МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра інформаційних систем

(повна назва кафедри)

ДИПЛОМНА РОБОТА

Створення мобільного додатку для керування часом та оптимізації продуктивності

Виконав(ла): студент(ка) групи ПМІ-44

спеціальності 122 – комп'ютерні науки

(шифр і назва спеціальності)

Монастирський Р.Я

(підпис) (прізвище та ініціали)

Керівник доц. Козій І.Я

(підпис) (прізвище та ініціали)

Рецензент доц. Недашковська А.М.

(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *122 – комп'ютерні науки*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою *проф. Шинкаренко Г.А.*

« *7*» *вересня* 2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Монастирському Руслану Ярославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи
Створення мобільного додатку для керування часом та оптимізації продуктивності

керівник роботи *доц. Козій І.Я*
затверджена Вченою радою факультету від « *13* » *вересня* 20 *22* р., № *15*

2. Строк подання студентом роботи *13.06.2023 р.*

3. Вихідні дані до роботи
Список літературних та інтернет джерел за тематикою роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд сучасного стану проблеми
2. Модель даних. Обрані методи та технології
3. Програмна реалізація
4. Результати апробації
5. Висновки та підсумки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схеми, діаграми, скріншоти

Презентація дипломної роботи

6. Консультанти розділів роботи

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

7. Дата видачі завдання *14.09.22 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих систем та технологій*	*Вересень-Жовтень*	
2.	*Постановка задачі*	*Жовтень-Листопад*	
3.	*Розробка схем та алгоритмів*	*Грудень-Січень*	
4.	*Програмна реалізація*	*Лютий-Березень*	
5.	*Апробація*	*Квітень-Травень*	
6.	*Оформлення роботи*	*Червень*	

Студент _____ *Монастирський Р.Я*
підпис

Керівник роботи _____ *доц Козій І.Я.*
підпис

Зміст

Вступ

У зв'язку зі стрімким розвитком технологій та поширенням мобільних пристроїв, розробка мобільних додатків стала актуальною та важливою галуззю інформатики. Дипломна робота присвячена розробці мобільного додатку, який допомагає в організації та керуванні задачами. Основна ідея додатку полягає в тому, щоб забезпечити ефективне планування робочого часу та підвищити продуктивність користувача.

У сучасному світі ефективна організація робочого процесу та підвищення продуктивності є одними з ключових факторів успіху в будь-якій сфері діяльності. Незалежно від того, чи ми працюємо над великими проектами, виконуємо рутинні завдання або розвиваємо нові ідеї, належна організація робочого часу і планування задач є критичними аспектами для досягнення максимальної продуктивності та досягнення успіху.

Однак, багатозадачність, перевантаженість і розсіяність стають основними викликами, з якими зіштовхуються багато людей у своїй професійній діяльності. Часто нам бракує системи, яка допомагала б нам керувати нашим часом та ресурсами ефективно, щоб досягти поставлених цілей. **Метою даної роботи** є розробка додатку, який надасть користувачам засоби для управління їх робочим процесом та збільшення продуктивності.

Актуальність даної роботи очевидна в сучасному швидкому темпі життя, коли час стає найціннішим ресурсом. Користувачі, будь то студенти, фахівці або підприємці, потребують ефективного інструменту, який допоможе їм керувати їхніми завданнями, пріоритетами та досягати цілей вчасно та з найкращою якістю.

Метою даної дипломної роботи є розробка додатку, який надасть зручні та потужні інструменти для планування та виконання завдань. Додаток буде базуватися на техніці помодоро, що дозволяє ефективно організовувати час та зберігати концентрацію під час роботи. Він буде підтримувати створення папок та проектів, надавати можливість встановлювати пріоритети для кожної задачі та зберігати історію виконаних робіт. Крім того, додаток буде надавати статистику щодо продуктивності користувача, що допоможе визначити ефективність робочого процесу та зробити необхідні корективи.

Ця дипломна робота має на меті розробити імплементацію такого додатку, який буде легким у використанні, зручним та функціональним для кожного користувача, незалежно від його професії та потреб. Перспективи його використання великі, оскільки він може бути корисним для широкого кола людей, які прагнуть покращити свою продуктивність і керувати своїм часом більш ефективно.

Існує шість кроків до виконання методу:

1. Складіть список завдань, які потрібно зробити найближчим часом (такі завдання називаються активними).

2. Зі списку активних завдань виберіть ті, які бажаєте зробити сьогодні.

3. Розставте приорітетність задач.

4. Увімкніть таймер на 25 хвилин. Починайте роботу.

5. Пройшло 25 хвилин - зробіть перерву на 5 хвилин. Після перерви знову запускаєте таймер на 25 хвилин

6. Повторюєте ітерації поки не виконаєте завдання

1. Постановка задачі

Розробити кросплатформений додаток для управління задачами, який допоможе користувачеві організувати свій час та ефективно виконувати завдання з використанням техніки помодоро. Додаток повинен забезпечувати створення задач, відстежування часу виконання, збереження результатів та надання статистики.

Функціональні вимоги:

- Авторизація: Додаток має забезпечувати можливість реєстрації та авторизації користувачів для збереження їхніх даних і налаштувань.
- Головний екран: Після входу в систему користувач повинен бачити головний екран зі списком папок і проектів. Користувач може створювати нові папки та проекти, редагувати або видаляти існуючі.
- Управління задачами: Користувач може створювати нові задачі у межах обраних проектів. Кожна задача має мати наступні атрибути: назва, опис, приорітет (зелений, жовтий, червоний, сірий), кількість помідорок (25-хвилинні інтервали) для виконання та дату виконання.
- Таймер Pomodoro: При початку роботи над задачею користувач може запустити таймер Pomodoro, який відраховує 25-хвилинні інтервали (помідорки) з короткими перервами між ними. Додаток повинен відображати час, залишений до завершення поточної помідорки, а також загальний час, витрачений на задачу.
- Збереження результатів: Після завершення задачі користувач може зберегти результати, які включають кількість використаних помідорок, тривалість роботи та інші відповідні дані. Результати

повинні бути збережені в базі даних для подальшого відображення та аналізу.

- Статистика: Додаток має надавати користувачу можливість переглядати статистику своєї продуктивності, включаючи загальну кількість виконаних задач, тривалість роботи над задачами, потрачений час на кожен проект, статистика продуктивності по місяцям. Статистика може бути представлена у вигляді діаграм або графіків для зручності користувача.
- Кросплатформовість: Додаток повинен бути розроблений з використанням React Native для фронтенду та .NET 6 для бекенду, забезпечуючи можливість використання на різних платформах, таких як iOS та Android.

Нефункціональні вимоги:

- Інтерфейс: Додаток повинен мати зручний та привабливий інтерфейс користувача, з урахуванням основних принципів дизайну та навігації.
- Безпека: Користувачі мають мати доступ тільки до своїх власних даних. Додаток повинен забезпечувати безпеку даних шляхом використання захисних механізмів, таких як шифрування та аутентифікація.
- Швидкодія: Додаток повинен працювати швидко та ефективно, забезпечуючи мінімальні затримки при завантаженні та обробці даних.
- Сумісність: Додаток повинен бути сумісним з останніми версіями операційних систем iOS та Android, а також забезпечувати коректну роботу на різних пристроях та роздільних здатностях екрану.

2. Аналоги

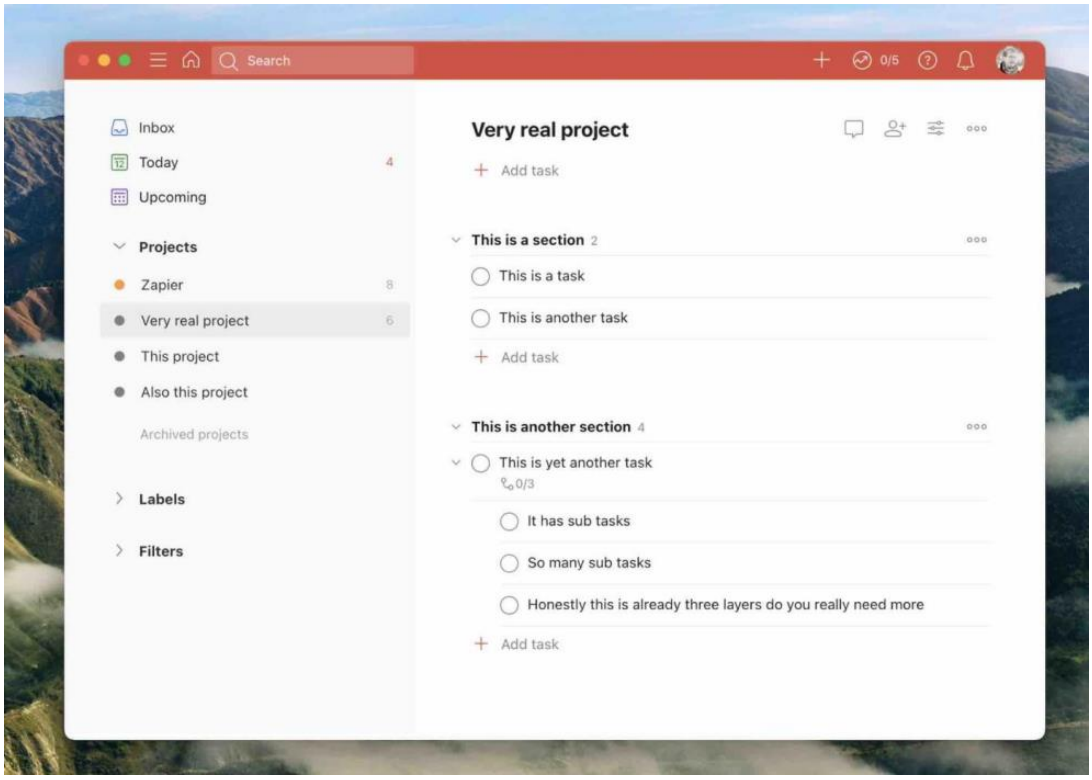


Рис. 1: Todoist від компанії Doist Inc

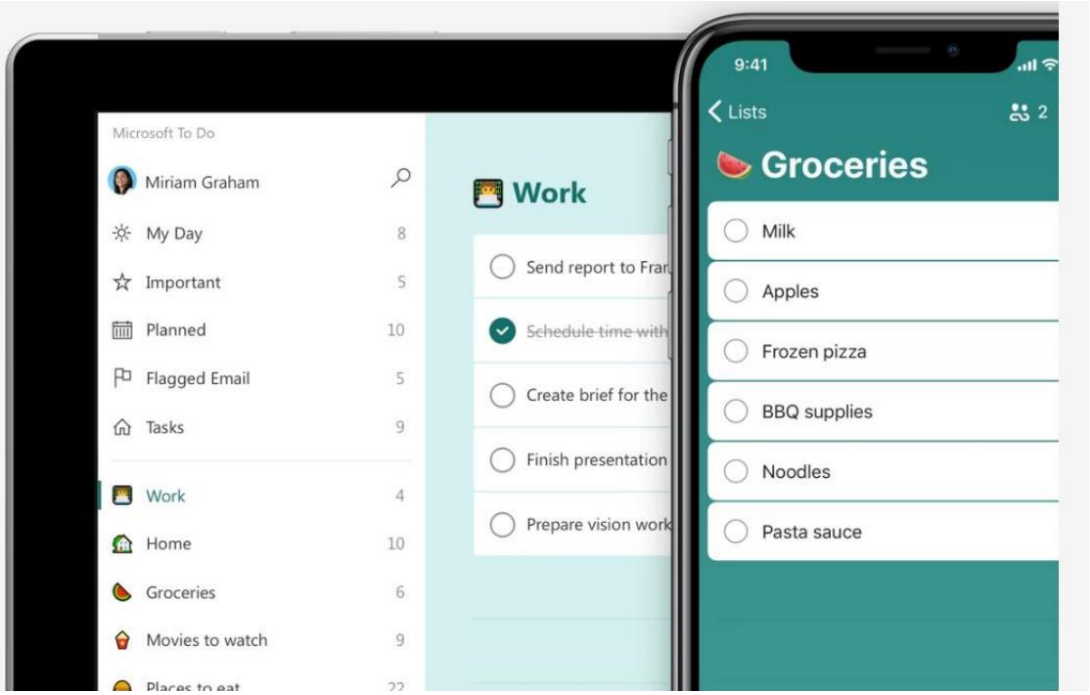


Рис. 2: Microsoft To Do

З існуючих аналогів є: Todoist від компанії Doist Inc. (Рис. 1), Microsoft To Do (Рис. 2). Ці додатки не фокусуються на самому процесі

виконання задач та відстеженні прогресу. Вони зосереджені лише на тому, щоб записати задачі та якось їх згрупувати. Саме тому я вирішив створити новий додаток який міняє підхід по ствренню та виконаню задач.

3. Огляд технологій

3.1) React Native [6] - це відкрите фреймворк для розробки мобільних додатків, який дозволяє використовувати JavaScript та React для побудови кросплатформних додатків для iOS та Android. Він був розроблений компанією Facebook і здобув велику популярність серед розробників мобільних додатків завдяки своїм перевагам із забезпеченням ефективності та швидкості розробки.

Технологія React Native базується на бібліотеці React [5], яка використовується для розробки веб-інтерфейсів. Вона дозволяє створювати компоненти, які мають власний стан та можуть оновлюватись без необхідності перезавантаження сторінки. Це дає змогу створювати додатки, які працюють плавно та реагують на дії користувача швидко.

Однією з ключових переваг React Native є можливість розробки кросплатформних додатків. За допомогою одного коду можна створювати додатки як для iOS, так і для Android. Це економить час та зусилля розробника, оскільки необхідно писати лише одну версію коду замість двох окремих для кожної платформи. Крім того, React Native надає можливість використовувати спільні компоненти, що полегшує процес розробки та підтримки.

React Native також забезпечує швидкість розробки, оскільки розробникам не потрібно володіти знаннями мови програмування для кожної платформи окремо. JavaScript [2] використовується для логіки додатків, а React Native перетворює цей код в нативний код платформи. Це

дозволяє розробникам використовувати відомий інструментарій із веб-розробки для побудови мобільних додатків.

React Native також підтримує гарну розширюваність та спільноту розробників. Існує велика кількість сторонніх бібліотек [4] та модулів, які дозволяють розширити функціональність додатків.

Однак, React Native також має деякі обмеження [7]. В деяких випадках, коли потрібно використовувати специфічні функції платформи або оптимізувати продуктивність, може знадобитися використання нативного коду. Також, в порівнянні з повноцінними нативними додатками, React Native може мати деяке обмеження у швидкості та доступі до певних функцій.

React Bridge [8] є ключовим компонентом в архітектурі React Native і відповідає за комунікацію між JavaScript-кодом та нативним кодом платформи (Objective-C/Swift для iOS, Java/Kotlin для Android).

Основна мета React Bridge - забезпечити міст між JavaScript-середовищем React Native та нативним середовищем мобільної платформи. Він дозволяє JavaScript-коду взаємодіяти з нативними компонентами, API та функціями платформи. Це забезпечує розробникам можливість використовувати специфічні можливості кожної платформи без необхідності писати повноцінний нативний код.

Робота React Bridge [9] полягає у перетворенні взаємодії JavaScript-коду з React-компонентами на виклики до нативного API платформи. Коли розробник створює React Native компонент, він може використовувати спеціальні компоненти, такі як View або Text, які відповідають нативним компонентам на кожній платформі. Коли відбувається взаємодія з такими

компонентами, React Bridge перетворює виклики JavaScript на виклики до відповідного нативного API.

React Bridge також забезпечує двосторонню комунікацію між JavaScript-кодом та нативним кодом. Це означає, що JavaScript-код може відправляти повідомлення до нативного коду, а нативний код може повертати дані та результати до JavaScript. Це дає змогу обмінюватися даними, подіями та станом між JavaScript- та нативним кодом, що робить можливим інтеграцію з функціями та фреймворками платформи.

Одним з ключових принципів роботи React Bridge є асинхронність. Оскільки взаємодія з нативним кодом може бути затратною в часі, React Bridge використовує асинхронні виклики для обміну даними та результатами між JavaScript та нативним кодом. Це дозволяє підтримувати плавну роботу додатків та уникнути блокування інтерфейсу користувача.

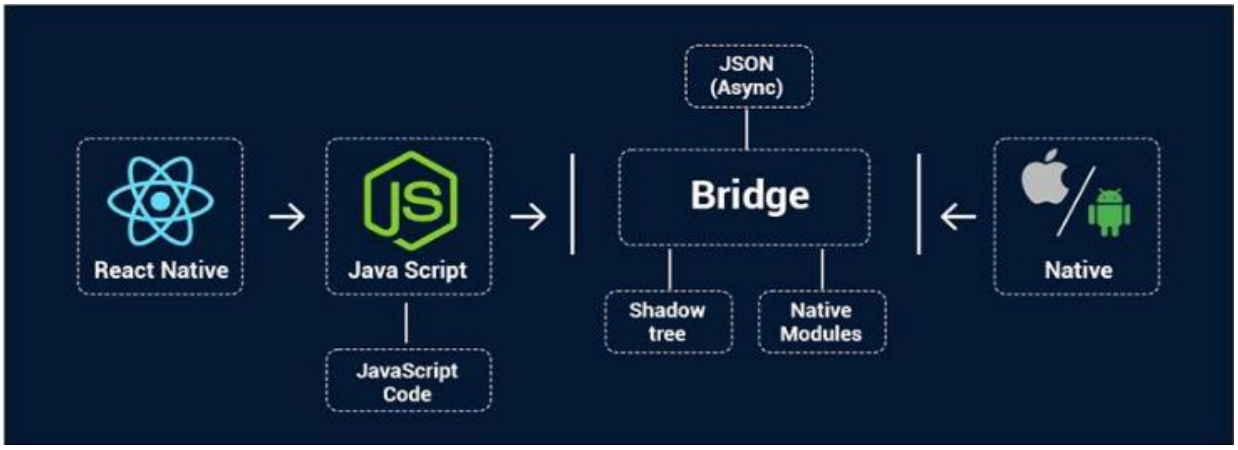


Рис.3: Загальна Схема роботи React Native Bridge

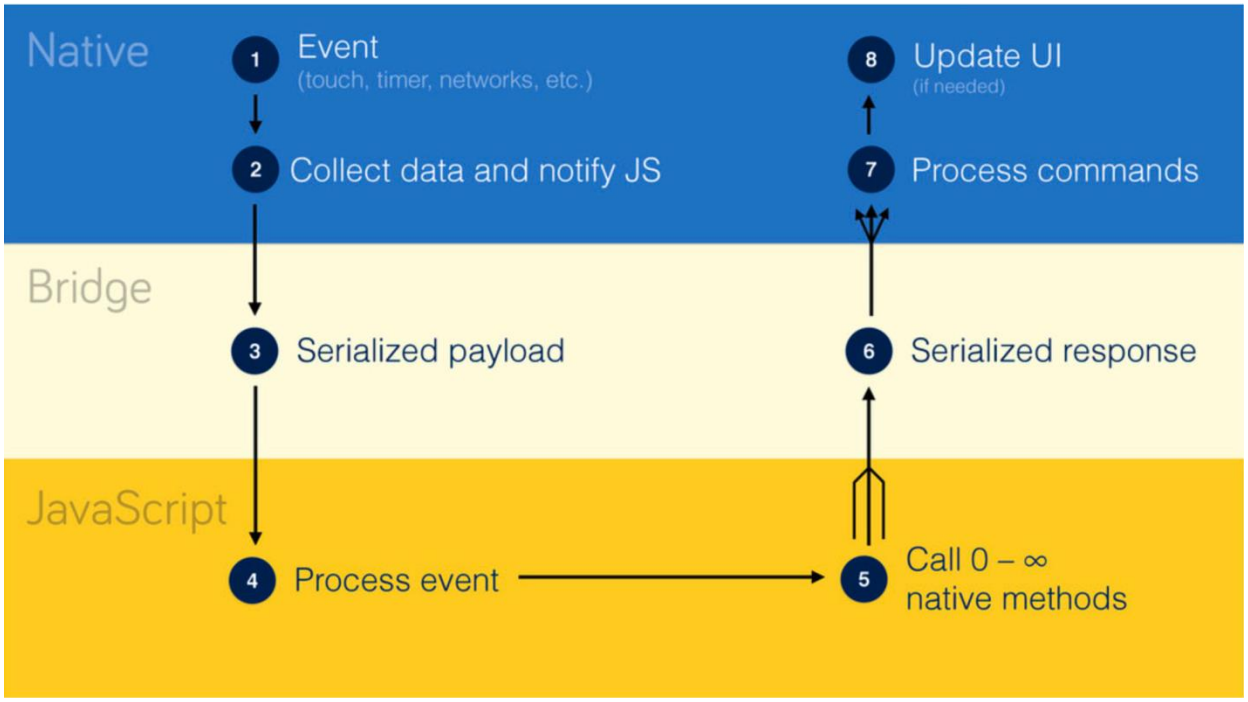


Рис. 4: Детальніша схема роботи React Native Bridge

3.2) Expo [10] - Експо для React Native є платформою розробки мобільних додатків, яка надає широкий спектр інструментів та можливостей для створення кросплатформних додатків з використанням React Native. Ця технологія стала популярною серед розробників мобільних додатків завдяки своїй простоті використання та широкому спектру функціональних можливостей.

Основною перевагою Expo для React Native є те, що вона спрощує процес розробки мобільних додатків, дозволяючи зосередитися на самому створенні додатку, а не на налаштуванні та конфігурації проекту. Основні компоненти Expo включають наступне:

1. Expo SDK [11]: Це набір інструментів, який надає доступ до функцій мобільного пристрою, таких як камера, геолокація, акселерометр, пуш-сповіщення та інші. SDK дозволяє розробникам взаємодіяти з цими функціями безпосередньо з JavaScript-коду.

2. Expo Client [12]: Це мобільний додаток, який можна встановити на смартфон або планшет для швидкого перегляду та тестування додатків, розроблених з використанням Expo. Expo Client дозволяє розробникам переглядати зміни в реальному часі під час розробки.
3. Expo CLI: Це інтерфейс командного рядка, який дозволяє створювати, налаштовувати та управляти проектами Expo. Він забезпечує швидку розробку, запуск та розгортання додатків Expo для різних платформ, таких як iOS, Android та веб.
4. Expo Snack [13]: Це онлайн-середовище для розробки мобільних додатків, де ви можете створювати та тестувати код безпосередньо в браузері. Expo Snack дозволяє вам швидко розпочати розробку без необхідності налаштовувати середовище розробки на локальному комп'ютері.

Окрім цього, Expo має вбудовану систему для легкої дистрибуції та оновлення додатків, що називається Over-the-Air (OTA) [14] оновлення. Це дозволяє розробникам оновлювати додатки на пристроях користувачів без необхідності оновлювати додаток через магазини додатків (App Store або Google Play).

Загалом, Expo для React Native є потужною та зручною платформою для розробки мобільних додатків. Вона спрощує процес розробки, надаючи широкий спектр функцій та інструментів, а також дозволяє швидко тестувати та розгортати додатки на різних платформах.

3.3) Redux [15] - Redux є популярною бібліотекою управління станом для JavaScript додатків, включаючи React Native. Вона дозволяє зберігати стан додатку в одному місці (централізовано) і легко оновлювати його відповідно до змін в додатку.

Redux є популярною бібліотекою управління станом для JavaScript додатків, включаючи React Native. Вона дозволяє зберігати стан додатку в одному місці (централізовано) і легко оновлювати його відповідно до змін в додатку.

Redux працює на основі концепції одностороннього потоку даних. Всі зміни стану додатку відбуваються шляхом відправлення дій (actions), які описують, що сталося в додатку. Ці дії переходять через функцію-редуктор (reducer), яка обробляє дію і оновлює стан додатку. Редуктор є чистою функцією, яка не змінює поточний стан, а повертає новий стан на основі попереднього стану та отриманої дії.

Основні складові Redux для React Native:

- Store (Сховище): Це централізований об'єкт, який містить стан додатку. Він зберігається в пам'яті і доступний для всіх компонентів додатку. Redux-сховище створюється на основі кореневого редуктора (root reducer), який комбінує різні редуктори в додатку.
- Actions (Дії): Дії описують події або зміни, які відбуваються в додатку. Вони є простими об'єктами JavaScript, які мають тип (type) і необов'язкові дані (payload). Дії відправляються до Redux-сховища за допомогою функцій, які називаються дією-створювачами (action creators).
- Reducers (Редуктори): Редуктори визначають, як стан додатку змінюється відповідно до дій. Вони приймають поточний стан і дію, і повертають новий стан. Redux рекомендує розділити редуктори на менші, незалежні функції, які керують конкретними частинами стану.
- Connect (З'єднання): Для зв'язку React Native компонентів з Redux-сховищем використовується функція connect. Ця функція підключає

компонент до Redux-сховища, надаючи йому доступ до стану і функцій дії-створювачів.

- Middleware (Проміжний програмний засіб): Middleware в Redux дозволяє розширити функціональність додатку, виконуючи додаткові дії перед тим, як дія досягне редуктора. Middleware може бути використаний для логування, асинхронної обробки, маршрутизації тощо.
- React Redux: Це офіційна бібліотека, яка надає зручні засоби інтеграції між React і Redux. Вона включає в себе компоненти, такі як Provider і connect, що допомагають роботі з Redux у React Native додатках.

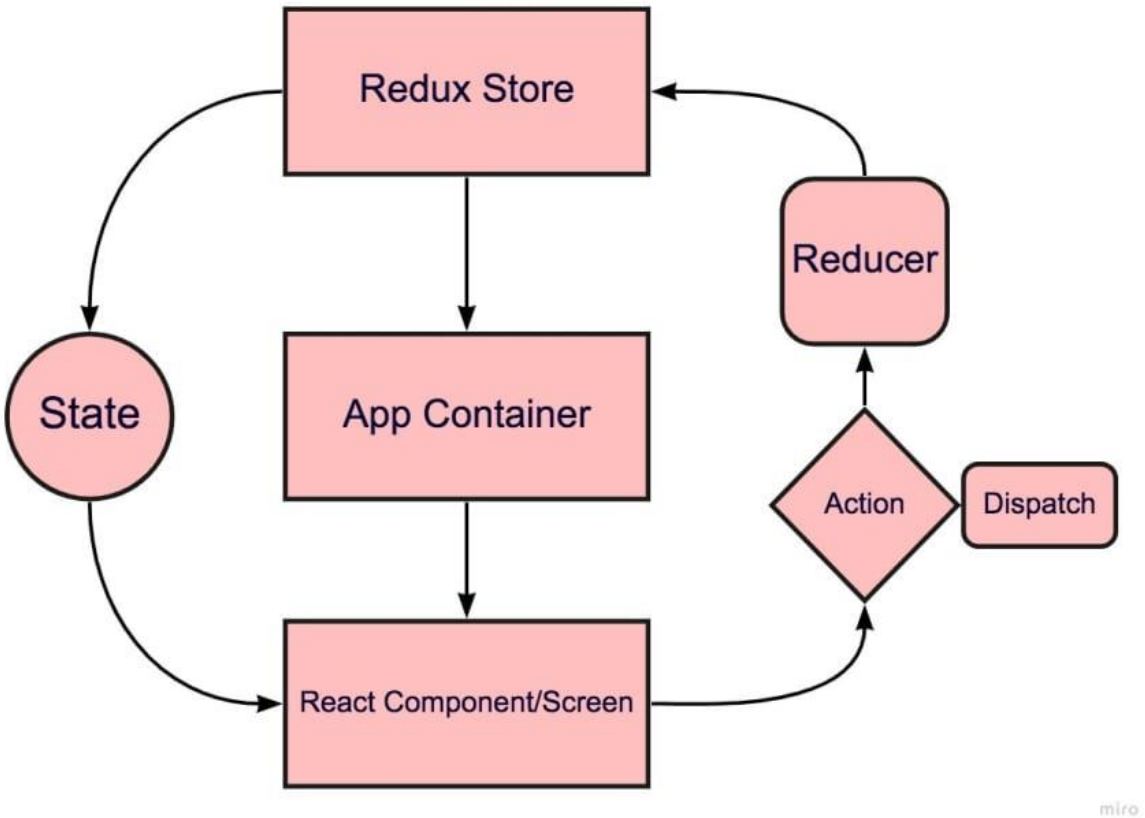


Рис. 5: Схема роботи Redux

Redux для React Native дозволяє легко керувати станом додатку, зменшує кількість пропусків даних між компонентами і полегшує

відлагодження. Використання Redux також сприяє створенню масштабованих і легко тестируємих додатків.

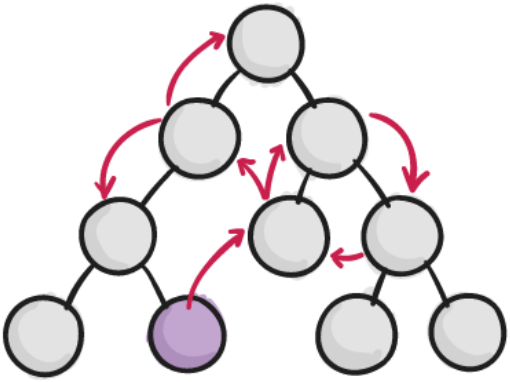


Рис. 6: Керування станами без Redux

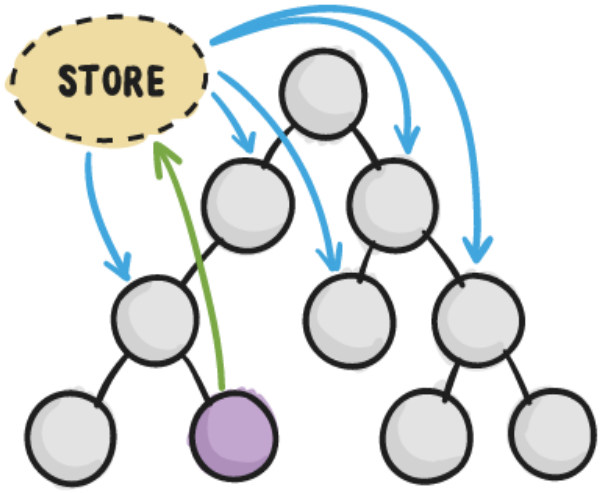


Рис. 7: Керування станами з Redux

3.4) React Navigation - React Navigation - це бібліотека для навігації в React Native додатках. Вона надає різні компоненти та навігатори, такі як Stack Navigator [16], Drawer Navigator [17] та Tab Navigator [18], що дозволяють створювати складні навігаційні структури для додатку

3.5) Axios [31] - Axios - це бібліотека, яка надає можливість здійснювати HTTP-запити з використанням JavaScript у браузері або на стороні сервера. У контексті React Native, Axios є популярним вибором для виконання мережевих запитів.

Особливості Axios для React Native:

- Простий у використанні: Axios має простий та зрозумілий API, що дозволяє легко виконувати HTTP-запити. Він надає методи для

різних типів запитів, таких як GET, POST, PUT, DELETE тощо, і дозволяє передавати параметри, заголовки та обробляти відповіді.

- Підтримка обіцянок [19] : Axios використовує обіцянки (promises) для обробки асинхронних запитів. Це дозволяє зручно управляти послідовністю запитів та обробкою відповідей.
- Підтримка перехоплення запитів [20] : Axios надає можливість перехоплювати та обробляти запити та відповіді, що дозволяє додатково налаштовувати поведінку мережевих запитів. Наприклад, ви можете додати перехопник для авторизації та обробки помилок.
- Завантаження та відстеження прогресу: Axios підтримує завантаження файлів та відстеження прогресу завантаження чи відвантаження даних. Це корисна функція для реалізації функціоналу завантаження або завантаження файлів у React Native додатках.
- Передача заголовків та параметрів: Axios дозволяє вказувати заголовки та параметри для запитів. Це дозволяє передавати дані, такі як токени авторизації, і налаштовувати запити залежно від потреб додатку.
- Перехоплення помилок [21]: Axios дозволяє обробляти та перехоплювати помилки, що виникають під час виконання запитів. Це дозволяє елегантно обробляти помилки та відповідати на них у вашому React Native додатку.

3.6) NativeBase [22]- NativeBase є популярною бібліотекою компонентів для розробки мобільних додатків на платформі React Native. Вона надає широкий набір готових компонентів із зручним API для швидкої та ефективної розробки інтерфейсу користувача.

NativeBase використовує React Native, що дозволяє розробникам створювати кросплатформові мобільні додатки з використанням

JavaScript. Вона пропонує широкий набір готових компонентів, таких як кнопки, тексти, списки, каруселі, форми, заголовки тощо, які можна використовувати для швидкої побудови інтерфейсу користувача. Це дозволяє зосередитися на функціональності додатку, а не на вирішенні дизайнерських задач з нуля.

Основні переваги NativeBase:

- Модульність [23]: NativeBase розбита на набір незалежних компонентів, що дозволяє легко вибирати саме ті компоненти, які потрібні для вашого проекту. Кожен компонент має свої налаштування та може бути легко настроєний під ваші потреби.
- Підтримка різних платформ: NativeBase надає можливість створювати додатки для iOS та Android з використанням єдиної бази коду. Вона надає стилізовані компоненти, які автоматично адаптуються до вигляду платформи, що полегшує створення нативного вигляду додатку.
- Налаштування стилів: NativeBase надає широкі можливості налаштування стилів компонентів. Ви можете змінювати кольори, шрифти, розміри та багато іншого. Це дозволяє вам створювати унікальний вигляд для вашого додатку.
- Підтримка тем [24]: NativeBase підтримує використання тем, що дозволяє легко змінювати зовнішній вигляд всього додатку за допомогою одного файла. Ви можете створити кілька тем і швидко перемикатися між ними без необхідності зміни коду компонентів.
- Широкі можливості розширення: NativeBase дозволяє легко розширювати компоненти або створювати власні компоненти на основі наявних. Ви можете додавати власний функціонал, стилі та властивості до компонентів NativeBase, що полегшує роботу з ними і дає велику гнучкість.

3.7) AsyncStorage [25] - AsyncStorage, є модулем в React Native, який надає інтерфейс для зберігання даних на мобільних пристроях. Він дозволяє розробникам зберігати дані на довготривалій основі, навіть після закриття додатка або перезапуску пристрою. AsyncStorage використовує асинхронний підхід для зберігання даних, що дозволяє забезпечити плавну роботу додатків.

Основні особливості AsyncStorage:

- Простота використання: AsyncStorage має простий API, який дозволяє легко зберігати, отримувати та видаляти дані. Це робить його ідеальним інструментом для зберігання простих налаштувань, токенів аутентифікації, стану додатків тощо.
- Крос-платформеність: AsyncStorage підтримується на різних платформах, таких як Android та iOS, що робить його універсальним рішенням для зберігання даних на мобільних пристроях, що працюють під управлінням React Native.
- Асинхронна робота: AsyncStorage використовує асинхронні операції для зберігання даних. Це означає, що ви можете здійснювати операції зберігання та отримання даних без блокування інтерфейсу користувача. Всі операції зберігання та отримання даних виконуються в фоновому режимі.
- Обмежений обсяг: AsyncStorage має обмеження на обсяг даних, які можна зберегти.

3.8) React Native SVG [26] - це бібліотека, що дозволяє використовувати векторну графіку в React Native додатках. Вона базується на популярному стандарті Scalable Vector Graphics (SVG) і надає можливість створювати та відображати векторні графічні об'єкти, такі як лінії, криві, фігури, текст та інші елементи. За допомогою React Native SVG можна контролювати розмір, положення, кольори та інші атрибути вашого векторного

зображення. Ця бібліотека також підтримує інтерактивність, дозволяючи додавати обробники подій до векторних елементів. Встановлення React Native SVG виконується так само, як і встановлення будь-якої іншої залежності через пакетний менеджер.

3.9) Victory Native [27] - це бібліотека для візуалізації даних у React Native додатках. Вона є адаптацією популярної бібліотеки для веб-розробки - Victory, створеної командою Formidable Labs. Victory Native дозволяє створювати різноманітні графіки, такі як лінійні графіки, кругові діаграми, стовпчасті графіки та багато інших. Ця бібліотека має багато можливостей для налаштування графіків, таких як вісі, маркери, кольори та стилі, що дозволяє створювати гарні та інтерактивні візуалізації даних у додатку.

3.10) ESLint [28] - є популярним інструментом для аналізу та виявлення потенційних помилок в коді JavaScript. Він дозволяє забезпечити однорідність та якість коду, а також допомагає відстежувати найкращі практики програмування.

ESLint може бути інтегрований в проект React Native для забезпечення високої якості коду. Він дозволяє встановлювати правила, які виконуються під час аналізу коду, і показує потенційні помилки та стилістичні недоліки. Основні переваги використання ESLint в проекті React Native включають:

- Консистентність коду: ESLint допомагає визначити та забезпечити стандартизацію коду в проекті. Він може перевіряти розділення рядків, використання пробілів, правильне використання крапок з комами та багато іншого.
- Потенційні помилки: ESLint аналізує код на предмет помилок, таких як неправильне використання змінних, невизначені змінні, неправильне використання функцій тощо. Це допомагає уникнути

потенційних помилок, які можуть вплинути на працездатність додатку.

- Підтримка ECMAScript-стандартів: ESLint підтримує останні версії ECMAScript та дозволяє використовувати нові функції та синтаксис JavaScript. Це дозволяє вам використовувати сучасні можливості JavaScript у своєму проекті React Native.
- Налаштування правил: ESLint надає широкі можливості для налаштування правил аналізу коду. Ви можете вибрати правила, які відповідають вашим пот

3.11) .NET Core [29] - це відкрита універсальна платформа розробки з відкритим кодом, яка підтримується корпорацією Майкрософт та спільнотою .NET на сайті GitHub за посиланням https://github.com/dotnet/core. Вона є кросплатформною, підтримує Windows, Mac OS та Linux

3.12) .NET CLR (Common Language Runtime) [33] - є ключовою складовою .NET, яка забезпечує виконання програмного коду, написаного на мовах програмування, які підтримуються платформою .NET. CLR є віртуальною машиною, що забезпечує виконання і керування програмним кодом, а також рядом сервісів для оптимізації продуктивності, безпеки та керування ресурсами.

Основні характеристики CLR включають:

- Виконання коду: CLR забезпечує виконання інтерпретацією або компіляцією програмного коду. Спочатку код перетворюється на проміжний мовний код (IL), який після цього може бути виконаний відразу (JIT-компіляція) або компільований заздалегідь (pre-JIT-компіляція).

- Керування пам'яттю: CLR відповідає за керування пам'яттю, автоматично виділяючи та звільняючи пам'ять для об'єктів. Вона також включає збірник сміття, який відповідає за виявлення та видалення непотрібних об'єктів з пам'яті.
- Безпека: CLR забезпечує безпеку виконання програмного коду, використовуючи механізми, такі як перевірка типів, обмеження доступу до ресурсів та контроль надправ. Це допомагає уникнути потенційно небезпечних дій, таких як переповнення буферу або недозволені дії з файловою системою.
- Керування виключеннями: CLR має вбудовану систему керування виключеннями, що дозволяє виявляти та обробляти помилки в програмі. Вона забезпечує можливість ловити та обробляти виключення на різних рівнях виконання коду.
- Підтримка багатьох мов програмувань: CLR надає підтримку для різних мов програмування, таких як C#, VB.NET, F# і т. д. Це означає, що програмний код, написаний на різних мовах, може бути компільований в спільний проміжний код і виконуватись в CLR.
- Інтеграція зі збіркою: CLR дозволяє створювати та використовувати збірки, які є основною одиницею розповсюдження та виконання програм на .NET. Збірка може містити один або кілька модулів, які складаються з IL-коду, ресурсів та метаданих.
- Інструменти розробки: CLR надає широкий спектр інструментів розробки, таких як відлагоджувач (debugger), профайлери продуктивності та інші, що допомагають розробникам створювати, тестувати та налагоджувати програми на платформі .NET.

3.13) ASP.NET Core [46] - є сучасною і потужною технологією для створення веб-додатків та API. Вона розроблена компанією Microsoft і є еволюційним наступником попередньої версії ASP.NET Web API.

ASP.NET Core API [34] надає розширені можливості для розробки надійних, масштабованих та високопродуктивних веб-додатків.

Основні риси та переваги ASP.NET Core API:

- Кросплатформенність: ASP.NET Core побудований на основі .NET Core, що дозволяє розробляти та запускати додатки на різних платформах, таких як Windows, macOS та Linux.
- Швидкодія: ASP.NET Core API використовує оптимізовану архітектуру та новий пайплайн обробки запитів, що забезпечує високу продуктивність та масштабованість додатків.
- Модульність: ASP.NET Core API побудований на основі модульної архітектури, де функціональність додатку може бути розширена за допомогою пакетів NuGet. Це спрощує розробку та підтримку додатків.
- Вбудована підтримка сервіс-контейнера: ASP.NET Core API має вбудований сервіс-контейнер, що дозволяє легко використовувати впровадження залежностей та керувати життєвим циклом об'єктів.
- Підтримка формату JSON: ASP.NET Core API надає вбудовану підтримку для роботи з форматом JSON, що дозволяє передавати дані в форматі JSON та отримувати їх відповіді.
- Автентифікація та авторизація: ASP.NET Core API надає різноманітні можливості для автентифікації та авторизації користувачів. Вона підтримує різні механізми аутентифікації, такі як JWT (JSON Web Tokens), OAuth та інші.
- Інструментарій для тестування: ASP.NET Core API має вбудований інструментарій для тестування, що спрощує написання та виконання автоматизованих тестів для ваших API.

ASP.NET Core API також надає широкий спектр додаткових можливостей, таких як маршрутизація, логування, фільтри дій та багато іншого. Вона є частиною екосистеми ASP.NET Core, що дозволяє легко інтегрувати API з іншими компонентами, такими як ASP.NET Core MVC та SignalR.

Загалом, ASP.NET Core API є потужним і зручним фреймворком для розробки веб-додатків та API. Вона надає розробникам багатий набір інструментів та можливостей, що допомагають створювати сучасні та ефективні додатки

3.14) Entity Framework Core (EF Core) [] - є однією з найпопулярніших технологій для доступу до баз даних в екосистемі .NET. Це об'єктно-реляційна технологія доступу до даних, яка надає простий і зручний спосіб роботи з базами даних з використанням об'єктно-орієнтованого підходу. EF Core є наступником попередньої версії Entity Framework, але був повністю переписаний для підтримки платформи .NET Core і .NET 5+.

Основні особливості EF Core:

- Крос-платформеність: EF Core підтримує роботу на різних платформах, включаючи Windows, macOS і Linux. Це дозволяє розробникам використовувати технологію на різних операційних системах залежно від їх потреб.
- Підтримка різних провайдерів баз даних: EF Core підтримує багато різних провайдерів баз даних, таких як Microsoft SQL Server, MySQL, PostgreSQL, SQLite і багато інших. Це дає можливість працювати з різними типами баз даних, зберігаючи при цьому єдиний об'єктно-орієнтований інтерфейс.
- Міграції баз даних [37] : EF Core надає можливість використовувати міграції для оновлення схеми бази даних. За допомогою міграцій можна створювати, оновлювати або видаляти таблиці, колонки,

обмеження та інші об'єкти бази даних. Це спрощує процес розвитку програмного забезпечення, особливо при зміні моделі даних.

- Підтримка LINQ [35] : EF Core використовує Language-Integrated Query (LINQ) для формулювання запитів до бази даних. Це дає можливість виразно і зручно отримувати дані з бази, використовуючи розширення мови C#.
- Слабо зв'язана архітектура [36] : EF Core розділяє функціональність доступу до даних від конкретного провайдера бази даних. Це означає, що розробник може легко замінювати провайдера бази даних без зміни основного коду додатку.
- Підтримка асинхронних операцій: EF Core надає підтримку асинхронного програмування, що дозволяє виконувати операції з базою даних асинхронно і не блокувати головний потік виконання програми.

EF Core спрощує процес доступу до даних і дозволяє розробникам зосередитись на бізнес-логіці своїх додатків. Завдяки широкому спектру функціональних можливостей, підтримці крос-платформеності і багатьом провайдерам баз даних, EF Core став популярним вибором для багатьох розробників .NET.

3.15) PostgreSQL [38] - це потужна реляційна база даних з відкритим кодом, яка надає розширені можливості для зберігання та обробки даних. Вона була розроблена з використанням мови програмування C і випущена в 1989 році як проект університету Каліфорнії. З тих пір PostgreSQL став одним з найпопулярніших виборів для баз даних, широко використовуваний у багатьох проектах та компаніях.

Деталі PostgreSql:

- **Архітектура** [42] : PostgreSQL має клієнт-серверну архітектуру, де серверна частина відповідає за зберігання даних та виконання запитів, а клієнтські додатки з'єднуються з сервером для доступу до даних. PostgreSQL підтримує одночасну роботу багатьох клієнтів та виконання запитів у різних процесах.
- **Мови програмування та інтерфейси**: PostgreSQL підтримує багато мов програмування, включаючи SQL, PL/pgSQL, Python, Java, C/C++, PHP та багато інших. Він також надає різноманітні інтерфейси для взаємодії з базою даних, такі як командний рядок, GUI-інструменти, бібліотеки API для різних мов програмування тощо.
- **Функціональні можливості**: PostgreSQL має багатий набір функціональних можливостей, які роблять його потужним інструментом для роботи з даними. Він підтримує стандартні операції реляційної бази даних, такі як створення, читання, оновлення та видалення даних. Крім того, PostgreSQL надає розширені можливості, такі як тригери, операції з транзакціями, оптимізація запитів, реплікація даних, географічні розширення та багато інших.
- **Безпека** [39]: PostgreSQL забезпечує різні механізми безпеки для захисту даних. Він підтримує автентифікацію на рівні ролей і дозволяє задавати рівні доступу до об'єктів бази даних. Також є можливість шифрування даних для зберігання конфіденційної інформації.
- **Розширення** [41]: PostgreSQL має гнучку систему розширень, яка дозволяє розробникам створювати власні функції, типи даних та індекси. Це дає можливість адаптувати PostgreSQL до конкретних потреб проекту та розширювати його функціональність.

- **Спільнота і підтримка** [40] : PostgreSQL має активну спільноту користувачів та розробників, яка надає підтримку та допомогу. Існує багато онлайн-ресурсів, форумів та документації, які можна використовувати для вивчення та вирішення проблем.
- **Висока продуктивність**: PostgreSQL володіє ефективною оптимізацією запитів, яка дозволяє виконувати запити швидко, навіть для великих обсягів даних. Він підтримує використання індексів, кешування, паралельну обробку запитів та інші техніки для досягнення високої продуктивності.

3.16) MediatR [43] - є бібліотекою для підтримки патерна Mediator (посередник) в програмуванні на платформі .NET. Вона надає спосіб спрощення комунікації між класами та додатками шляхом використання подій або запитів.

Основна ідея MediatR полягає у тому, щоб розділити логіку взаємодії між різними компонентами програми, зокрема запитами і запитовими обробниками. Це дозволяє збільшити модульність, зменшити залежність і підтримувати принцип єдиної відповідальності (Single Responsibility Principle).

Основні поняття в MediatR:

- Запит (Request): Це об'єкт, який представляє запит, який треба виконати. Він може містити необхідні дані для виконання операції.
- Запитовий обробник (Request Handler): Це клас, який виконує логіку обробки запиту. Він має метод, який приймає запит і повертає результат.
- Подія (Notification): Це об'єкт, який представляє повідомлення про певну подію, яка сталася в програмі.

- Обробник подій (Notification Handler): Це клас, який виконує логіку обробки події. Він має метод, який приймає повідомлення і виконує відповідні дії.
- Медіатор (Mediator): Це клас, який виступає в ролі посередника між запитами та обробниками. Він приймає запити або події та передає їх відповідним обробникам для виконання.

Основна перевага використання MediatR полягає в тому, що вона дозволяє розібивати логіку програми на невеликі, розсіяні по різним класам модулі. Це полегшує тестування, підтримку та розширення коду. Крім того, це забезпечує слабку залежність між класами, що сприяє покращенню розширюваності і підтримує принцип інверсії залежностей (Dependency Inversion Principle).

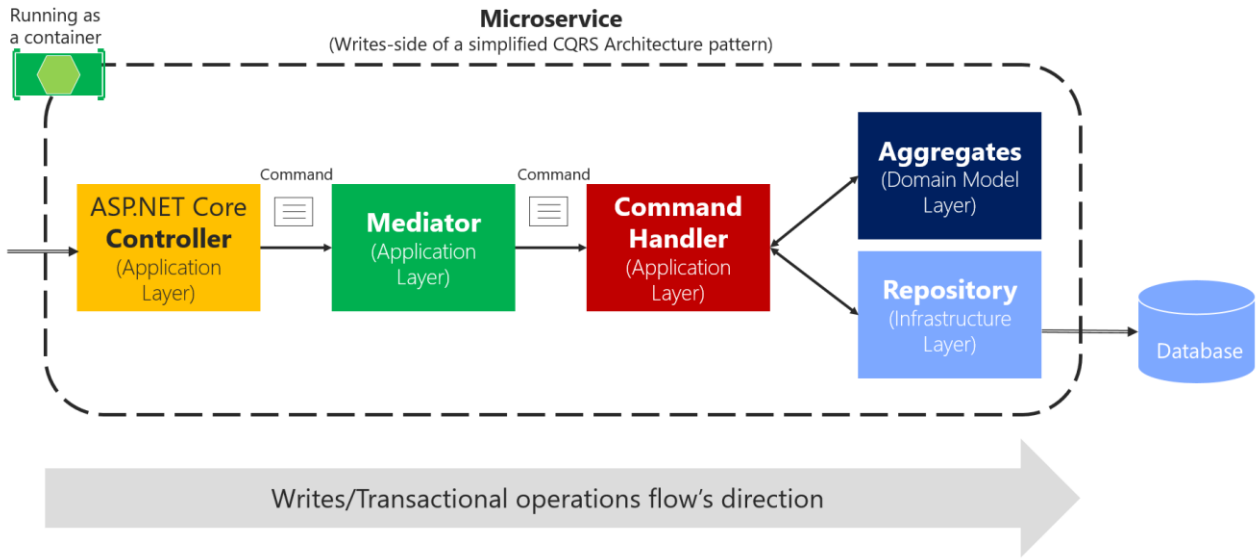


Рис. 8: Схема роботи Mediatr

3.17) FluentValidation [44] - є технологією валідації даних для .NET-програм. Вона дозволяє зручно та ефективно валідувати дані, що надходять до програми, забезпечуючи високу надійність та якість введених даних. FluentValidation надає зручний та простий спосіб визначення правил валідації та повідомлень про помилки.

Основні переваги FluentValidation:

- Чистий та лаконічний синтаксис: FluentValidation пропонує простий та лаконічний синтаксис для визначення правил валідації. Замість написання складних умов чи користування атрибутів валідації, можна використовувати ланцюжок методів для опису правил.
- Розширюваність: FluentValidation дозволяє розширювати його функціональність, додавати власні правила валідації та повідомлення про помилки. Це дозволяє адаптувати технологію під конкретні потреби проекту.
- Валідація складних об'єктів: FluentValidation підтримує валідацію складних об'єктів та глибоке вкладення. Можна визначити правила валідації для вкладених об'єктів, колекцій та ієрархічних структур даних.
- Локалізація повідомлень про помилки [45] : FluentValidation дозволяє легко налаштувати локалізацію повідомлень про помилки. Можна визначити різні мови та мовні ресурси для виведення зрозумілих повідомлень користувачам.
- Підтримка тестування: FluentValidation надає зручні можливості для тестування валідації даних. Можна легко написати автоматизовані тести для перевірки правильності валідації та повідомлень про помилки.

3.18) Authentication JwtBearer - це компонент, що надає підтримку аутентифікації та авторизації з використанням JSON Web Tokens (JWT) у додатках, побудованих на платформі ASP.NET Core. JWT є стандартом для безпечної передачі інформації між двома сторонами у вигляді JSON-об'єктів.

Технологія JWTBearer у ASP.NET Core дозволяє перевіряти та верифікувати JWT-токени для аутентифікації користувачів. Це важлива

складова системи безпеки, оскільки дозволяє забезпечити, що тільки дійсні та валідні токени можуть отримати доступ до захищених ресурсів.

Основні компоненти Microsoft.AspNetCore.Authentication.JwtBearer включаються у бібліотеку Microsoft.AspNetCore.Authentication.JwtBearer. Ці компоненти забезпечують підтримку розбору та верифікації JWT-токенів, валідацію підпису та перевірку прав доступу.

Основні параметри включають наступне:

- Issuer - вказує видавця токенів (Issuer), який генерує JWT-токени.
- Audience - визначає отримувача токенів (Audience), який може використовувати токени для доступу до ресурсів.
- Authority - вказує URL-адресу, яка перевіряє валідність JWT-токенів.

4. Структура проектів

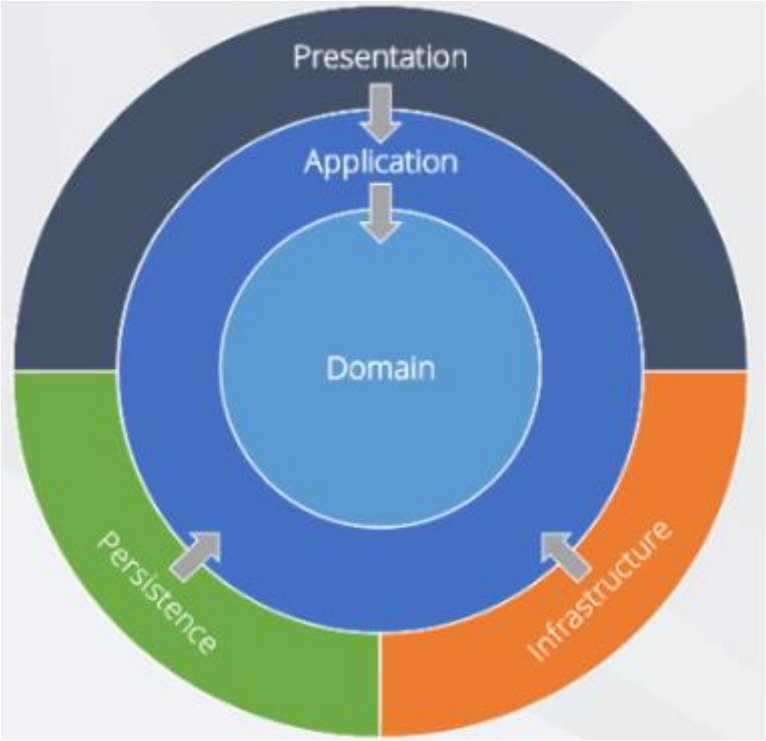


Рис. 9: Бекенд архітектура.

Clean Architecture (Чиста архітектура) [47] - це архітектурний підхід до розробки програмного забезпечення, який забезпечує високу ступінь розділення відповідальностей, підвищує зручність тестування і підтримку коду, а також полегшує масштабування проекту. Цей підхід був запропонований Робертом С. Мартіном, відомим також як Uncle Bob, і знайшов широке застосування в сучасній програмній інженерії.

Основна ідея Clean Architecture полягає в тому, щоб розділити систему на шари залежності, кожен з яких виконує конкретні функції і має визначені межі. Ці шари побудовані на принципах інверсії залежностей (Dependency Inversion Principle) і відокремлення концернів (Separation of Concerns).

Основні компоненти Clean Architecture включають такі шари:

4.1) Domain Layer (Доменний шар): Це внутрішній шар, який визначає бізнес-логіку і правила додатку. Він повинен бути незалежним від будь-яких інших шарів і не повинен містити жодних залежностей від зовнішніх фреймворків або інфраструктури. Цей шар містить сутності (Entities), використовувачів (Use Cases) та інтерфейси репозиторіїв (Repository Interfaces).

4.2) Use Case Layer (Шар використання): Цей шар містить реалізації використання (Use Case Implementations) або служби додатку, які взаємодіють з доменним шаром. Він визначає, як використовувати доменні сутності для розв'язання конкретних задач. Шар використання не має залежностей від будь-яких інших шарів.

4.3) Interface Adapters (Адаптери інтерфейсів): Цей шар перетворює дані зі зовнішніх джерел (наприклад, бази даних або зовнішніх служб) в формат, зрозумілий для внутрішніх шарів, і наоборот. Він містить реалізації

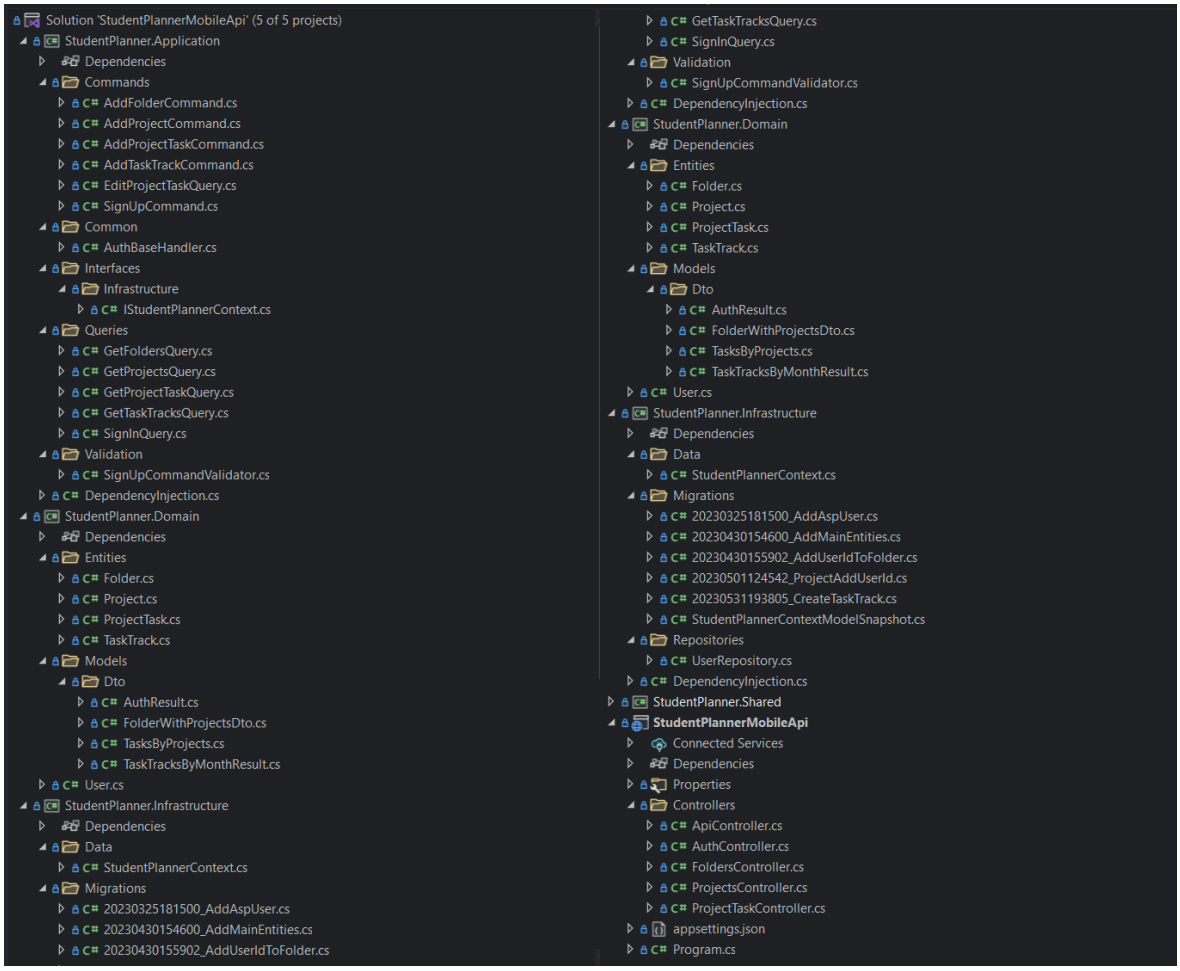
репозиторіїв (Repository Implementations), фреймворків та інших залежностей, які використовуються для взаємодії з зовнішніми системами. Цей шар також включає презентери (Presenters) або контролери, які взаємодіють з інтерфейсом користувача.

4.4) Frameworks and Drivers (Фреймворки та драйвери): Цей зовнішній шар містить фреймворки, бібліотеки або будь-які зовнішні драйвери, такі як фронтенд, бази даних, мережеві служби тощо. Він використовується для взаємодії з зовнішніми системами та передачі даних з інтерфейсу користувача до шару адаптерів інтерфейсу.

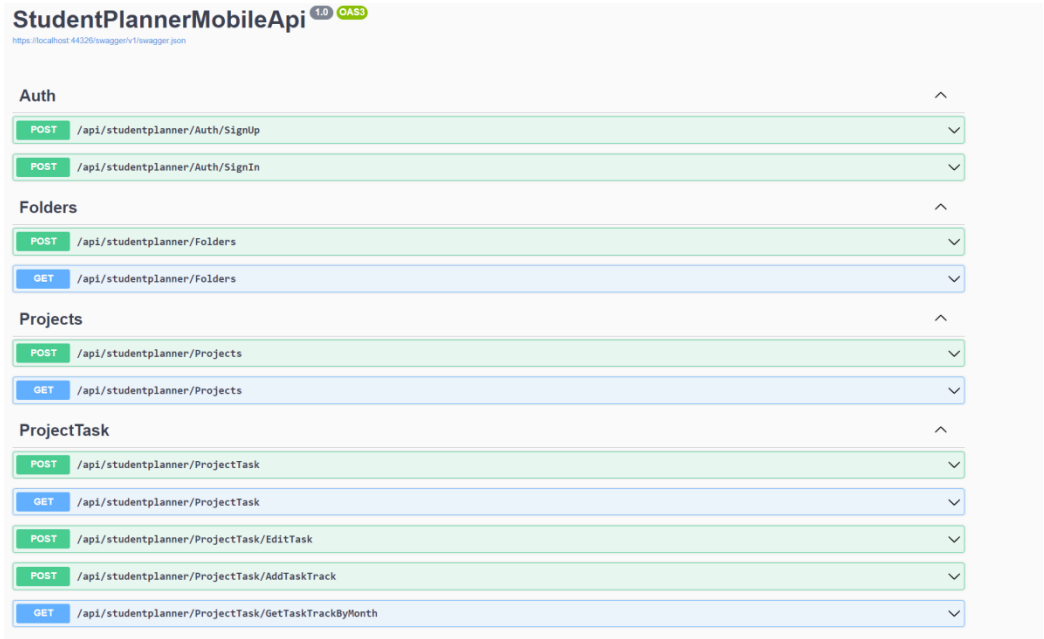
4.5) Переваги Clean Architecture включають:

- **Розділення відповідальностей**: Clean Architecture дозволяє явно визначити межі між компонентами системи і розділити відповідальності. Це полегшує розуміння, тестування та підтримку коду.
- **Зручне тестування**: Завдяки відокремленню бізнес-логіки в доменному шарі, тести можуть бути написані без залежностей від зовнішніх фреймворків або інфраструктури. Це полегшує автоматизоване тестування і підвищує його стабільність.
- **Легка масштабовність**: Шарова структура дозволяє легко замінювати або розширювати компоненти системи без впливу на інші шари. Це полегшує зміни і розвиток системи з плином часу.

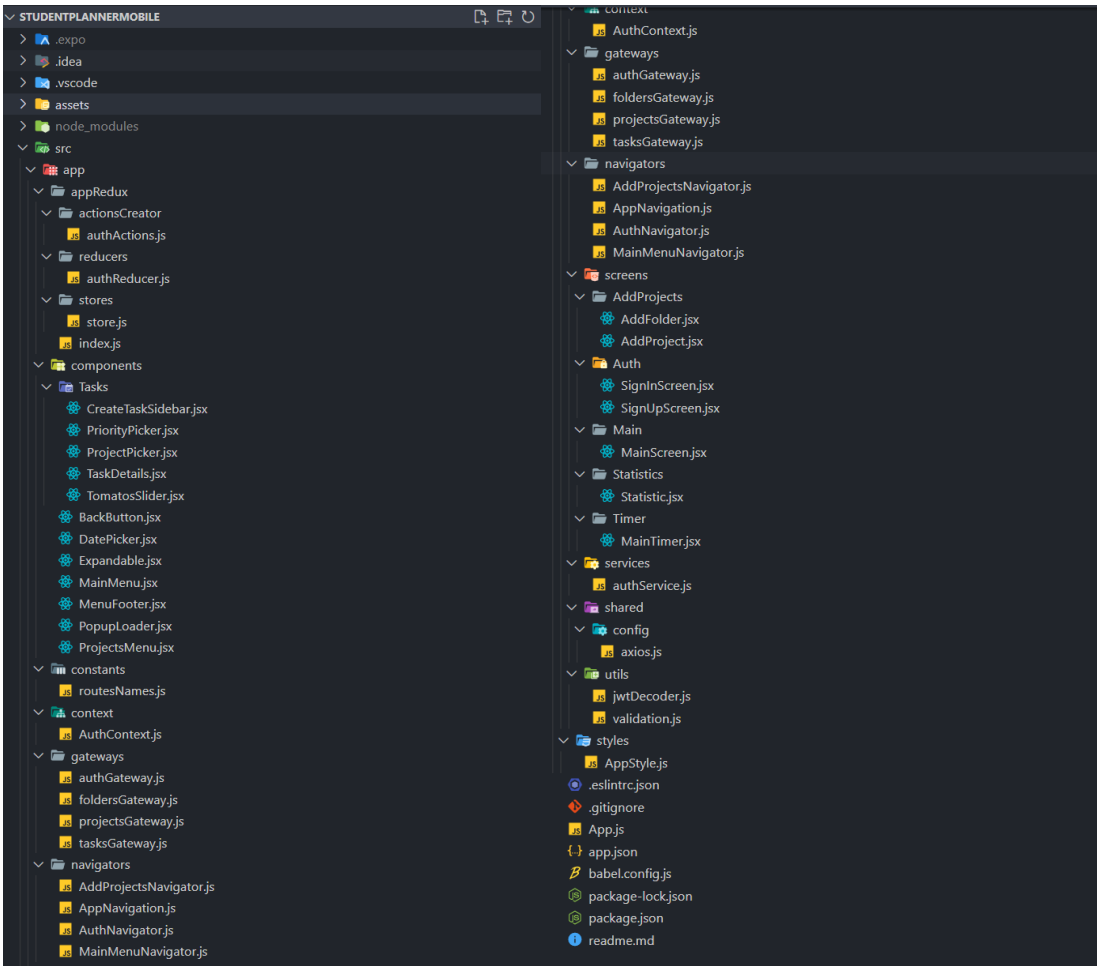
4.6) Структура .NET проекту



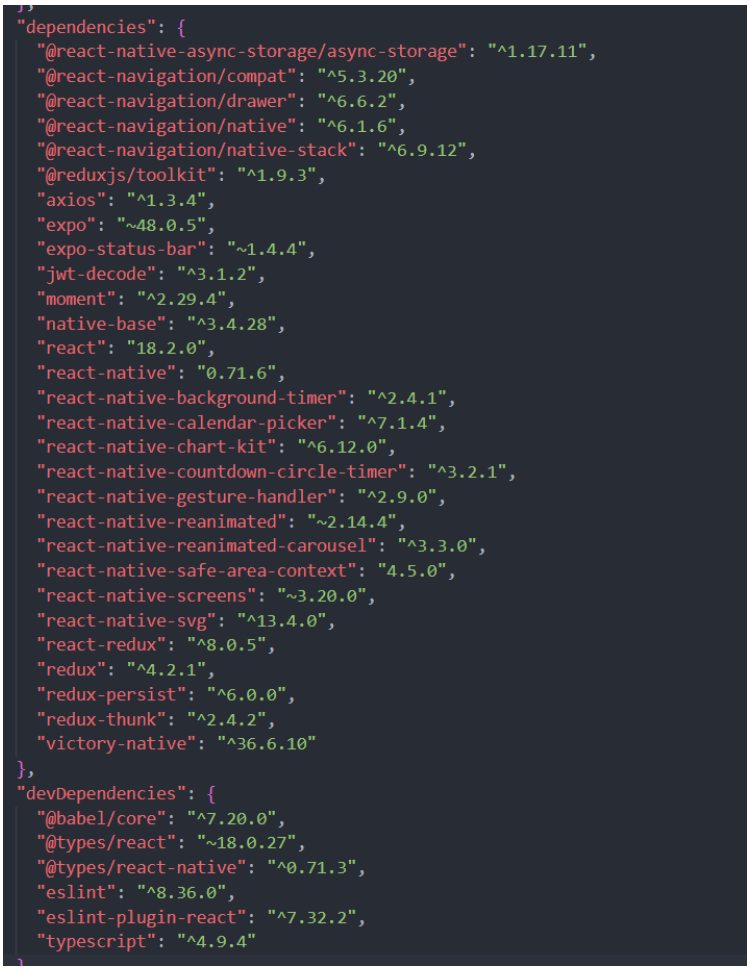
4.7) RESTful документація API в Swager за специфікацією OpenAPI



4.8) Структура React Native проекту

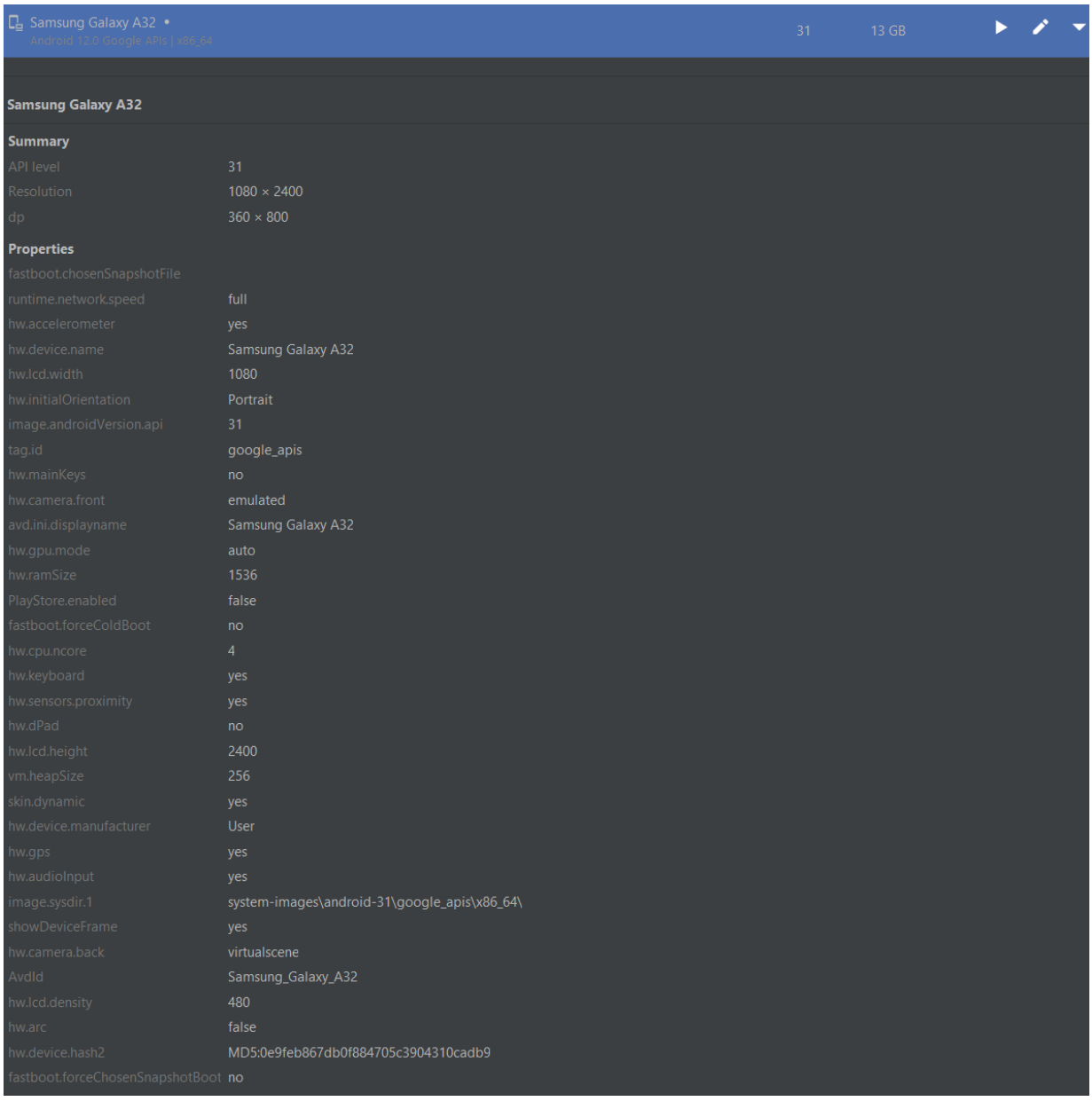


4.9) Список залежностей React Native проекту та їх версії



```
"dependencies": {
  "@react-native-async-storage/async-storage": "^1.17.11",
  "@react-navigation/compat": "^5.3.20",
  "@react-navigation/drawer": "^6.6.2",
  "@react-navigation/native": "^6.1.6",
  "@react-navigation/native-stack": "^6.9.12",
  "@reduxjs/toolkit": "^1.9.3",
  "axios": "^1.3.4",
  "expo": "~48.0.5",
  "expo-status-bar": "~1.4.4",
  "jwt-decode": "^3.1.2",
  "moment": "^2.29.4",
  "native-base": "^3.4.28",
  "react": "18.2.0",
  "react-native": "0.71.6",
  "react-native-background-timer": "^2.4.1",
  "react-native-calendar-picker": "^7.1.4",
  "react-native-chart-kit": "^6.12.0",
  "react-native-countdown-circle-timer": "^3.2.1",
  "react-native-gesture-handler": "^2.9.0",
  "react-native-reanimated": "~2.14.4",
  "react-native-reanimated-carousel": "^3.3.0",
  "react-native-safe-area-context": "4.5.0",
  "react-native-screens": "~3.20.0",
  "react-native-svg": "^13.4.0",
  "react-redux": "^8.0.5",
  "redux": "^4.2.1",
  "redux-persist": "^6.0.0",
  "redux-thunk": "^2.4.2",
  "victory-native": "^36.6.10"
},
"devDependencies": {
  "@babel/core": "^7.20.0",
  "@types/react": "~18.0.27",
  "@types/react-native": "^0.71.3",
  "eslint": "^8.36.0",
  "eslint-plugin-react": "^7.32.2",
  "typescript": "^4.9.4"
```

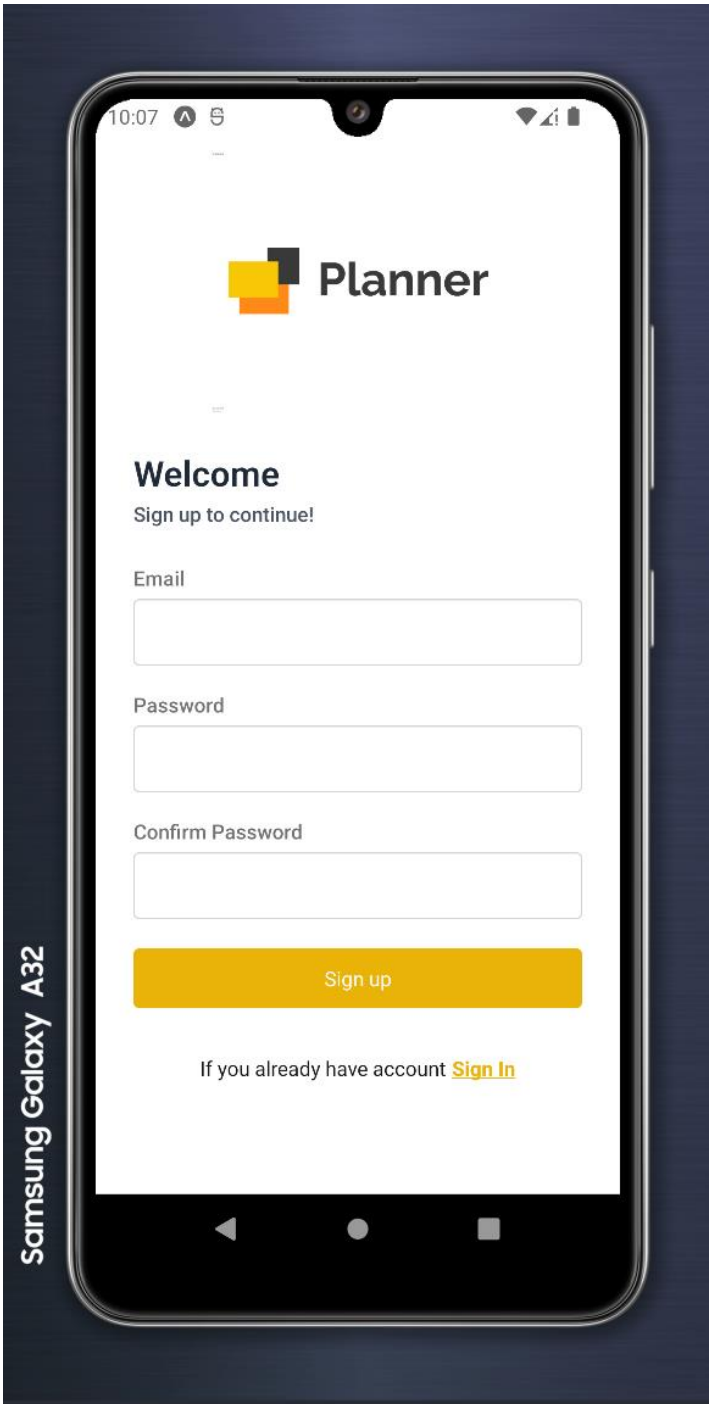
4.11) Конфігурація мобільного пристою в Android Emulator. Мобільний пристрій Samsung Galaxy A32



4.12) Запущений девайс в Android Emulator

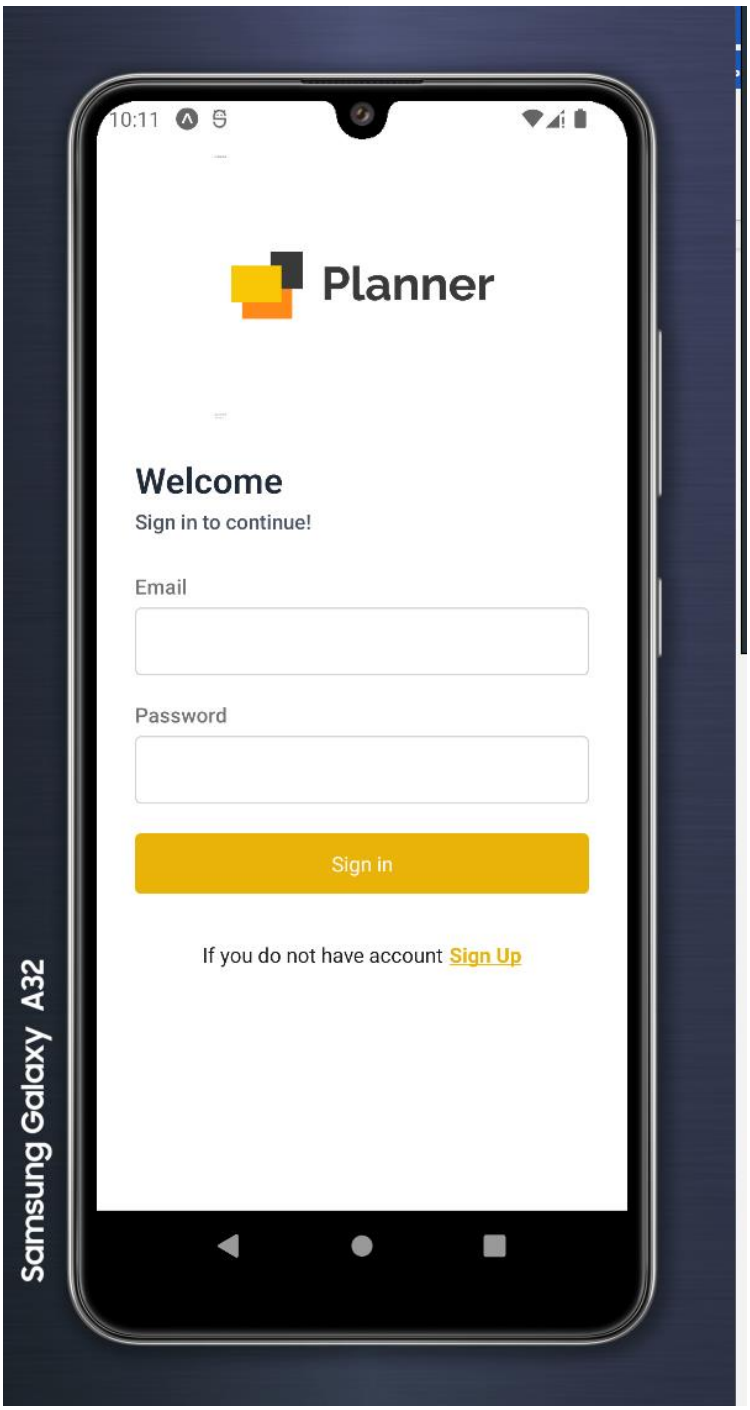
5.Програмна реалізація

5.1) Екран реєстрації



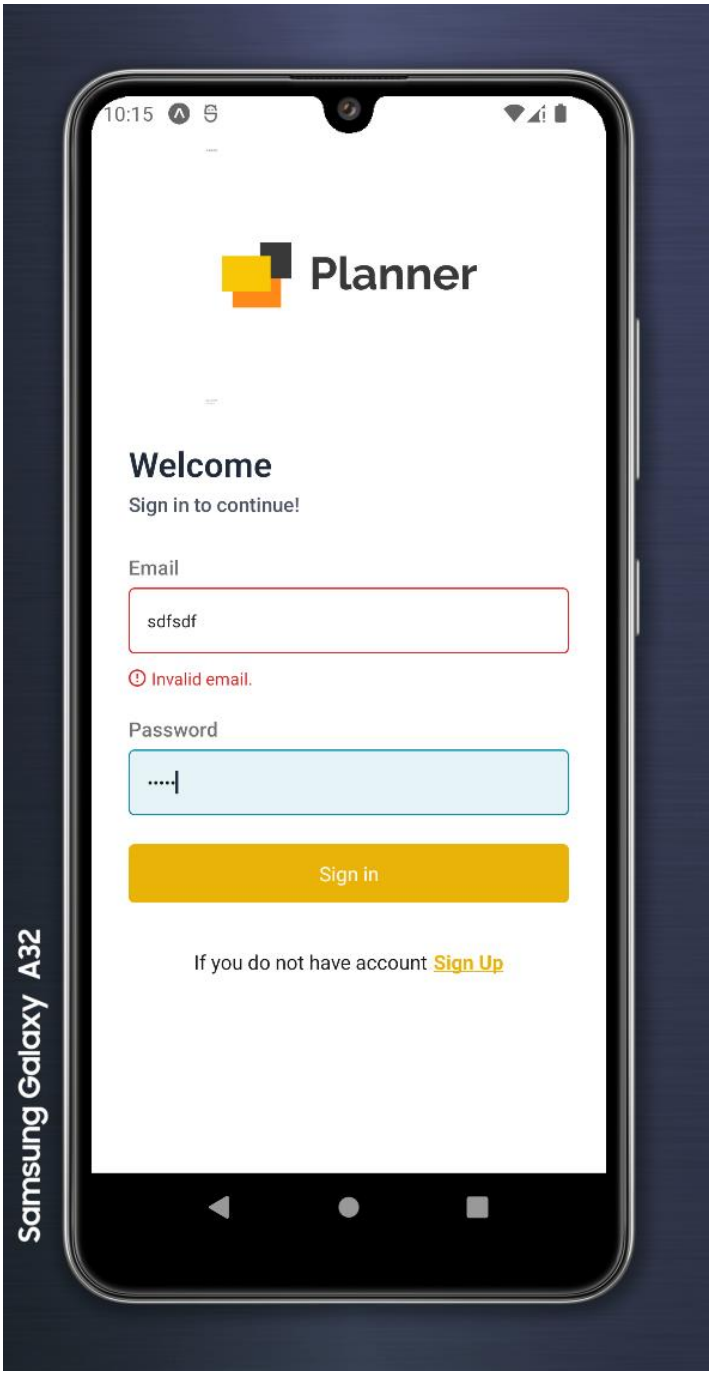
Екран реєстрації є початковим вікном, яке використовується для створення нового облікового запису або реєстрації користувача в системі Користувач починен ввести свою пошту, пароль, та повторно ввести пароль

5.2) Екран авторизації



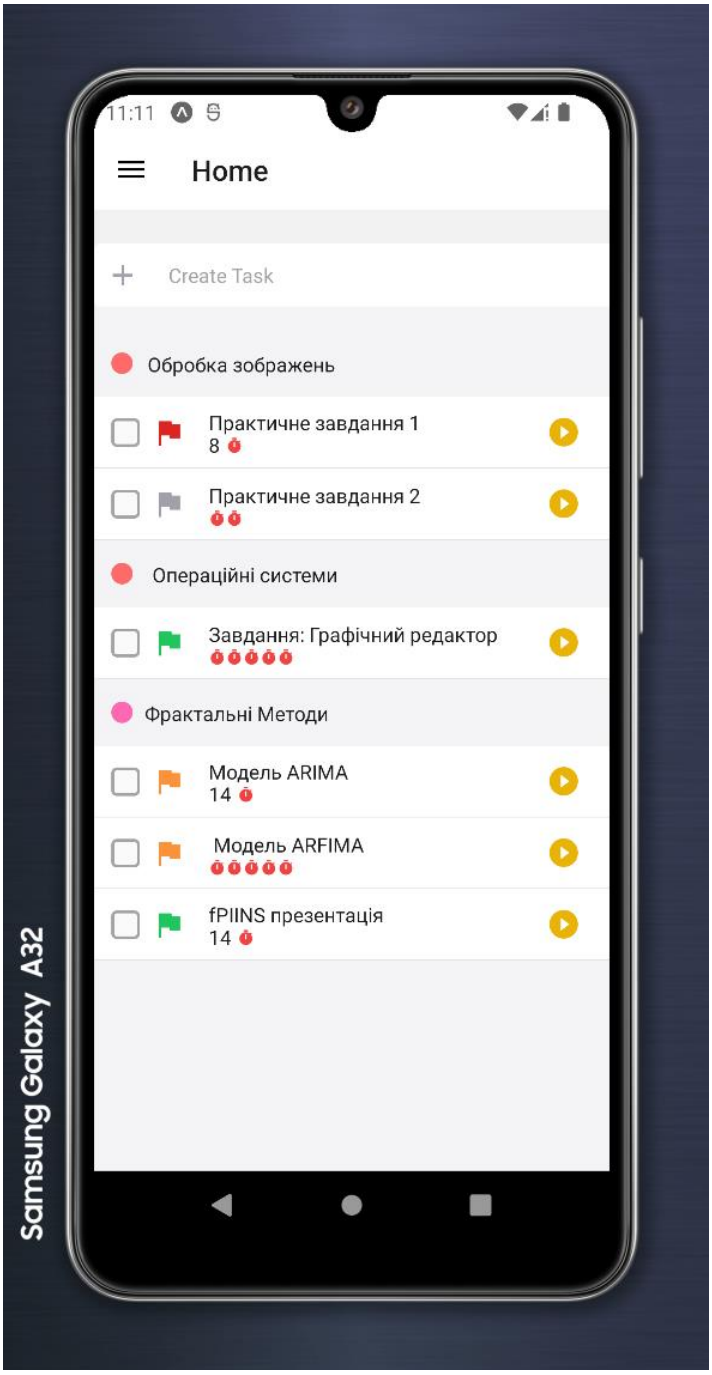
Екран авторизаці теж є початковим вікном, де зареєстрований користувач повинен увійти в обліковий запис

5.3) Екран авторизації валідація форм



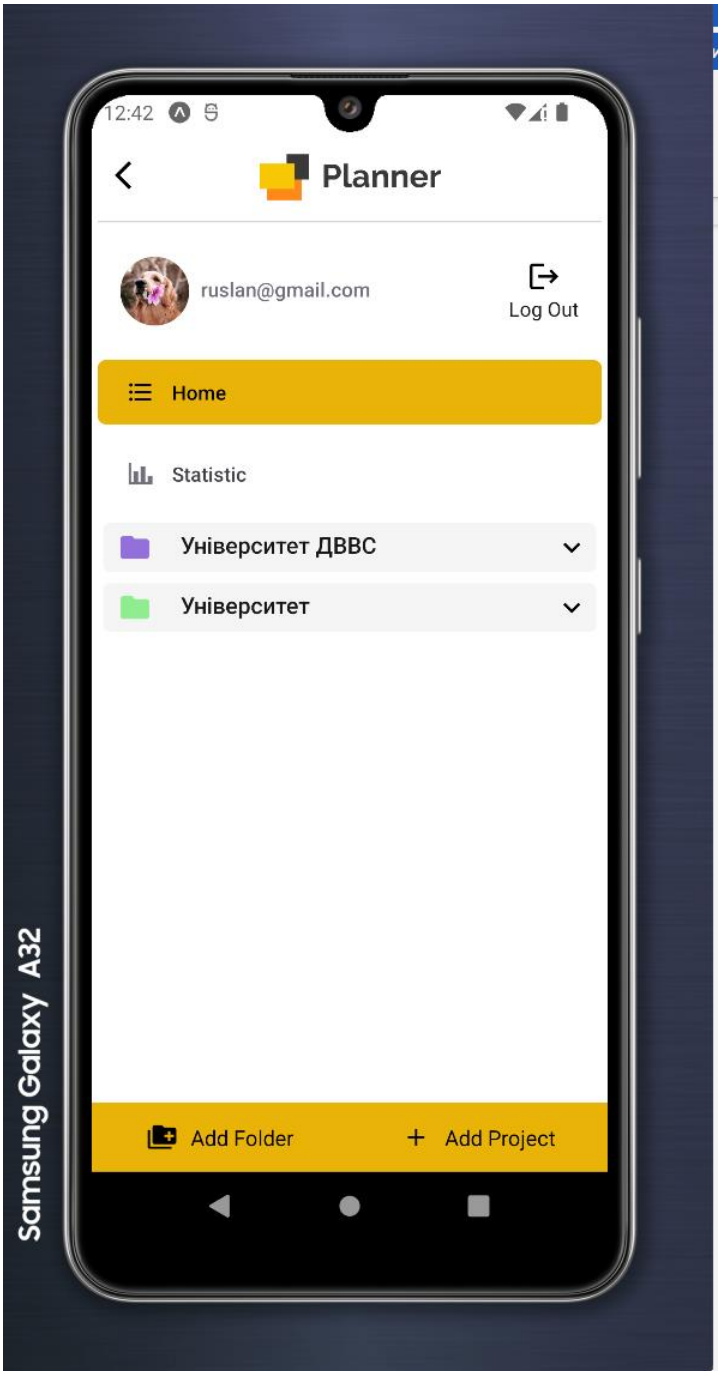
На формах реєстрації та авторизації присутня валідація для безпеки додатку і коректної роботи

5.4) Головний екран з задачами



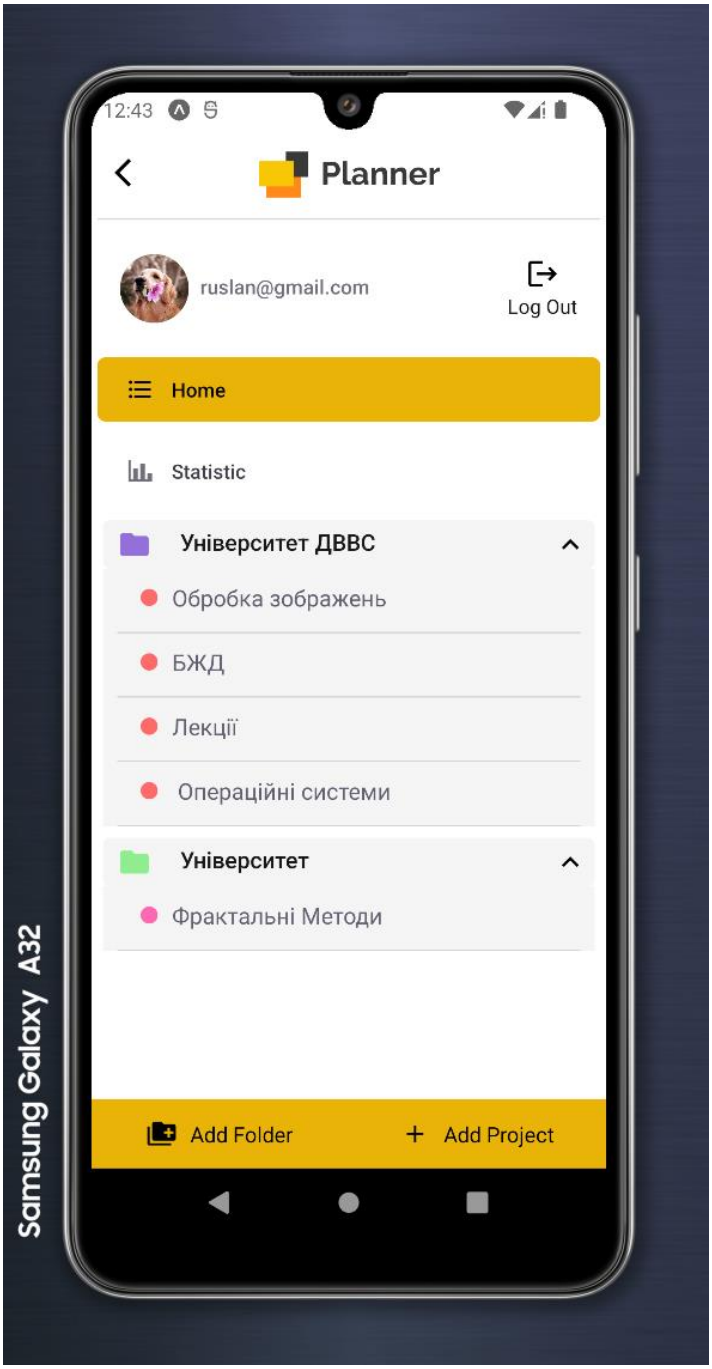
Головний екран містить задачі які створив користувач з різними налаштуваннями такі як: тількість помідорок, приорітет, дату, проект

5.5) Екран меню

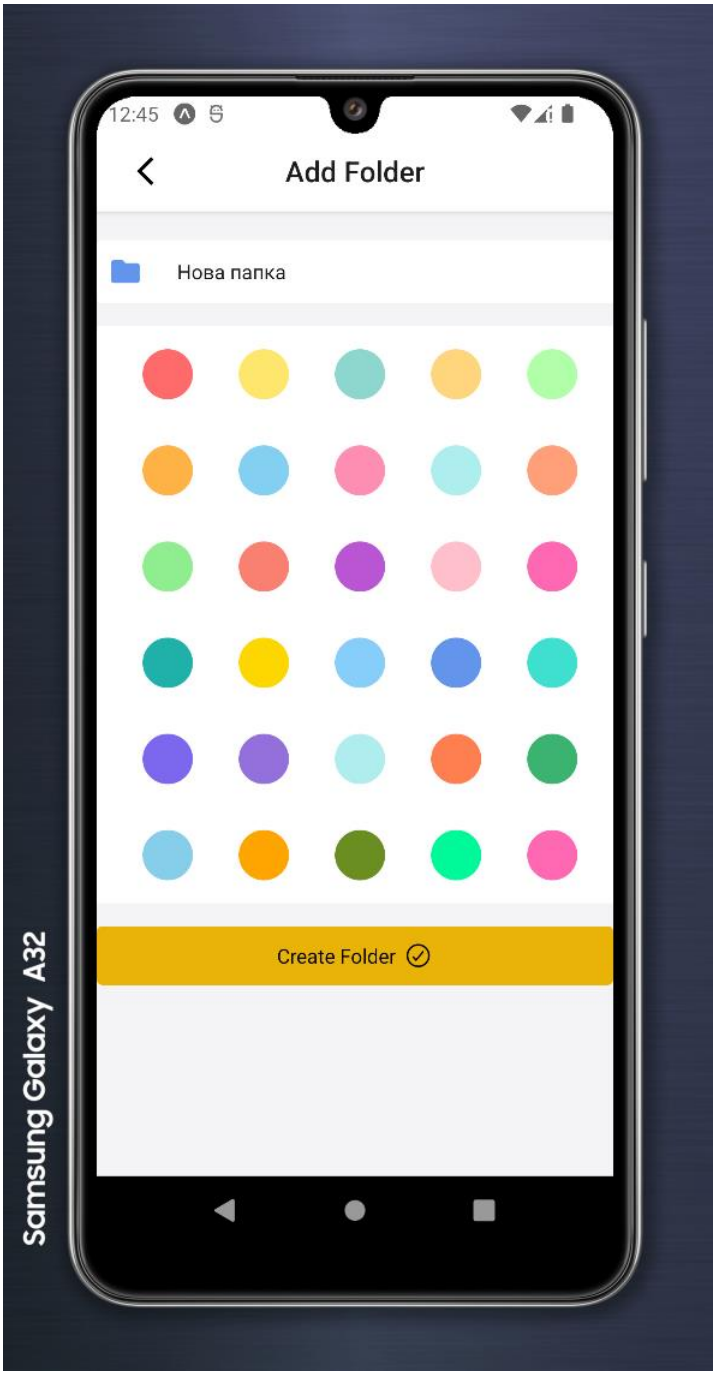


Бокове меню яке пістить навігацію а також створені проекти згруповані по папкам. Є можливість створювати нові папки та проекти

5.6) Екран меню з розгорнутими папками

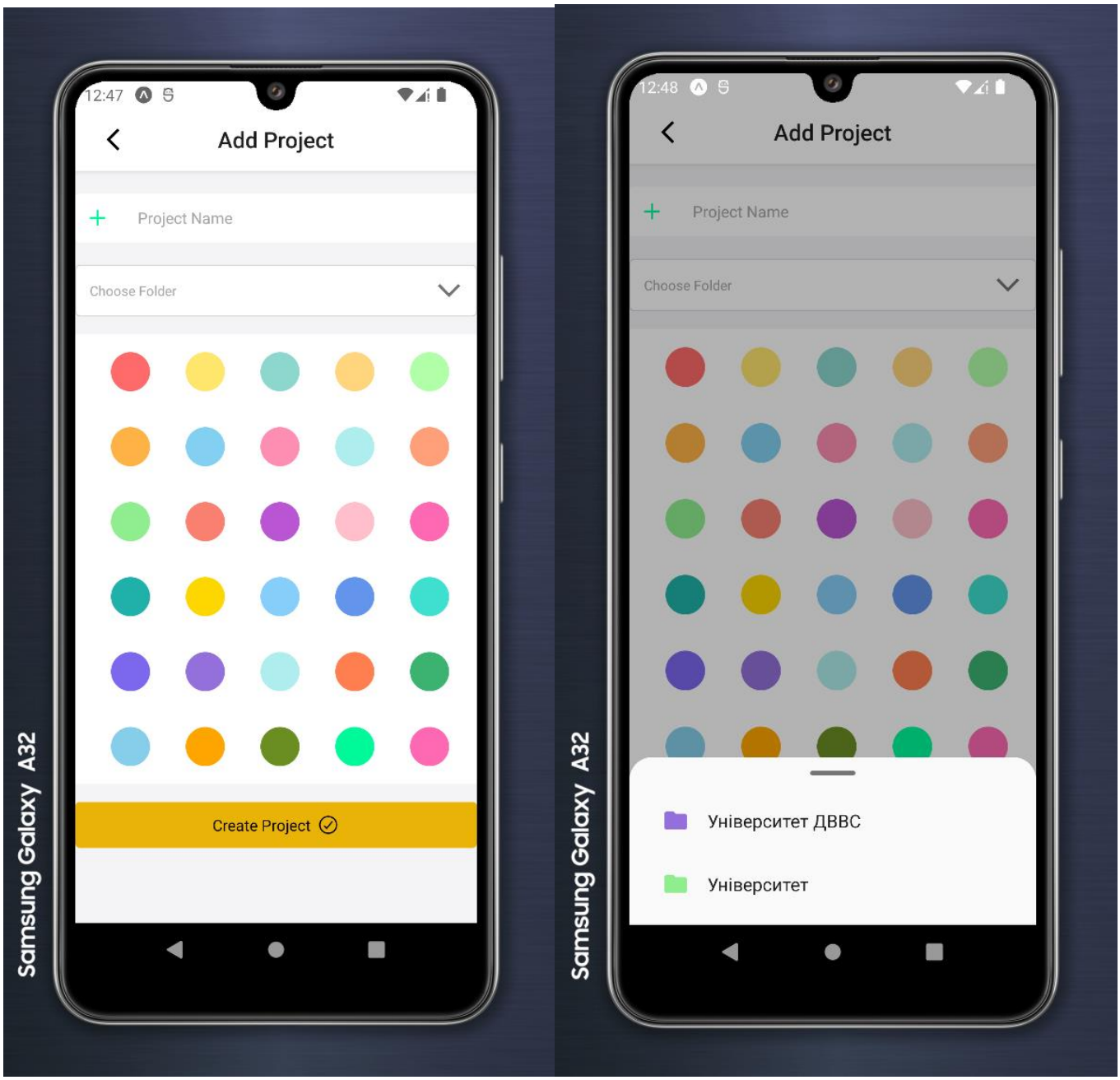


5.7) Створення нової папки



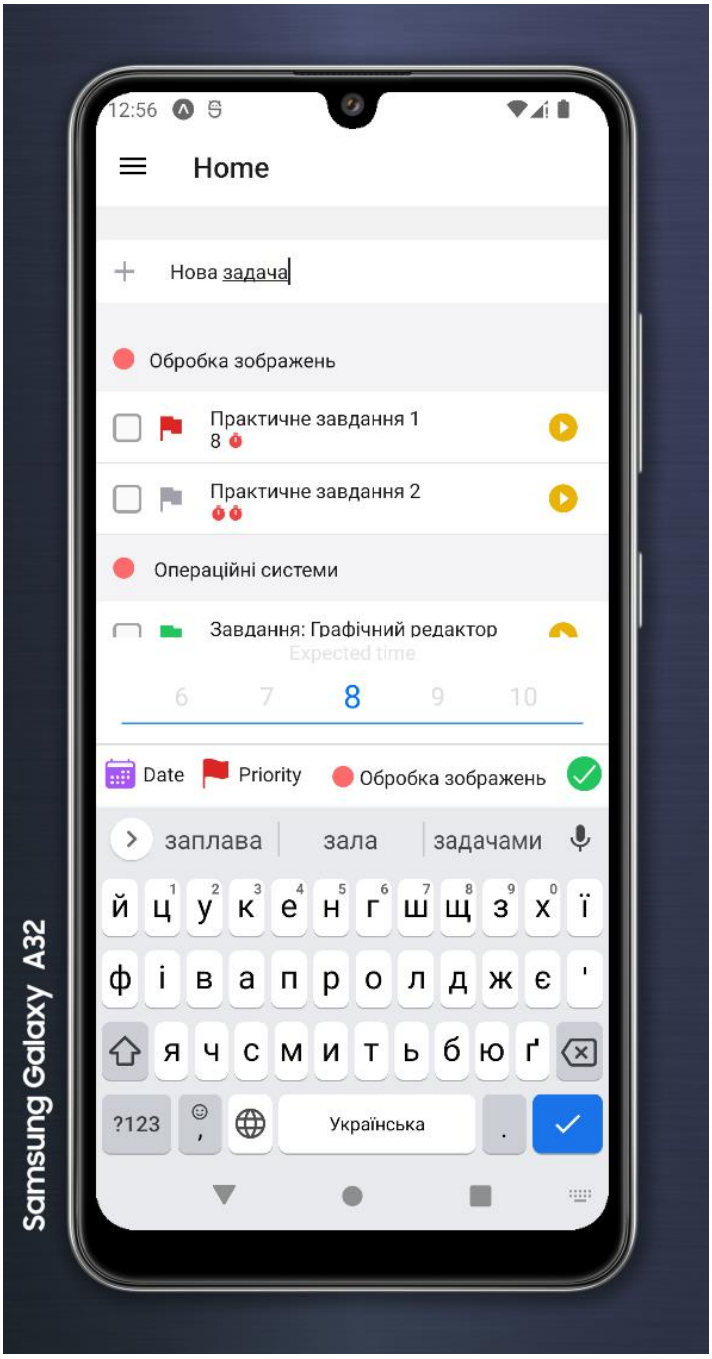
Екран створення нової папки яка потрібна для зручного групування проектів та задач

5.8) Створення нового проекту



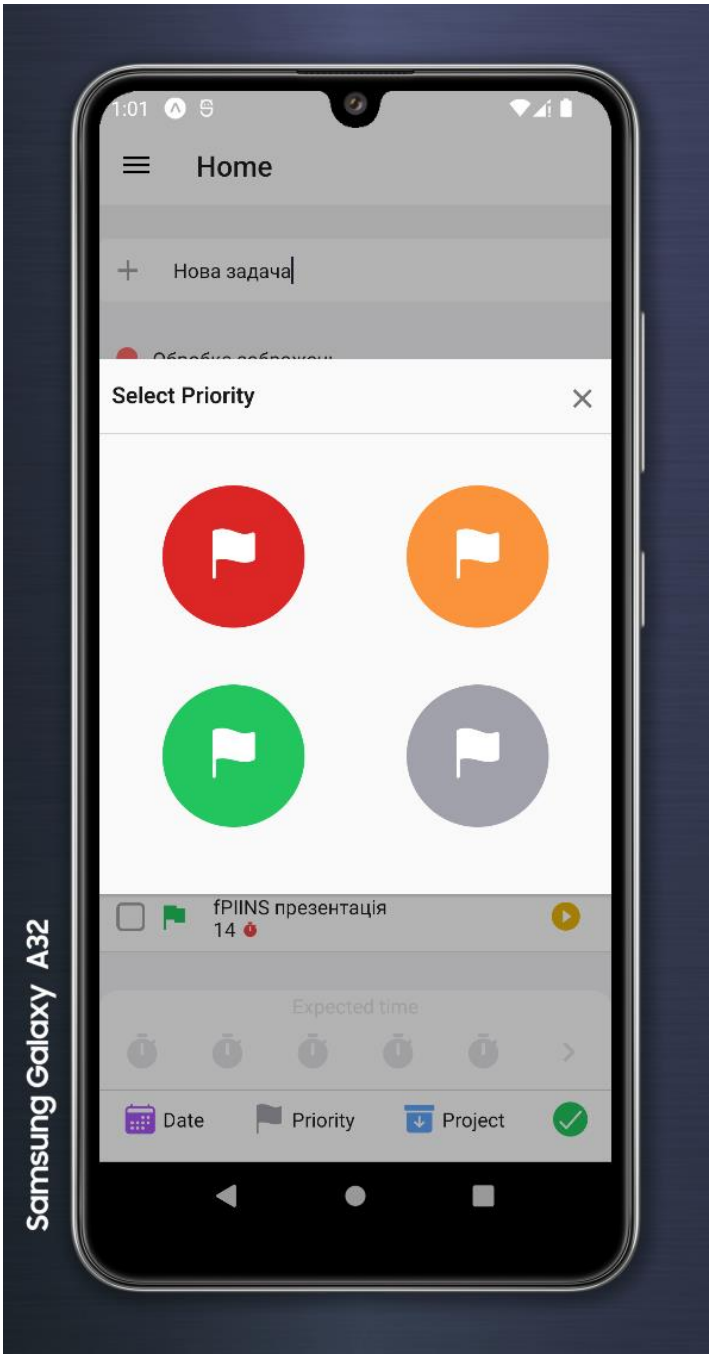
Екран створення проект до якого буду створюватися задачі. На екрані присутні різні кольори для кращого візуального сприйняття а також ці кольори використовуються для побудови діграми у розділі статистики.

5.9) Створення нової задачі



Екран створення задачі який нада можливість обрати кількість томатів які планується потратити на роботу, дату, приорітет та проект

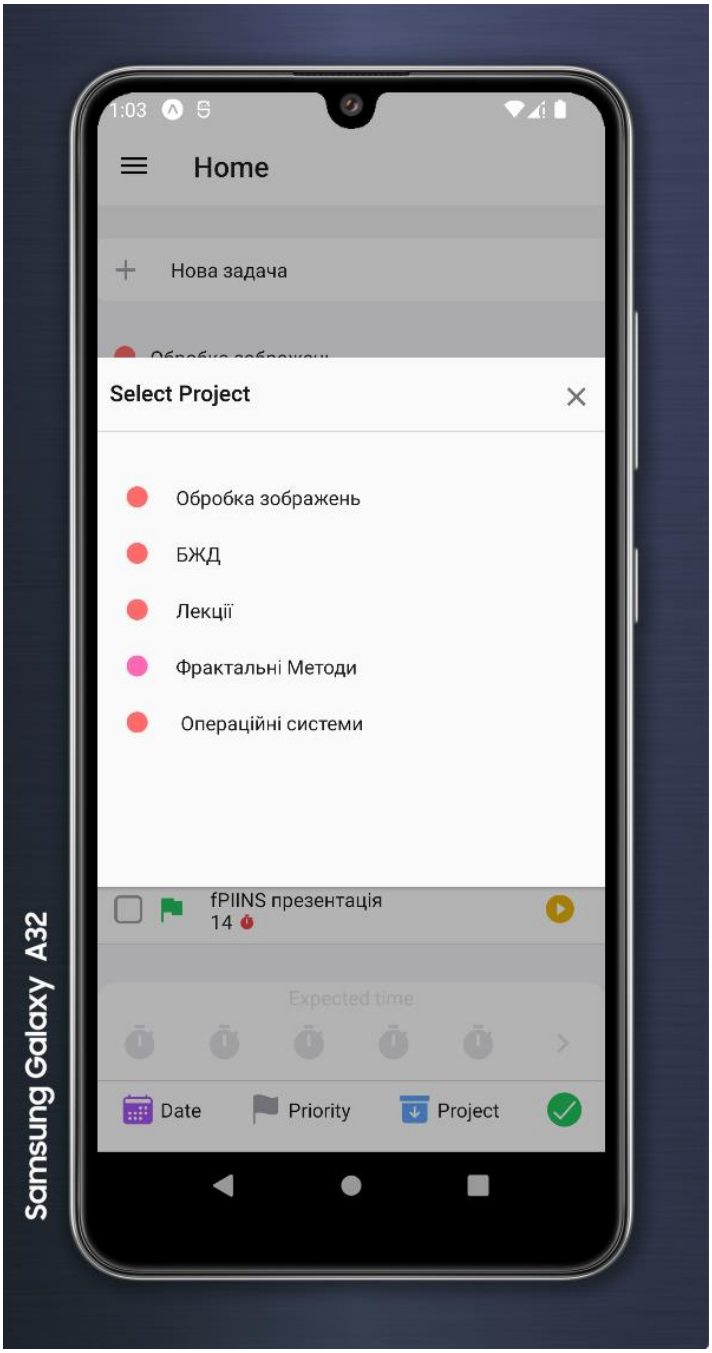
5.10) Вибір приорітету



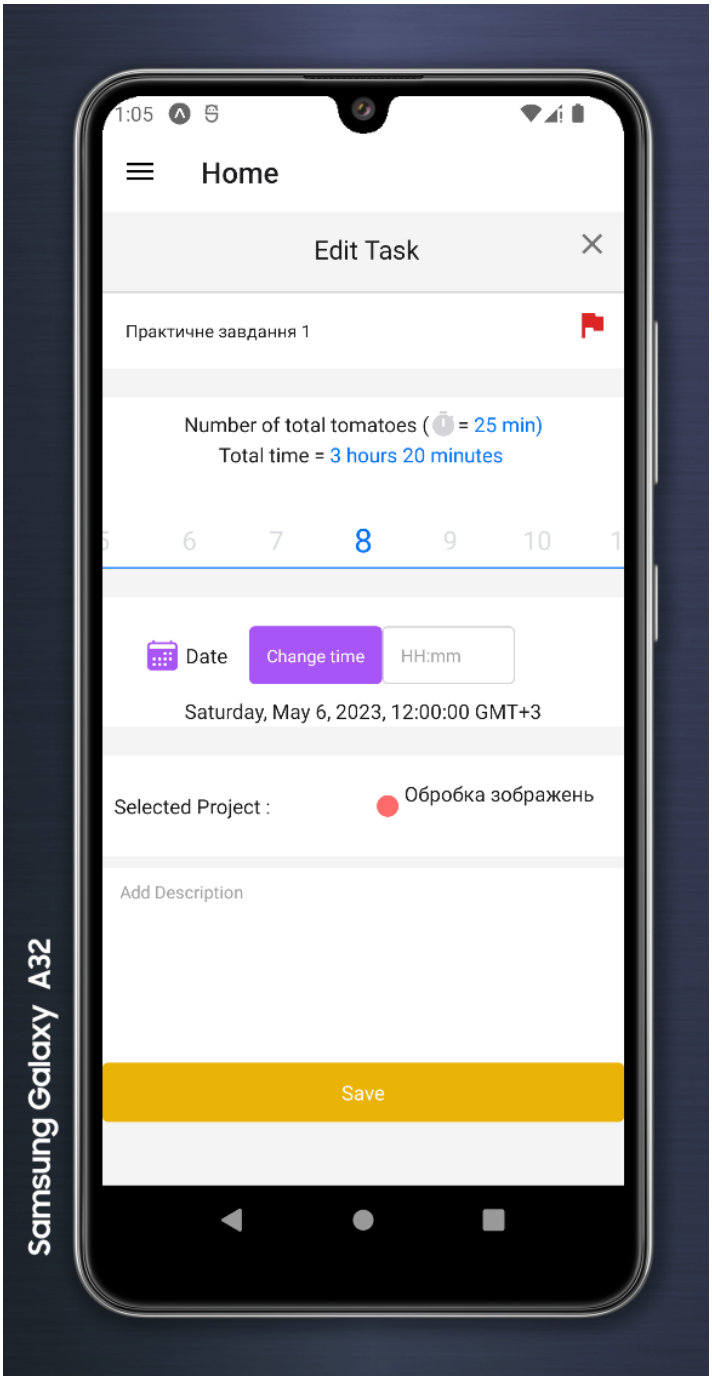
Модальне вікно в якому можна обрати приорітет. Червоний прапорець самий великий приорітет , а сірий самий низький приорітет

5.11) Вибір дати

5.12) Вибір проекту для задачі

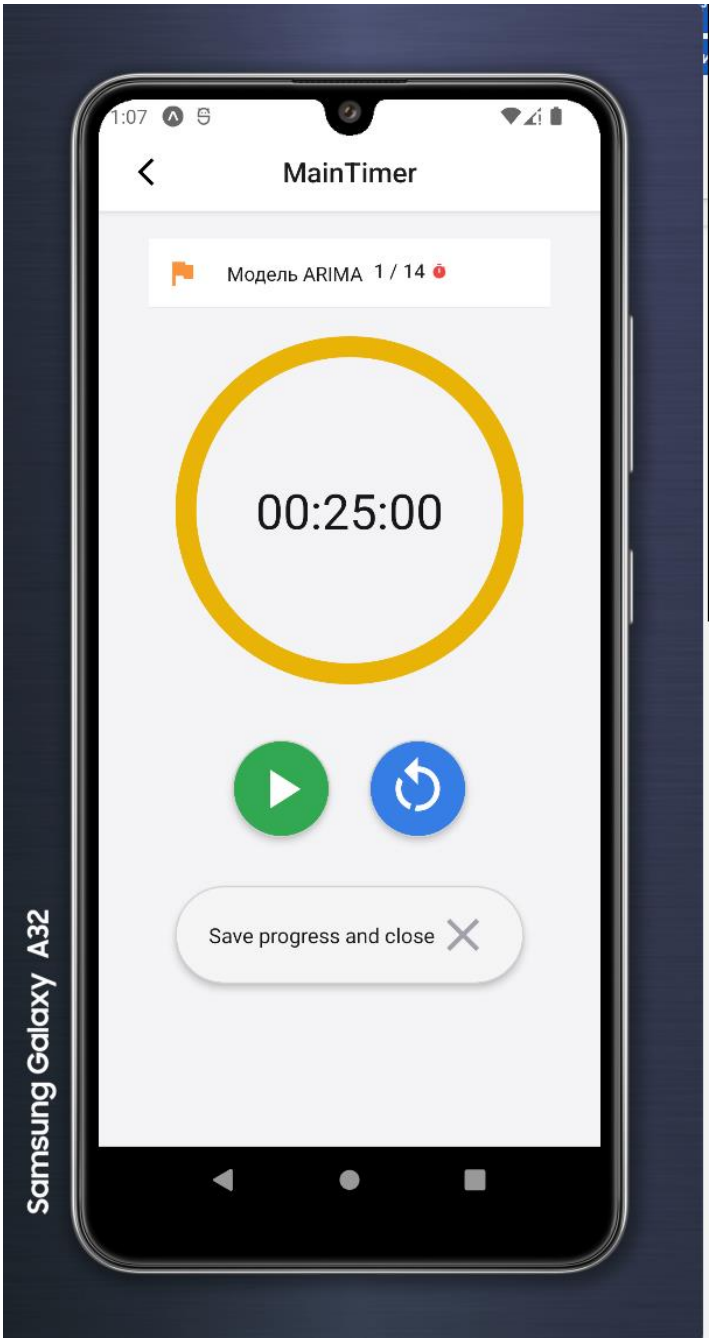


5.13) Редагування задачі

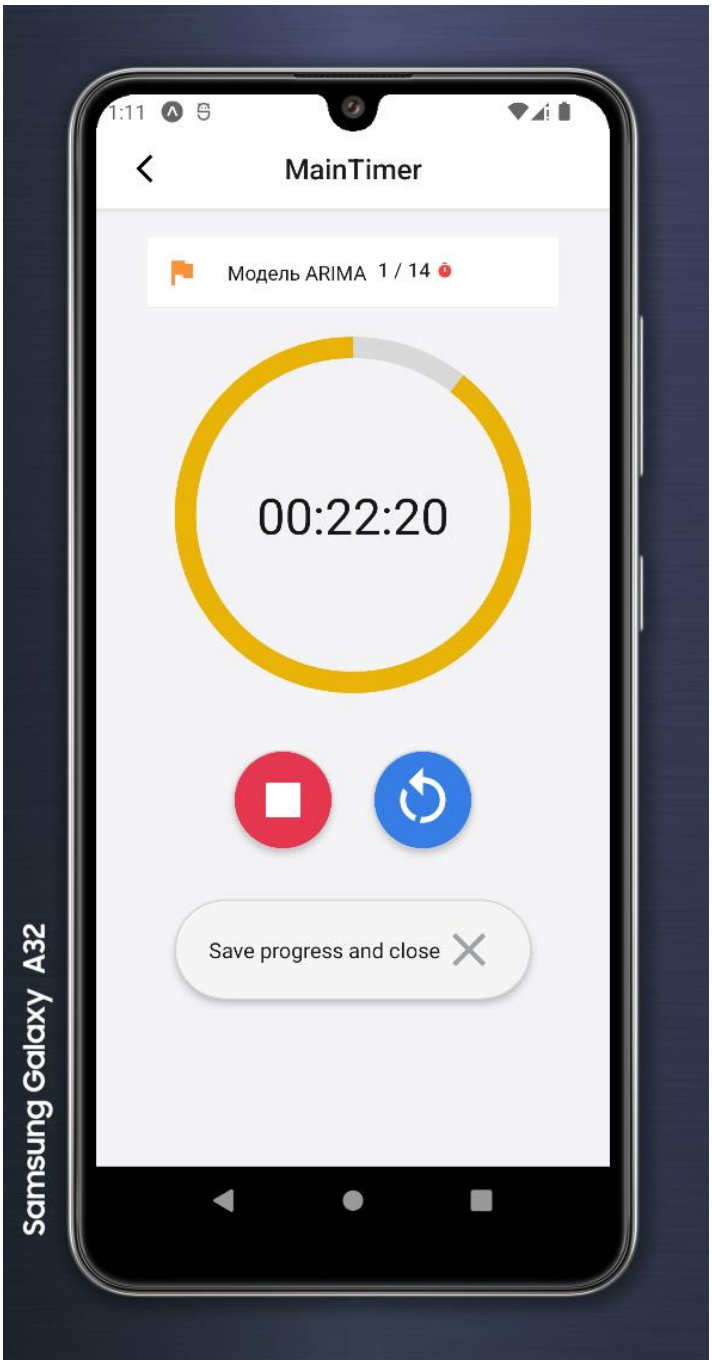


Екран ствредагування задачі надає змогу змінити терміни задачв, час на виконання, проет, опис назву та приорітет

5.14) Запуск таймера



Екран з таймером запускається після того користувач нажме кнопку страт на задачі. На цьому екрані присутні кнопки старт, скидання таймера, скидання таймера зі збереженням прогресу та кнопка паузи.

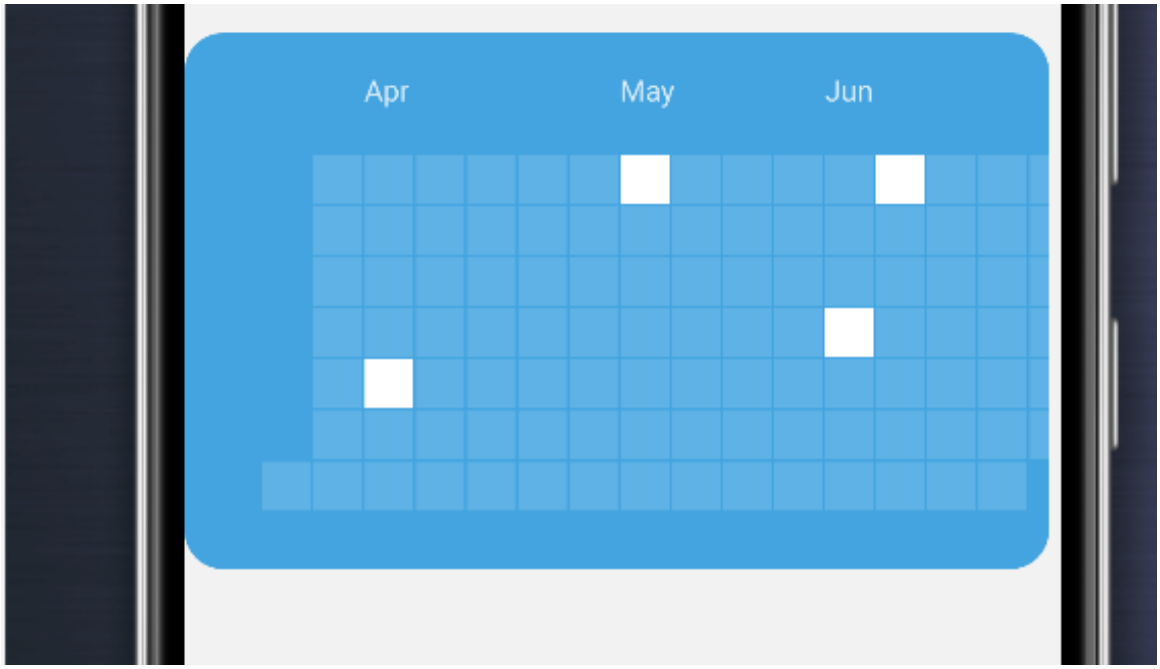


Для візуалізації додано кроговий таймер який зменшується разом з часом

5.15) Графік продуктивності по місяцям

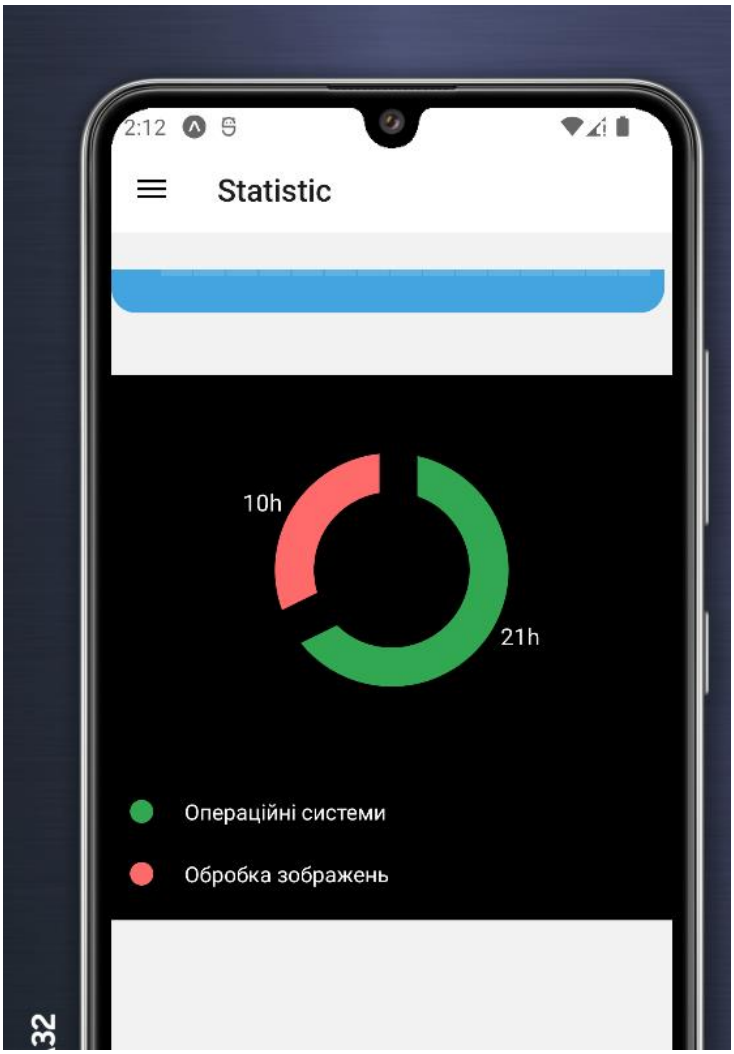
В кладці статистики присутній графік продуктивності по місяцям де відображається загальна кількіст потрачених годин на всі задачі.

5.16) Графік діяльності по дням та місяцям



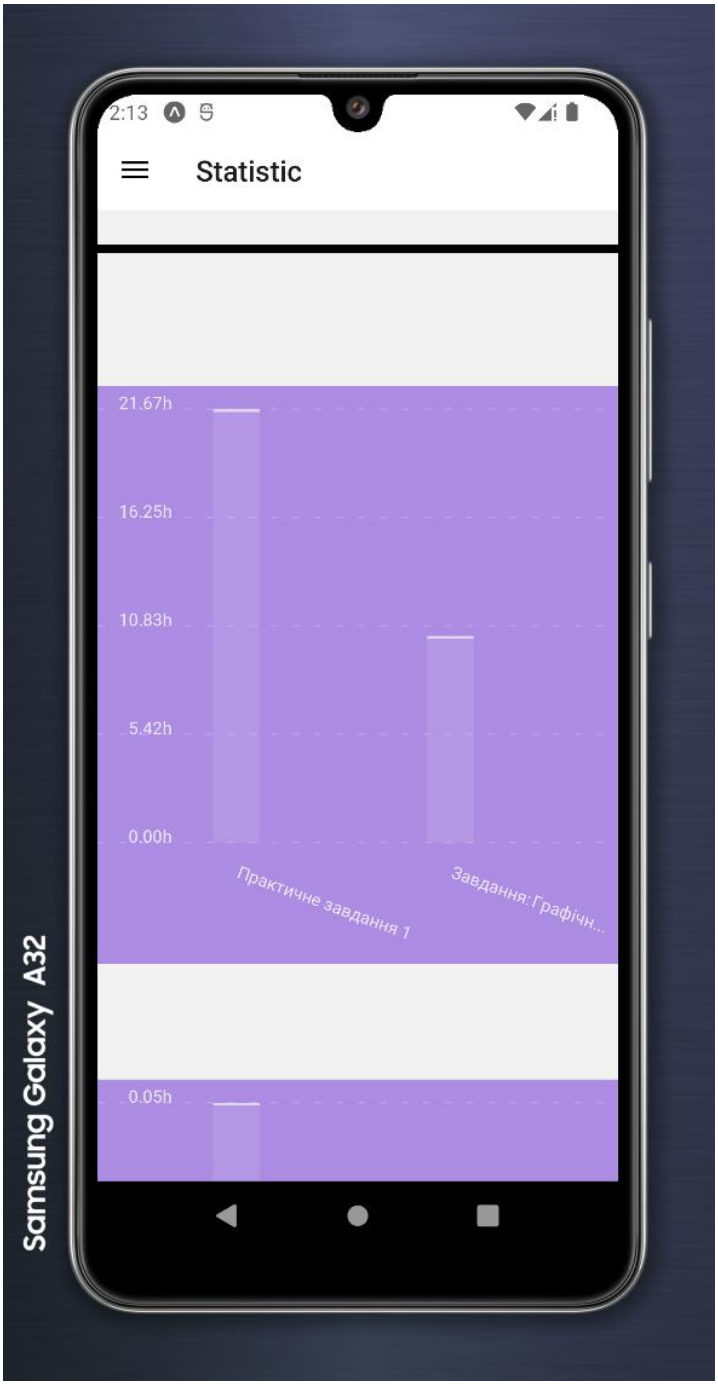
В кладці статистики присутній графік продуктивності по місяцям де відображається активність по дням.

5.17) Графік продуктивності по проектам



В кладці статистики присутній графік продуктивності по проектам де відображаєтсья потрачений час на кожний проект.

5.18) Гістограма продуктивності по задачам



В кладці статистики присутній графік продуктивності по задачам де відобраєається потрачений час на кожну задачу.

Висновки

Я ознайомився з емулятором андроїд пристроїв та успішно навчився конфігурувати їх, вивчивши різноманітні можливості, які ця технологія пропонує для розробки мобільних додатків на платформі React Native. При цьому було детально досліджено та вивчено тонкощі реалізації різних компонентів, що дозволяє створювати ефективні та функціональні рішення.

Крім того, було проведено ознайомлення з різними варіантами розгортки проекту мобільного додатку, щоб мати повну картину про можливість його впровадження та розповсюдження. На основі цього, я також здобув навички користування допоміжними бібліотеками для React Native та вміння керувати залежностями між ними, що є необхідними для ефективного розвитку та підтримки проектів.

Окрім того, були проведені дослідження сучасних підходів до розробки програмного забезпечення та їх архітектур, що дозволило поглибити розуміння процесу розробки та вибір оптимальних рішень. В результаті була спланована послідовність реалізації функціоналу та його дизайну, що стало основою для подальшого створення мобільного додатку.

На основі набутих знань та практичних навичок, я розробив мобільний додаток з архітектурою клієнт-сервер, який відрізняється від існуючих аналогів своєю унікальністю та покращеними можливостями для підвищення продуктивності.

Окрім того, розв'язав ряд проблем, включаючи проблему багатозадачності та розсіяності, шляхом зосередження на конкретних задачах та ефективному плануванні часу та пріоритетів. Також вдалося вирішити проблему автоматичного відстежування роботи над задачами та побудови статистики, що сприяє покращенню контролю та ефективності управління проектом.

Список використаних джерел

1. Nabers David React: Cross-Platform Application Development with React Native: Build 4 real-world apps with React Native: [пер. з англ.] / Наберс Девід React: крос-платформна розробка програм за допомогою React Native: створіть 4 реальні програми за допомогою React Native / Nabers David - Packt Publishing, 2018 – 182 с.
2. Adam Scott JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron 1st Edition: [пер. з англ.] / Адам Скотт JavaScript всюди: Створення крос-платформних програм за допомогою GraphQL, React, React Native і Electron перше видання / Adam Scott - O'Reilly Media, 2020 – 341 с
3. React Native Cookbook — Second Edition by Daniel Ward
4. Nmp js бібліотеки - Режим доступу: https://www.npmjs.com/search?q=react%20native
5. Головна сторінка React - Режим доступу: https://uk.legacy.reactjs.org/
6. Документація React Native - Режим доступу: https://reactnative.dev/docs/getting-started
7. Обмеження React Native - Режим доступу: https://www.ava.codes/posts/react-native-limitations
8. How React Native App Development Works Under the Hood - Режим доступу: https://mobidev.biz/blog/how-react-native-app-development-works
9. How React Native App Development Works Under the Hood - Режим доступу: https://devopedia.org/react-native
10. Головна сторінка Expo - Режим доступу: https://docs.expo.dev/
11. Expo SDK - Режим доступу: https://docs.expo.dev/workflow/upgrading-expo-sdk-walkthrough/

12. Expo Client - Режим доступу: https://docs.expo.dev/technical-specs/expo-updates-1/#client
13. Expo Snack - Режим доступу:https://docs.expo.dev/workflow/snack/
14. Expo OTA - Режим доступу:https://blog.expo.dev/over-the-air-updates-from-expo-are-now-even-easier-to-use-376e2213fabf
15. Головна сторінка Redux - Режим доступу: https://redux.js.org/
16. React Native Stack Navigator - Режим доступу: https://reactnavigation.org/docs/stack-navigator/
17. React Native Drawer Navigator - Режим доступу: https://reactnavigation.org/docs/drawer-based-navigation/
18. React Native Tab Navigator - Режим доступу: https://reactnavigation.org/docs/bottom-tab-navigator/
19. Axios Promises - Режим доступу: https://javascript.info/promise-basics
20. Axios interceptors - Режим доступу: https://axios-http.com/docs/interceptors
21. Axios Error handling - Режим доступу: https://axios-http.com/docs/handling_errors
22. Головна сторінка Native Base - Режим доступу: https://docs.nativebase.io/getting-started
23. Native base list of components - Режим доступу: https://docs.nativebase.io/?utm_source=HomePage&utm_medium=header&utm_campaign=NativeBase_3
24. Native Base themes - Режим доступу: https://docs.nativebase.io/default-theme
25. React Native Async Storage documentation - Режим доступу: https://reactnative.dev/docs/asyncstorage
26. React Native SVG documentation - Режим доступу: https://www.npmjs.com/package/react-native-svg
27. Victory Charts documentation - Режим доступу: https://formidable.com/open-source/victory/docs/native/

28. Eslint documentation - Режим доступу: https://eslint.org/docs/latest/use/getting-started
29. Microsoft .NET documentation - Режим доступу: https://learn.microsoft.com/en-us/dotnet/core/introduction
30. Android Studio documentation - Режим доступу: https://developer.android.com/docs
31. Axios documentation - Режим доступу: https://axios-http.com/ru/docs/intro
32. Ef Core documentation - Режим доступу: https://learn.microsoft.com/en-us/ef/core/
33. .NET CLR - Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/clr
34. Asp Net Web API - Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio
35. C# LINQ documentation - Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/csharp/linq/
36. Ef Core architecture - Режим доступу: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/contact-manager/iteration-4-make-the-application-loosely-coupled-cs
37. Ef Core migrations - Режим доступу: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/
38. PostgreSql documentation- Режим доступу: https://www.postgresql.org/docs/
39. PostgreSql security - Режим доступу: https://www.postgresql.org/docs/7.0/security.htm
40. PostgreSql community - Режим доступу: https://www.postgresql.org/community/
41. PostgreSql extensions - Режим доступу: https://www.postgresql.org/download/products/6-postgresql-extensions/
42. PostgreSql architecture - Режим доступу: https://www.postgresql.org/docs/current/tutorial-arch.html

43. Mediatr documentation- Режим доступу: https://discordnet.dev/guides/other_libs/mediatr.html
44. Fluent validation documentation- Режим доступу: https://docs.fluentvalidation.net/en/latest/
45. Fluent validation localization - Режим доступу: https://docs.fluentvalidation.net/en/latest/localization.html
46. Asp Net Core documentation- Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/limitingidentitybyscheme?view=aspnetcore-7.0
47. Clean Architecture - Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
48. Swagger documentation- Режим доступу: https://swagger.io/docs/