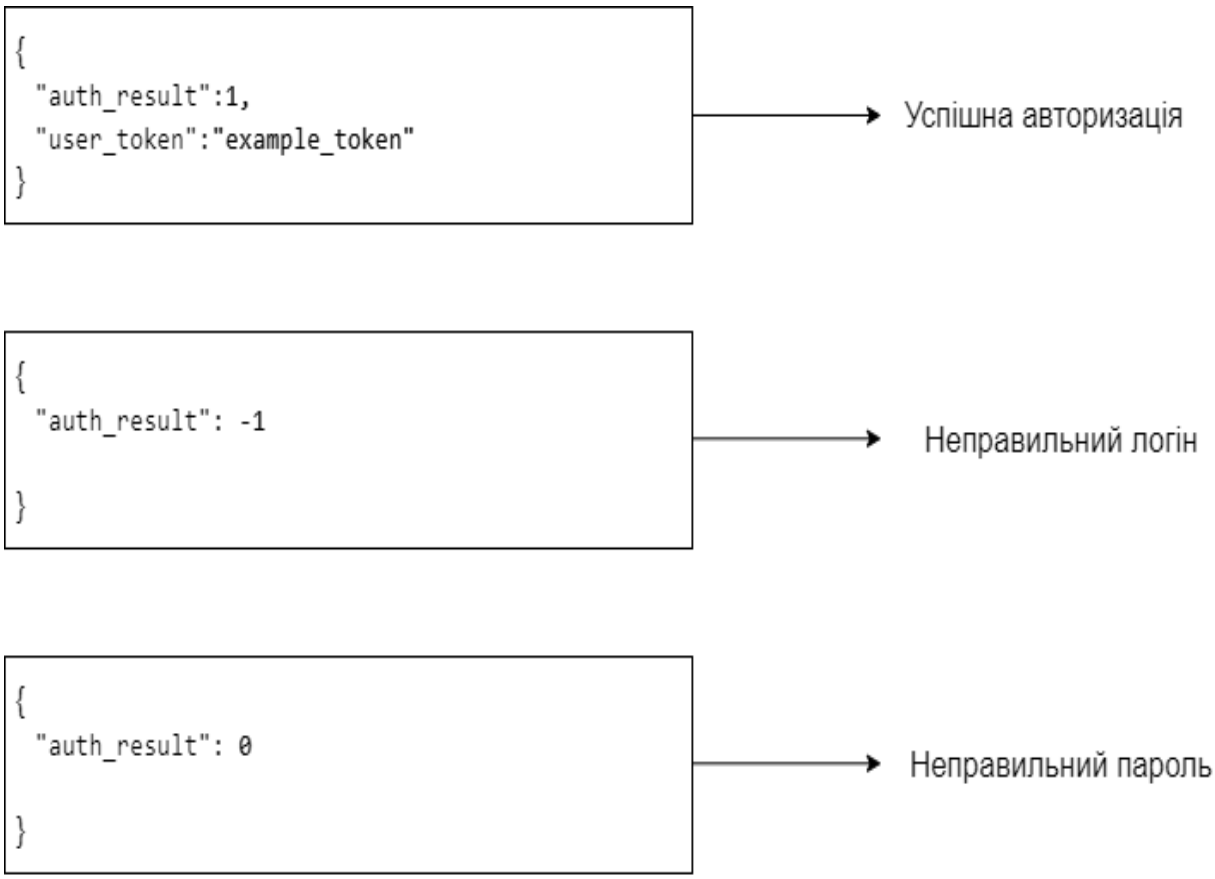


Рис 3.7 Приклади відповіді серверу

3.4. Локальне збереження даних

Для зменшення навантаження на сервер, після кожного успішного запиту ми кешуємо дані локально. При наступному відкритті додатку, в першу чергу ми будемо показувати локальні дані, і лише при необхідності звертатися до сервера за оновленням.

Як інструмент для зберігання даних локально було вирішено вибрати Hive.

Hive – це швидка NoSql база даних, яка працює за принципом ключ – значення. Є шифрувати дані, так що за допомогою цього інструменту ми також можемо зберигати приватні дані.

У порівнянні з SQLite та звичайним S.Prefs швидкість просто вражає.

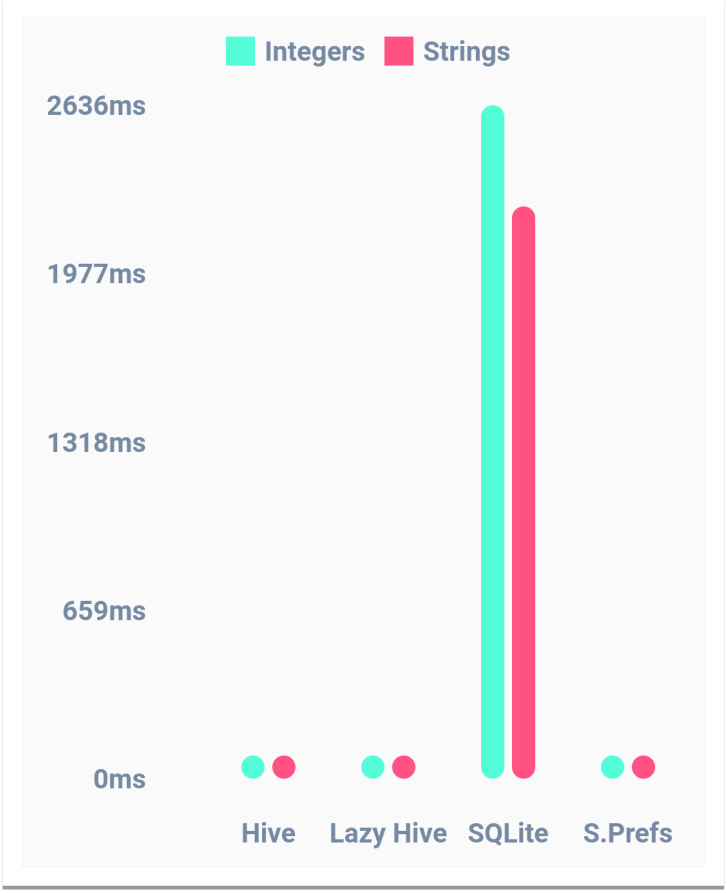


Рис 3.8 Швидкість читання за 1000 ітерацій

Рис 3.9 Швидкість запису за 1000 ітерацій

Для початку роботи з Hive, ми повинні написать адаптер для структури, яку ми будемо кешувати. Це допоможе Hive правильно зчитувати за записувати дані.

```
@HiveType(typeId: 0)
class UserDataModel extends Equatable{

  @HiveField(0)
  final String token;

  UserDataModel({
    this.token
  });


  @override
  List<Object> get props => [token];

}
```

Рис 3.9.1 Адаптер для Hive

3.5. Тестування

Функціональне тестування - це тестування програмного забезпечення з метою перевірки можливості функціональних вимог, тобто здатності програмного забезпечення в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить програмне забезпечення, які завдання воно вирішує. Використовуючи методологію функціонального тестування, перевіримо роботу авторизації та роботу с розкладом.

Тест №1

Мета: протестувати функцію авторизації користувача у додатку.

Очікуваний результат: отримання JSON відповіді про успішну авторизацію.

Вхідні дані: користувач знаходиться на екрані Login.

Процедура тестування: користувач вводить свій логін та пароль, після чого тисне кнопку “Увійти”.

Отриманий результат: співпадає з очікуваним.

Висновок: тест пройдено.

Тест №2

Мета: протестувати функцію отримання та кешування розкладу.

Очікуваний результат: отримання JSON розкладу та кушевання даних.

Вхідні дані: користувач знаходиться на екрані Home.

Процедура тестування: користувач за допомогою нижньої навігації переходить на екран Schedule.

Отриманий результат: співпадає з очікуваним.

Висновок: тест пройдено.

Юзабіліті-тестування (перевірка ергономічності) - метод оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників, випробувачів і підсумовуванні отриманих від них висновків.

Для проведення тестування учасникам надавалося мобільний пристрій з заздалегідь встановленим додатком.

Тестувальники успішно впоралися з усіма поставленими перед ними модератором тестування завданнями за прийнятний час і без яких-небудь ускладнень. В результаті тестування було виявлено, що додаток в достатній мірі має дружній інтерфейс і у користувачів не виникає нерозв'язних проблем при роботі з ним.

ВИСНОВКИ

У ході дипломної роботи була поставлена мета спроектувати та розробити мобільний додаток «Rozcloud».

Досягнення зазначеної мети здійснено шляхом вирішення таких основних завдань:

- Аналіз аналогів
- Визначення вимог до функціоналу кінцевої версії сервісу
- Складання діаграм прецедентів
- Розглядання сучасних підходів та технологій розробки мобільних додатків
- Проєктування архітектури мобільного додатку
- Розроблення мобільного додатку
- Проведення тестування мобільного додатку

Спроєктовано архітектуру та розроблено мобільний додаток відповідно всім визначеним нами вимогам та готовий до експлуатації першою версією нашого сервісу.

При подальшому розвитку продукту, «Rozcloud» може стати дуже сильному інструментом організації та управління навчальним процесом. Додаток має перспективи подальшого розвитку.

У майбутньому планується розробляти проект дотримуючись раніше встановлених версій, поступово впроваджуючи нові модулі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Как работает Flutter [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/476018/
2. Flutter TDD Clean Architecture [Електронний ресурс] - Режим доступу: https://resocoder.com/category/tutorials/flutter/tdd-clean-architecture/
3. Flutter dev [Електронний ресурс] - Режим доступу: https://flutter.dev/
4. Dart dev [Електронний ресурс] - Режим доступу: https://dart.dev/
5. Unit testing: Bloc library and Codemagic [Електронний ресурс] - Режим доступу: https://blog.codemagic.io/flutter-unit-testing-bloc-with-codemagic/
6. Flutter, Widgets, Element, RenderObject [Електронний ресурс] - Режим доступу: https://medium.com/@soo4kee/flutter-widgets-elemets-renderobjects-abafb6a1b88c
7. Створення додатку на Flutter: перші кроки [Електронний ресурс] - Режим доступу: https://dou.ua/lenta/articles/flutter-first-steps/
8. Введение в Flutter [Електронний ресурс] - Режим доступу: https://metanit.com/dart/flutter/1.1.php
9. Flutter Interact: All you need to know [Електронний ресурс] - Режим доступу: https://medium.com/@sercanyusuf/flutter-interact-all-you-need-to-know-207f5ffccfb9
10. Dart package [Електронний ресурс] - Режим доступу: https://pub.dev/
11. Bloc Pattern на простом примере [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/475404/
12. Flutter Bloc паттерн + Provider + тесты + запоминаем состояние [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/485002/

13. RxDart для самых маленьких…проектов [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/474968/
14. Тестирование Flutter приложения [Електронний ресурс] - Режим доступу: https://habr.com/ru/hub/dart/
15. Getters and Setters в Dart и Flutter [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/464095/
16. Introduction to Material Design. [Електронний ресурс] - Режим доступу: https://material.google.com/
17. First Steps with Flutter [Електронний ресурс] - Режим доступу: https://pusher.com/tutorials/flutter-widgets
18. An Introduction to Flutter: It’s all widgets! [Електронний ресурс] - Режим доступу: https://medium.com/flutter-community/an-introduction-to-flutter-its-all-widgets-aabd0f86ca1c
19. How to build responsive apps with Flutter: widgets review [Електронний ресурс] - Режим доступу: https://inveritasoft.com/blog/how-to-build-responsive-apps-with-flutter-widgets-review
20. Заблуждения Clean Architecture [Електронний ресурс] - Режим доступу: https://habr.com/ru/company/mobileup/blog/335382/
21. Коротко о главном: Clean Architecture, Robert C. Martin [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/464185/
22. Плюсы и минусы Clean Architecture [Електронний ресурс] - Режим доступу: https://ru.coursera.org/lecture/android-app-architecture/pliusy-i-minusy-clean-architecture-0CcvN
23. Чистая архитектура на Android и iOS [Електронний ресурс] - Режим доступу: https://apptractor.ru/develop/chistaya-arhitektura-na-android-i-ios.html
24. Как реализовать чистую архитектуру на Android [Електронний ресурс] - Режим доступу: https://devcolibri.com/how-to-implement-a-clean-architecture-on-android/

25. Clean Architecture [Електронний ресурс] - Режим доступу: https://medium.com/clean-code-channel/the-clean-architecture-32346e029902
26. How to implement Clean Architecture [Електронний ресурс] - Режим доступу: https://proandroiddev.com/how-to-implement-a-clean-architecture-on-android-2e5e8c8e81fe
27. Dio Connectivity Retry Interceptor [Електронний ресурс] - Режим доступу: https://resocoder.com/2020/03/23/dio-connectivity-retry-interceptor-flutter-tutorial/
28. Freezed – Data Class & Union in One Dart Package [Електронний ресурс] - Режим доступу: https://resocoder.com/2020/02/11/freezed-data-class-union-in-one-dart-package/
29. Injectable – Flutter & Dart Equivalent to Dagger & Angular Dependency Injection [Електронний ресурс] - Режим доступу: https://resocoder.com/2020/02/04/injectable-flutter-dart-equivalent-to-dagger-angular-dependency-injection/
30. Dart Const Tutorial – All You Need to Know (Const Expressions, Canonical Instances and More) [Електронний ресурс] - Режим доступу: https://resocoder.com/2020/01/06/dart-const-tutorial-all-you-need-to-know-const-expressions-canonical-instances-and-more/
31. Грамотная клиент-серверная архитектура: как правильно проектировать и разрабатывать web API [Електронний ресурс] - Режим доступу: https://tproger.ru/articles/web-api/

Додаток 1

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Орлов Володимир Андрійович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

23.04.20		Володимир Орлов
(дата)	(підпис)	(ім'я, прізвище)

ДОДАТКИ