МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Модулі планування та ідентифікації для мобільного додатку PlannIt»

за спеціальністю 122 «Комп'ютерні науки»,

освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91 Бабій Євгеній Андрійович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2020 р.

Науковий керівник _____ к.т.н., доц., Баранова І.В.
(підпис)

Голова комісії _____ Шифрін Д. М
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ

Зав. секцією ІТП

_____ В. В. Шендрик

«___» _____ 2020 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Бабій Євгеній Андрійович

1 Тема проекту *Модулі планування та ідентифікації для мобільного додатку PlannIt*
затверджена наказом по університету від «26» листопада 2020 р. № 1824-III

2 Термін здачі студентом закінченого проекту «11» ___грудня___ 2020 р.

3 Вхідні дані до проекту Функціональні вимоги до процесів, які виконують розроблювані модулі додатку

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, проєктування функціональних процесів, спринт розробки модуля ідентифікації, спринт розробки модуля профілю користувача.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Графічна презентація (актуальність, постановка задачі, аналіз додатків-аналогів, функціональні вимоги, моделювання роботи, діаграма ВВ, архітектура додатку, ескізи основних екранів, засоби реалізації, демонстрація роботи модулів, висновки)

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Аналіз предметної області та додатків-аналогів	1.10.2020-5.10.2020	
2	Визначення мети та задач	6.10.2020-7.10.2020	
3	Вибір методів та інструментів реалізації	8.10.2020-9.10.2020	
4	Моделювання та проєктування процесів модулів додатку	10.10.2020-16.10.2020	
5	Проєктування інтерфейсів	17.10.2020-20.10.2020	
6	Спринт реалізації модуля планування	21.10.2020-16.11.2020	
7	Спринт реалізації модуля профілю користувача	17.11.2020-1.12.2020	
8	Завершення роботи та підведення підсумків	2.12.2020-7.12.2020	

Магістрант _____ Бабій Є.А.

Керівник роботи _____ к.т.н., доц. Баранова І.В.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Модулі планування та ідентифікації для мобільного додатку PlannIT».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 31 найменувань, додатків. Загальний обсяг роботи − 78 сторінки, у тому числі 52 сторінок основного тексту, 4 сторінки списку використаних джерел, 26 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці функціональних модулів додатку, які виконують функції тайм-менеджменту шляхом створення задач на день та ідентифікації користувача.

В роботі проведено аналіз предметної області та порівняльна характеристика додатків-аналогів, характеристика проєкту методом SMART, планування робіт діаграмою Ганта.

Також виконано проектування функціонування додатку, взаємодії між акторами, діаграми послідовностей використання системи.

Представлені спринти розробки модулів планування та ідентифікації, розробка інтерфейсу модулів.

Результатом проведеної роботи є реалізовані модулі планування та ідентифікації додатку.

Практичне значення роботи полягає у тому, що розроблені модулі є функціональними частинами комерційного додатку “PlannIT”.

Ключові слова: мобільний додаток, планування, модульне програмування, спринт, програмне забезпечення.

ЗМІСТ

ВСТУП

Планування майбутніх дій, побудова планів для виконання – важливий і незамінний процес у будь-який момент людського життя. Думки про те, що потрібно робити далі, спіткають усіх: бізнесмена, медпрацівника, актора або звичайного школяра, незалежно від життєвого статусу чи професії. Тому галузь планування будь-чого була й буде присутня для кожного [1].

Сучасний технологічно розвинений світ дозволяє вирішувати проблеми різними способами, методами, демонструє спеціальні технології для цього. Планування є динамічним процесом і потребує можливості доступу до нього у будь-який момент. Раніше використовували блокноти або записні книжки. В наш час оптимально використовувати мобільний телефон. Смартфони дозволяють завантажити потрібний додаток і використовувати його у зручний для вас час [2].

Мобільний додаток — програмне забезпечення, яке спрямоване для роботи на смартфонах та планшетах. Багато мобільних застосунків можна завантажити на пристрій з онлайн магазинів мобільних цифрових товарів [3].

Спочатку мобільні застосунки були інструментами для менеджменту інформації, аналоги електронної пошти, календарю, списку особистих контактів, примітки та інформацію про біржовий ринок. Концепція мініатюрного функціонального комп’ютера набирала попит, а наявність інструментів для розробників призвели до швидкого розвитку асортименту додатків для всіх категорій пристроїв. З розвитком програмного забезпечення для телефонів, велика кількість повсякденних функцій перейшла під контроль смартфонів [4].

Об’єктом дослідження є інформаційні технології, які виконують функції планування задач для виконання. Предметом дослідження є мобільний додаток планування особистого розпорядку. Метою магістерської роботи є реалізація проєкту, що передбачає розробку модулів створення та ідентифікації задач мобільного додатку для планування своїх цілей та подій. Для досягнення даної мети були поставлені наступні задачі:

- аналіз предметної області, аналогів та потенційних користувачів;
- формування технічного завдання для майбутнього продукту;
- створення ескізів та дизайну інтерфейсу користувача;
- реалізація модулів мобільного додатку.

Практичне значення роботи полягає в тому, що проєкт є складовою комерційного проєкту по запуску мобільного додатку в цифровий продаж, а модулі, розроблені в дипломній роботі, є складовою основного функціоналу комерційного додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження сфери роботи

Планування — заздалегідь визначений і зрозумілий порядок дій, які потрібні для досягнення певної цілі. Плануванням називають оптимальний розподіл ресурсів для досягнення поставленої мети. За замовчуванням, під визначенням ресурси мається на увазі час. Велика кількість сфер функціонують за допомогою грамотного тайм-менеджменту [5].

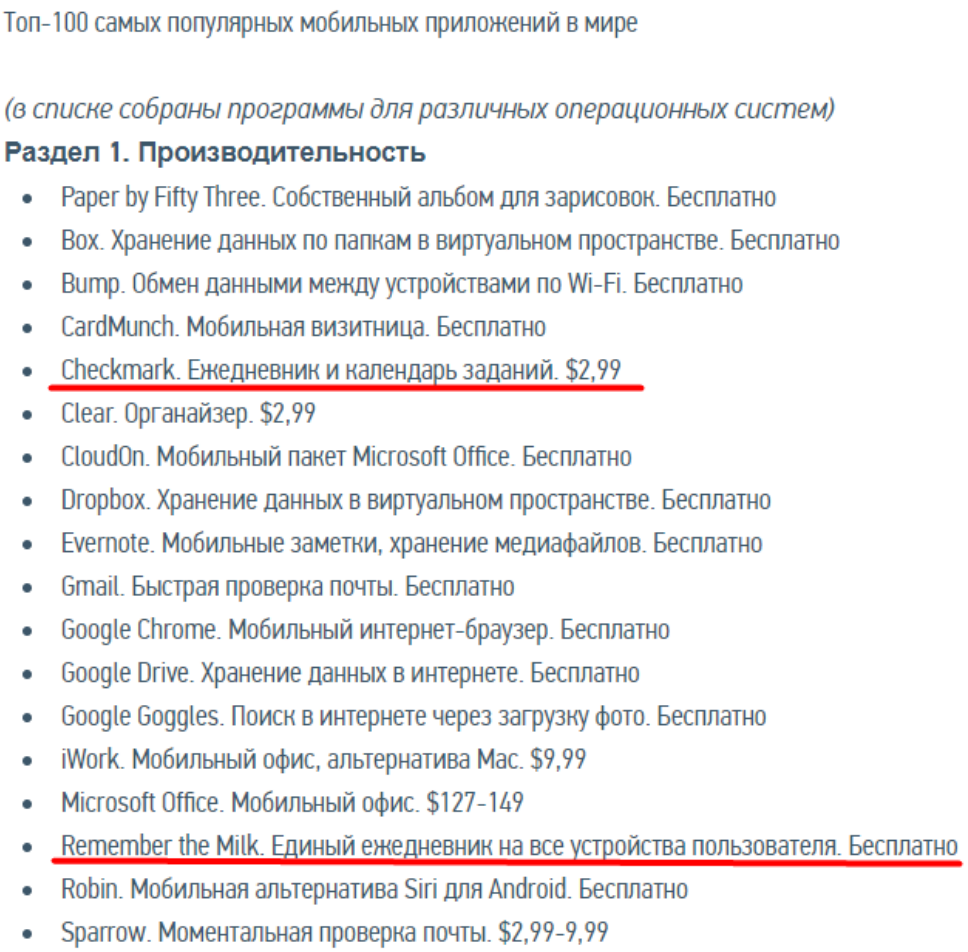
Тайм-менеджмент — сукупність методик оптимальної організації часу для виконання поточних задач, проєктів та календарних подій. Типовими підходами в керуванні часом є постановка пріоритетів, розбиття великих завдань та проєктів на окремі дії та делегування іншим людям. Головними допоміжними інструментами для керування часом є особистий календар, список поточних завдань та список проєктів [6].

Одним з візуальних методів формування своїх планів відносно часу є діаграма Ґанта, яка використовується для ілюстрації плану, графіка робіт за будь-яким проєктом. Така діаграма є одним з засобів планування та управління проєктами, і являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню [7].

1.2 Аналіз проекту в порівнянні зі схожими

Концепцію діаграми Ґанта широко використовують у сучасному плануванні роботи, тому задля зручності, її імпортували у цифровий вид і на основі цього розробляють різноманітні мобільні застосунки для планування. Тайм-менеджмент

додатками активно користуються люди усього світу. Даний вид додатків зустрічається двічі у світовому рейтингу мобільних застосувань по категоріям [8].



Топ-100 самых популярных мобильных приложений в мире

(в списке собраны программы для различных операционных систем)

Раздел 1. Производительность

- Paper by Fifty Three. Собственный альбом для зарисовок. Бесплатно
- Box. Хранение данных по папкам в виртуальном пространстве. Бесплатно
- Bump. Обмен данными между устройствами по Wi-Fi. Бесплатно
- CardMunch. Мобильная визитница. Бесплатно
- Checkmark. Ежедневник и календарь заданий. $2,99
- Clear. Органайзер. $2,99
- CloudOn. Мобильный пакет Microsoft Office. Бесплатно
- Dropbox. Хранение данных в виртуальном пространстве. Бесплатно
- Evernote. Мобильные заметки, хранение медиафайлов. Бесплатно
- Gmail. Быстрая проверка почты. Бесплатно
- Google Chrome. Мобильный интернет-браузер. Бесплатно
- Google Drive. Хранение данных в интернете. Бесплатно
- Google Goggles. Поиск в интернете через загрузку фото. Бесплатно
- iWork. Мобильный офис, альтернатива Mac. $9,99
- Microsoft Office. Мобильный офис. $127-149
- Remember the Milk. Единый ежедневник на все устройства пользователя. Бесплатно
- Robin. Мобильная альтернатива Siri для Android. Бесплатно
- Sparrow. Моментальная проверка почты. $2,99-9,99

Рисунок 1.1 - Рейтинг мобільних додатків по категоріям

Торгівельний простір мобільних застосунків у наш час неосяжно великий, з усіма аналогами ознайомитись неможливо, тому розглянемо і порівняємо можливості деяких мобільних додатків для тайм-менеджменту з загальних рейтингів двох найбільших інтернет магазинів – Google Play та App Store [9].

Додаток Any.DO

Програма Any.DO - один з найпопулярніших планувальників серед користувачів Android і iOS. Додаток відрізняється зручним і простим інтерфейсом і може синхронізуватися з декількома пристроями. Для додавання завдань в додатку можна користуватися голосовим набором - при цьому включається інтелектуальне введення, що дозволяє вибрати запис з готових варіантів [10].

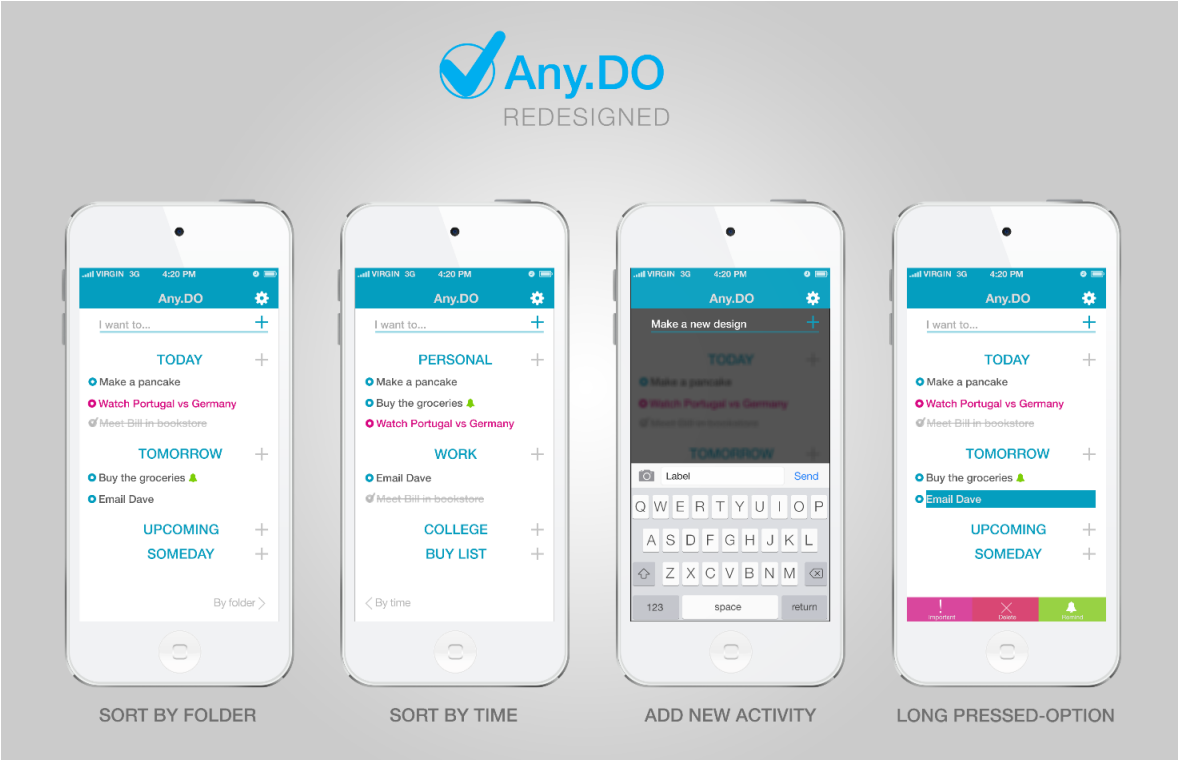


Рисунок 1.2 - Інтерфейс додатку «Any.DO»

Переваги планувальника Any.DO:

- введення тексту з можливістю прикріплення до нього відеофайлів, зображень або фото - можливість, відсутня у більшості схожих утиліт;
- просте додавання власних списків з головного меню;
- зручне перемикання між режимами;
- робота з різними видами даних - в тому числі списками справ, покупок.

Додаток працює не тільки на смартфонах, але і в браузері на ПК. Інтеграція з операційною системою дозволяє йому видавати повідомлення прямо в рядку стану. Серед інших плюсів варто відзначити мульти-користувацьку роботу, захист інформації за допомогою коду і геотеги. Базова версія програми є безкоштовною, але функціональність можна розширити, використовуючи платну версію [11].

Додаток Remember the milk

Програма, яка забезпечує ефективне планування зі зберіганням інформації на сервері розробника. У планувальника є також підтримка користувацьких списків і тегів, синхронізація і просте керування [12].

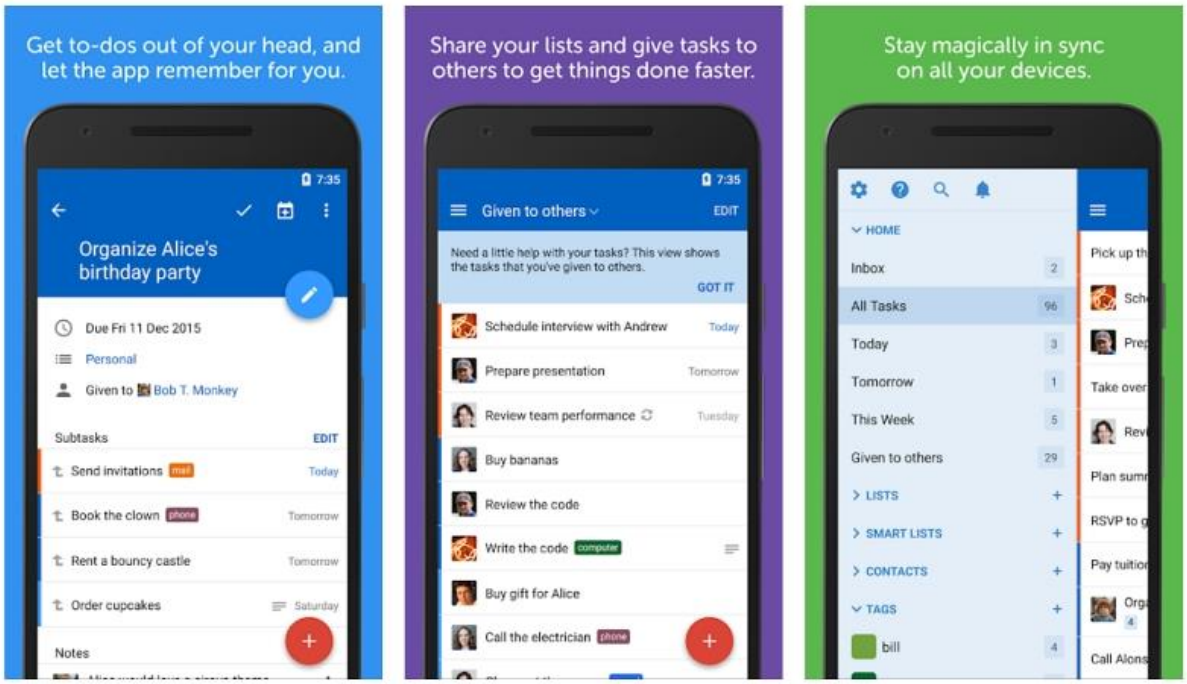


Рисунок 1.3 - Інтерфейс додатку «Remember the milk»

Переваги планувальника Remember the milk:

- контроль робочих процесів за допомогою сортування і розрахованої на багато користувачів взаємодії;
- переміщення всіх завдань за замовчуванням в папку вхідних;
- зв'язок завдань з геолокацією і контактами;
- систему нагадувань, що взаємодіє з різними сервісами, від Evernote і Microsoft Outlook до Твіттера.

Одним з головних переваг утиліти є підтримка практично всіх популярних операційних систем: Android, iOS, BlackBerry OS, macOS і Windows. Ще одна важлива особливість - отримання повідомлень про заплановані справи на пошту, по SMS і в месенджерах. У платній версії також є необмежена автосинхронізація і PUSH-повідомлення [11].

Додаток Trello

Додаток Trello є продуктом, що створений за тим же принципом, що і програмне забезпечення для управління великими проєктами. З її допомогою можна створювати завдання для колективу, сім'ї і навіть інтернет-магазину [13].

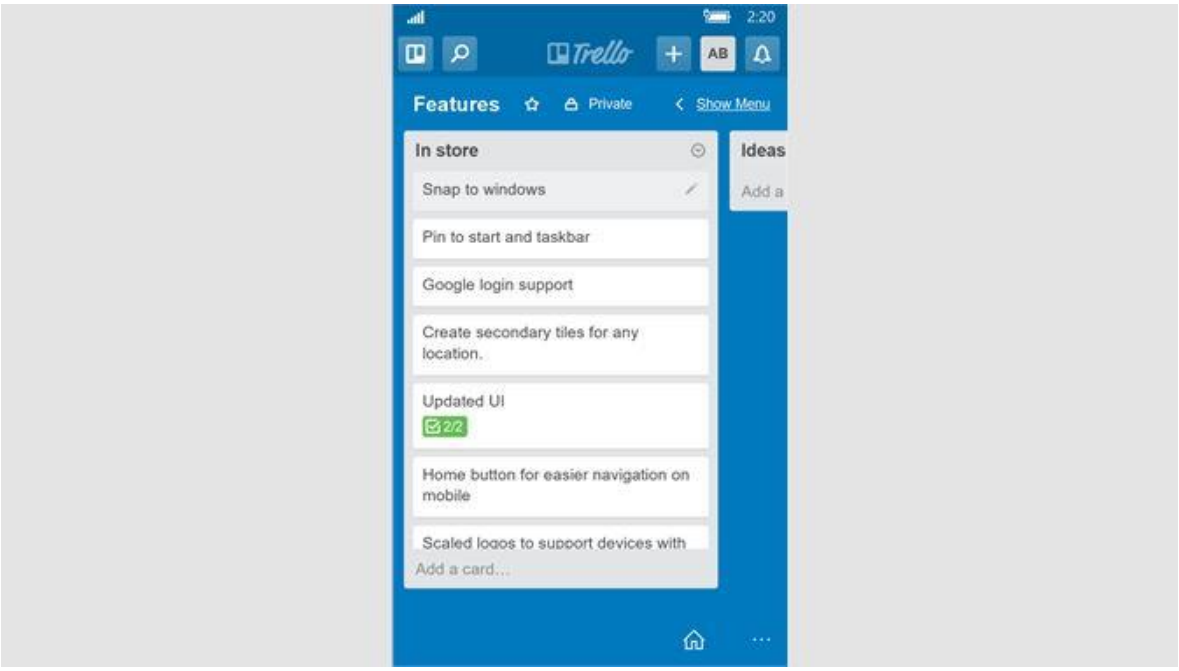


Рисунок 1.4 - Інтерфейс додатку «Trello»

Переваги планувальника Trello:

- створення списків завдань для індивідуального і загального використання;
- запрошення в групу колег, членів сім'ї та друзів;
- призначення завдань іншим користувачам;
- відповіді на коментарі;
- прив'язка карток до координат на карті;
- візуалізація задач.

Програма підтримується різними операційними системами - Microsoft Windows, macOS, iOS і Android. Серед її плюсів - можливість стежити за проєктами, простий і зручний інтерфейс, завантаження файлів і налаштування дедлайнів. При цьому планувальник не вимагає оплати за використання [11].

Додаток Google Calendar

Календар Google – безкоштовна відкрита інтерактивна служба "Календар". За допомогою календаря Google простіше відстежити всі важливі життєві події – дні

народження, збори, спортивні заходи, прийоми в лікарів – усе в одному місці. За допомогою календаря Google можна легко призначати заходи й розсилати запрошення, надавати до них доступ друзям і родичам (чи зберігати тільки для особистого використання), а також знаходити цікаві для вас заходи в Інтернеті [14].

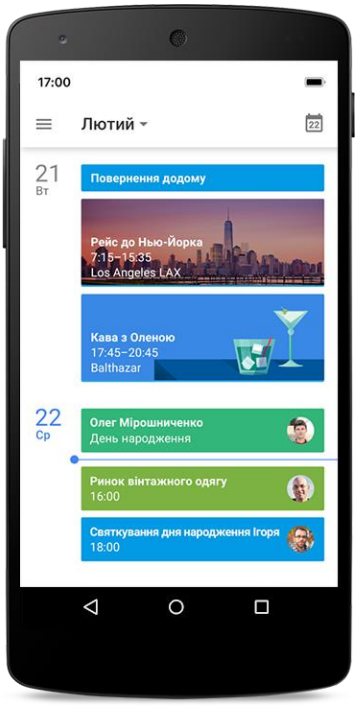


Рисунок 1.5 - Інтерфейс додатку «Google Calendar»

Переваги планувальника Google Calendar:

- широкий асортимент різнотипного функціоналу;
- нетривіальний інтерфейс користувача;
- повна взаємодія та синхронізація з акаунтом Google;
- наявність десктопної та веб версій;
- взаємодія з іншими програмами схожого типу (iCalendar, MS Outlook та ін.);
- візуалізація габаритного списку задач [11].

Кожен з вищерозглянутих мобільних додатків-аналогів має свої функціональні, візуальні та структурні особливості. Дані про них проаналізовано і складено порівняльну таблицю особливостей кожного представлених застосунків.

Таблиця 1.1. Порівняльний аналіз існуючих аналогів

Характерна ознака	Назва розглянутого додатку			
	Any.DO	Remember the milk	Trello	Google Calendar
Кросплатформність	✓	✓	✓	✓
Наявність десктопної або web-версії	✓	✓	✓	✓
Синхронізація даними з іншими користувачами додатку	✓	✓	✓	✓
Взаємодія з іншими сервісами чи додатками	-	✓	✓	✓
Нетривіальна концепція користувацького інтерфейсу	-	-	✓	-
Інтуїтивно-зрозумілий алгоритм користування	✓	-	✓	-
Мультимовність інтерфейсу	✓	✓	✓	✓
Безкоштовний доступ до базових функцій додатку	✓	✓	✓	✓
Платний доступ для додаткових можливостей	✓	✓	-	✓
Функціональна можливість, що вирізняє додаток серед інших	✓	✓	✓	✓
Наявність ігрової складової	-	-	-	-

Проаналізувавши предметну область застосування інформаційних технологій для вирішення поставленої задачі, з огляду та порівняння мобільних застосунків-аналогів, можна сформулювати основну проблему дослідження.

Проблема дослідження полягає у тому, що жоден з розглянутих мобільних додатків не має всі представлені характеристики. Кожен застосунок має унікальний

набір характеристик, але він не є оптимальним набором для зручного й багатофункціонального користування.

Після аналізу таблиці 1.1 вірним рішенням у цьому випадку є розробка необхідних модулів для власного мобільного додатку, який буде володіти всіма виділеними функціями.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі

Отже, після формування вирішення проблеми дослідження та списку ознак оптимального додатку, можна визначити ціль магістерської роботи.

Мета роботи − реалізація модулів планування та ідентифікації в мобільного додатку, що дозволить користувачеві керувати своїми запланованими задачами та персональним профілем.

Мета вважається досягнутою тільки тоді, коли дана проблема обраної сфери усунута шляхом виконання низки хронологічних етапів втілення продукту:

- Проаналізувавши предметну область, зрозуміти тренди, технології та патерни руху обраної сфери.

- Після огляду аналогів рішення основної проблеми дослідження виділити ключові моменти для розгляду та покращення особистого майбутнього продукту.

- На основі висунутих характеристик та вимог, сформувати технічне завдання розроблюваного додатку.

- Спроектувати інтерфейс користувача застосунку шляхом втілення усіх вимог та дотримання характеристик, описаних у технічному завданні.

- Реалізувати модулі мобільного додатку відповідно до обраної методології Scrum, використовуючи спринти для розробки та розглянуті інструменти реалізації застосунків.

Розроблюваний мобільний додаток в цілому надасть можливість користуватися функціоналом інтерфейсів із зазначеними характеристиками, що описані нижче:

- кросплатформність продукту – можливість користуватись на операційній системи Android та IOS;

- взаємодія між користувачами;

- взаємодія з іншими сервісами чи додатками;

- нетривіальна концепція інтерфейсу користувача додатку;

- простий алгоритм користування;

- мультимовність інтерфейсу;

- безкоштовний доступ до базових функцій;

- наявність ігрової складової.

Модуль планування відповідатиме за роботу із сторінками планування та редагування завдань. Дані сторінки повинні виконувати такі функції:

Сторінка планування на день

- повинно бути перемикання між днями, як +1/ -1 день, так і на вибір;

- нове модальне вікно чи скрін, в якому можна створити завдання;

- якщо користувач клацає на існуючу задачу, то він може її відредагувати чи видалити;

- відображення червоної лінії на графіку, щоб було видно поточний момент;

- повинно бути нормальне відображення завдань, які паралельні в часі (типу стека завдань або таймлайну);

Сама задача виглядає на графіку, як квадрат, в якому є опис завдання.

Приклади: google calendar, any.do, focuster.

Створення, редагування, видалення завдань

Являє собою модальне вікно або окремий скрін. На цій сторінці повинні бути:

- назва;

- можливість створювати суб-завдання;

- час початку завдання;

- час закінчення завдання;

- щоденне оповіщення на початку дня про завдання;

- сповіщення по telegram, whatsapp (за скільки часу і коли) для premium користувачів.

При цьому повинна бути можливість створення завдань паралельних в часі. Прикріплення файлів до задачі не передбачено. При видаленні завдання, якщо вона є повторюваною, з'явиться модальне вікно, в якому можна буде видалити цю задачу або видалити всі повторювані.

Модуль ідентифікації відповідатиме за роботу із сторінкою профілю користувача. Вона повинна виконувати такі функції:

Сторінка профілю користувача

Сторінка відображає особисту інформацію та параметри додатку користувача з функцією їх зміни. Основні вимоги:

- email;
- profile name;
- вибір часу за замовчуванням тривалості завдання;
- вибір мови;
- можливість прив'язки кредитної картки;
- номер телефону з можливістю зміни;
- можливість видалення облікового запису.

2.2 Вибір методів та інструментів реалізації

Для реалізації даного проєкту було обрано методологію Scrum. Найважливіший елемент методології Scrum - це Sprint (Спринт).

Спринт - відрізок часу, який береться для виконання певного (обмеженого) списку завдань, що покращують чи створюють новий продукт. Рекомендується брати 2-4 тижні (тривалість визначається від складності). Абсолютно все в ній крутиться навколо спринту, бо саме під час нього відбувається створення продукту. Зазвичай, тривалість спринту становить близько 30 днів (1 місяць), але іноді його роблять рівним двом тижням. Думки з цих питань розділилися, так як деякі вважають, що підготувати і організувати спринт на 30 днів набагато важче, ніж на два тижні, що є логічним.

У методології Scrum при завершенні спринта повинен обов'язково вийти деякий результат, який відрізняється від попереднього. Варто, однак, розуміти, що це може бути не закінчений продукт як такої, адже він може вдосконалюватися нескінченно. Тут потрібно триматися орієнтира: закінчення спринту – явний результат,

відповідний до описаного в документації. Наступний спринт вже, наприклад, покращує його, і, знову ж таки, в кінці спринту дає новий результат чи продукт [15].

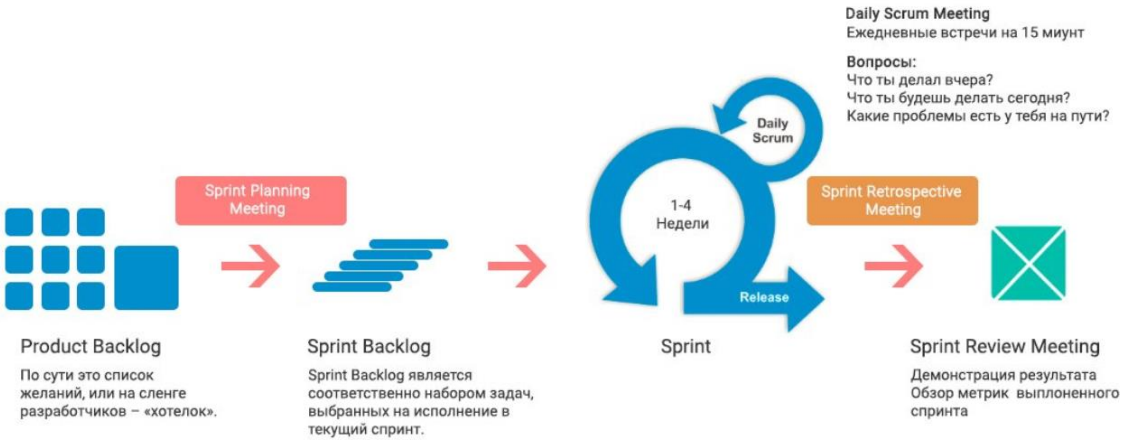


Рисунок 2.1 – Життєвий цикл одного спринту

Як видно з рисунку, життєвий цикл методології Scrum складається з підготовчих етапів для спринту і завершальних етапів. На кожному цьому етапі відбувається певна подія [16].

Інструментом для реалізації frontend складової обрано фреймворк Java Script – React Native. React Native – це інструмент для розробки мобільних додатків з відкритим вихідним кодом, створений Facebook і співтовариством. Ви можете створювати додатки для Android і iOS, використовуючи фреймворк. Це потужний кросплатформний інструмент для розробки мобільних додатків, який дозволяє їх відносно швидко створювати нові застосунки. Фреймворк стрімко набирає популярність серед розробників [17].

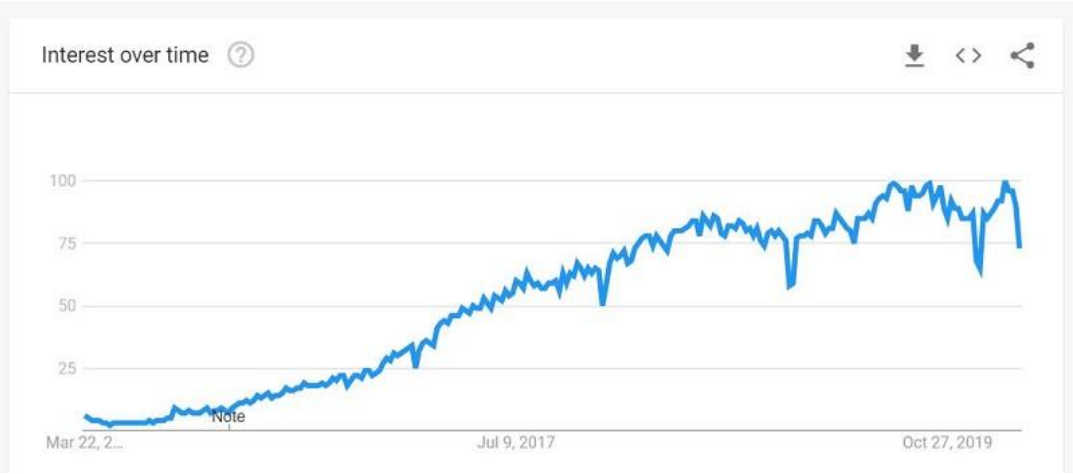


Рисунок 2.2 – Графік зростання популярності React Native (процент зацікавлених спеціалістів)

React Native надає компоненти для тексту, зображень, введення з клавіатури, гортання списків, індикатора виконання, анімації, буфера обміну, посилань і т.д. Ці компоненти значно прискорюють процес розробки додатків, а функція «Hot Reloading» також економить багато часу, оскільки дозволяє перезавантажити програму без повторної компіляції всього коду.

Інструментом для реалізації backend складової обрано фреймворк Java Script – NodeJS. Процес синхронізації з NodeJS є швидким і організованим, так як події керують архітектурою, яка обслуговує як клієнтську, так і серверну сторону. Цикл обробки подій через протокол веб-сокета обробляє багатокористувацьку функцію. Він працює в TCP і уникає перевантаження HTTP. NodeJS також робить RTA легким, масштабується, які обслуговує і зручним з точки зору розробки програмного забезпечення [18].

У 2019 році за даними найбільшого в світі опитування серед розробників StackOverflow, Node.js залишається найпопулярнішим фреймворком. Цю технологію використовує майже 50% від усіх фахівців [19].

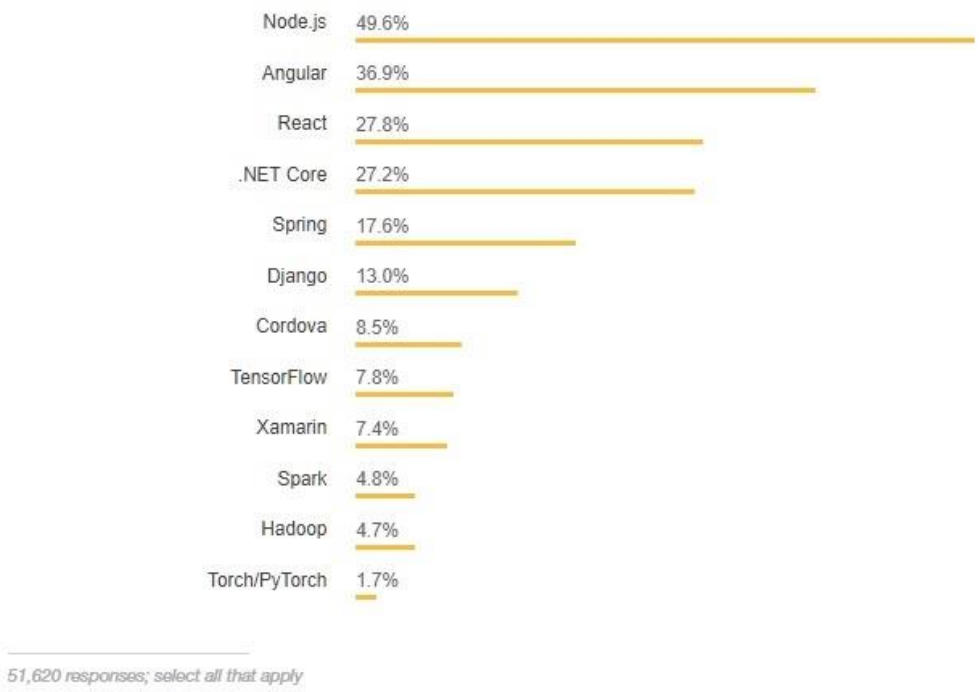


Рисунок 2.3 – Графік зростання популярності NodeJS (процент зацікавлених спеціалістів)

3 ПРОЕКТУВАННЯ ФУНКЦІОНАЛЬНИХ ПРОЦЕСІВ

Моделювання розроблюваного мобільного додатку є одним з найважливіших етапів створення якісного програмного продукту. Під час даного процесу розробляються моделі функціонування додатку, проектування userflow користувача, визначаються основні етапи функціонування, створюються ескізи та формулюються вимоги користувача до майбутнього інтерфейсу застосунку, з'ясовуються інформаційні архітектури.

3.1 Моделювання процесу основної функції мобільного додатку

IDEF0 (Integrated DEFinition) - методологія функціонального моделювання та графічна проекція, призначена для зображення і характеристики бізнес-процесів в своєрідній ієрархічній формі. Відмінною особливістю IDEF0 є саме акцент на підпорядкованість об'єктів. Метод IDEF0 використовує головне вікно для представлення функцій у процесі та показує відношення до дочірньої та батьківської основ. Ця методика дає креслення для розуміння систем організації [20].

Можна виділити основні переваги побудови контекстної діаграми IDEF0:

− забезпечує простий та зрозумілий системний аналіз;

− пропонує покращені методи відносин та спілкування;

− дає розуміння про систему, її складові та їх зв’язки.

Діаграму IDEF0 наведено на рис. 3.1. Вона демонструє процес роботи мобільного додатку при виконанні основного призначення.

На вході ми отримуємо потребу користувача у створенні списку задач, а на виході отримуємо цей самий список задач у вигляді таймлайну.

Управліннями розробки мобільного додатку є нормативна документація, тобто технічне завдання додатку, а механізмами в свою чергу: користувач, мобільний додаток та база даних.

Точку зору моделювання процесу роботи мобільного додатку представляє розробник програмного забезпечення.

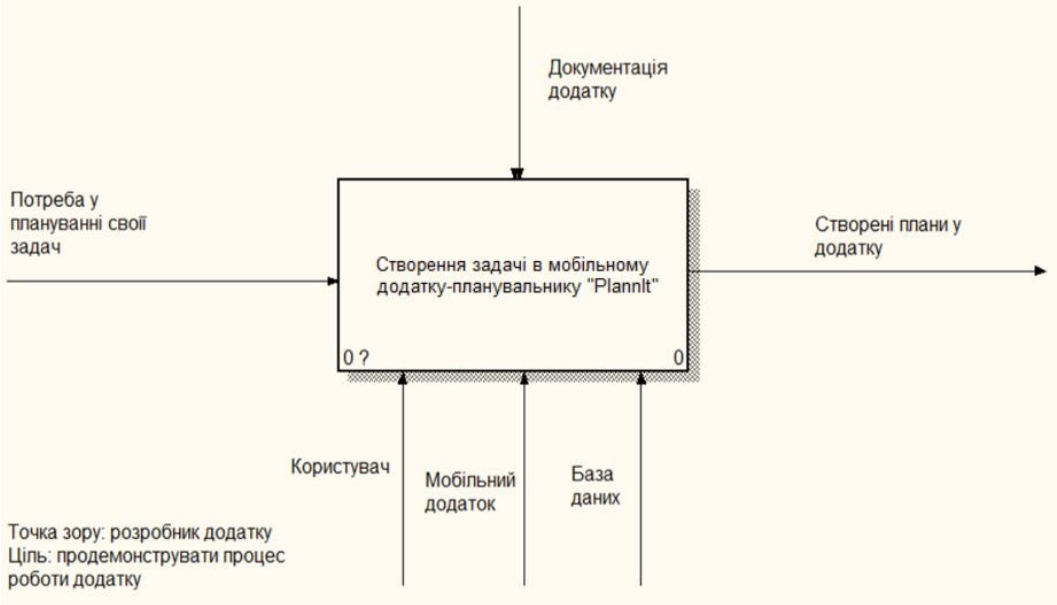


Рисунок 3.1 – Контекстна діаграма IDEF0 для роботи додатку

Декомпозиція першого рівня у нотації IDEF0 відображає взаємодію процесів роботи мобільного додатку (рис. 3.2).

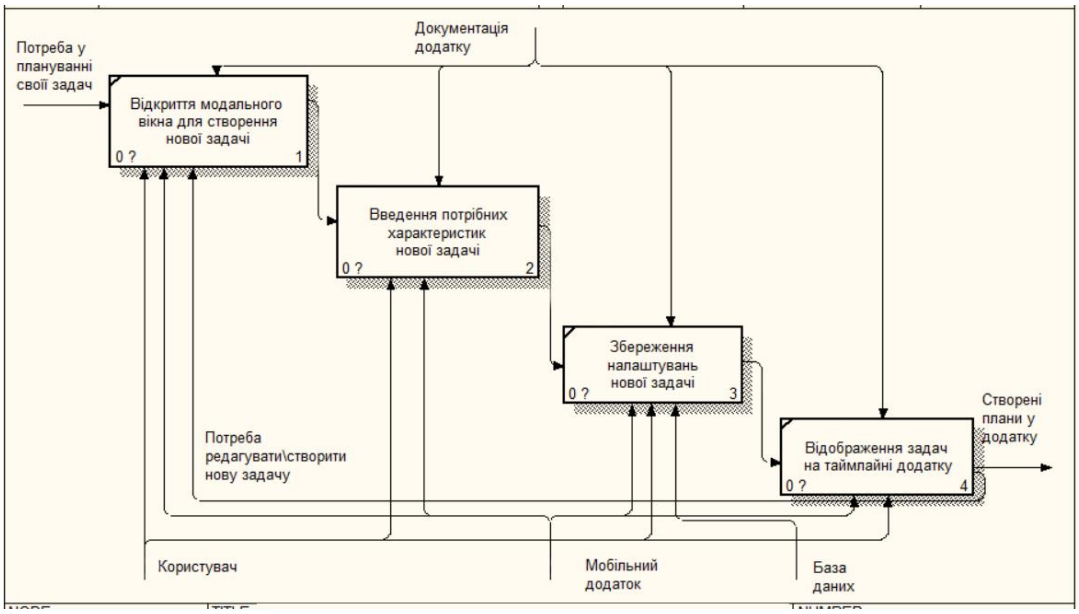


Рисунок 3.2 – Перший рівень декомпозиції IDEF0 для роботи додатку

Декомпозиція другого рівня у нотації IDEF0 відображає взаємодію процесів задання характеристик нової задачі мобільного додатку (рис. 3.3).

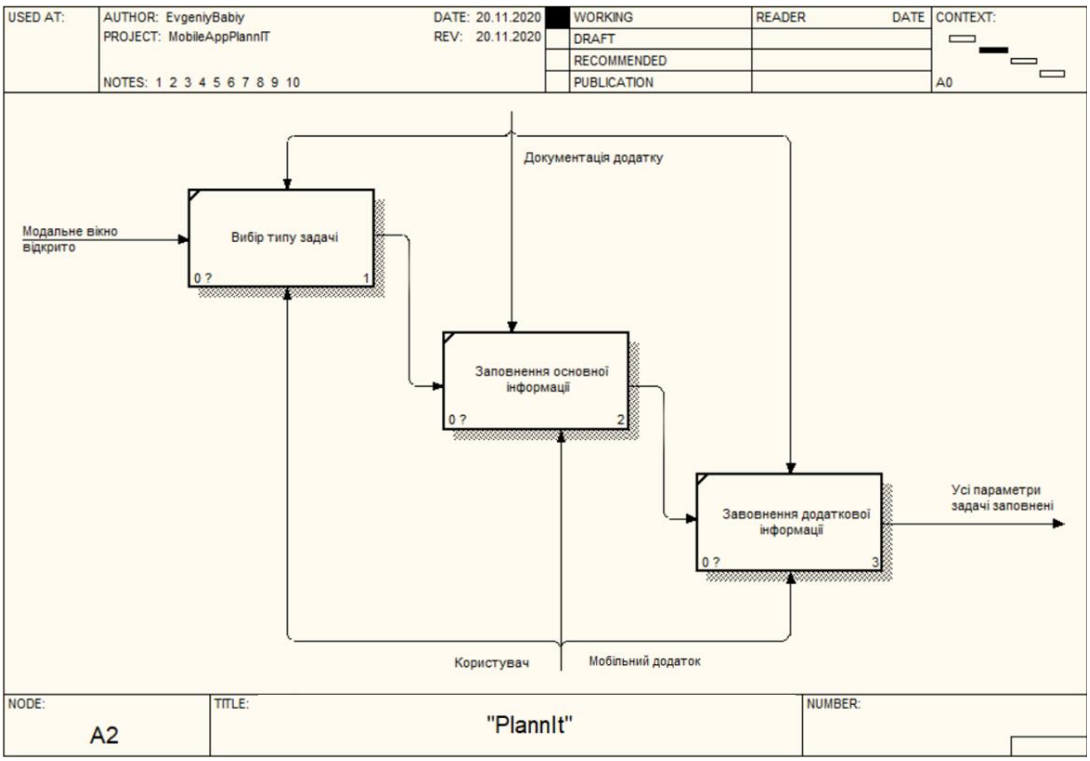


Рисунок 3.3 – Другий рівень декомпозиції IDEF0 для роботи додатку

3.2 Моделювання функціонування системи

Щоб зрозуміти функціювання програмного продукту використовується модель варіантів використання, яка включає акторів, варіанти використання системи, взаємодію між ними.

Діаграма варіантів використання являє собою список дій, які зазвичай визначають взаємодію між роллю (відомій в уніфікованій мові моделювання як актор), і системою для досягнення мети [21].

Актором може бути людина чи інша зовнішня сутність. Діаграма варіантів використання допомагає представити:

− сценарії, в яких система чи додаток взаємодіють з людьми, організаціями чи зовнішніми системами;

– роль та значимість кожного окремого процесу чи актору в системі;

− обсяг розроблюваної системи.

Користувач – людина, що має доступ до функціоналу мобільного додатку.

БД – система, що надає додатку інформаційну наповненість, збереження даних, які використовуються мобільним додатком.

Після детального аналізу було виділено наступні основні варіанти використання даного мобільного додатку:

- ВВ1 Запуск додатку – починає роботу додатку та запускає усі необхідні процеси, ініціатором є користувач;
- ВВ2 Створення задачі – формування задачі шляхом заповнення інформацією модального вікна створення, ініціатором є користувач, у процесі бере участь актор БД;
- ВВ3 Відображення таймлайну – формування списку створених задач у вигляді лінійної шкали часу, ініціатором є актор БД;
- ВВ 4 Відкриття модального вікна створення задачі – перехід до вікна з полями для введення інформації, що стосується обраної задачі, ініціатором є користувач;
- ВВ 5 Задання параметрів – заповнення полів модального вікна потрібною інформацією, яка характеризує нову задачу, у процесі бере участь користувач;
- ВВ 6 Підтвердження створення – підтвердження створення задачі шляхом натискання відповідної кнопки модального вікна, ініціатором є користувач, у процесі бере участь актор БД;
- ВВ7 Перехід до профілю користувача – відкриття розділу, де відображається інформація та параметрами користувача, ініціатором є користувач, у процесі бере участь актор БД;
- ВВ8 Перегляд інформації профілю – відображення усіх даних профілю користувача, ініціатором є користувач;
- ВВ9 Зміна даних користувача – зміна даних користувача у розділі профілю, ініціатором є користувач, у процесі бере участь актор БД;

- ВВ10 Авторизація – введення даних після запуску додатку для входу до особистого акаунту, ініціатором є БД, у процесі бере участь користувач.

На основі виділених вище акторів та ВВ розроблена діаграма варіантів використання, наведена на рис 3.4.

Рисунок 3.4 – Діаграма ВВ користувача та БД

3.3 UML-моделювання діяльності

При моделюванні поведінки проектованої системи часто виникає необхідність в деталізації алгоритмічної і логічної реалізації виконуваних системою операцій. Для цього в UML використовується діаграми діяльності.

Існування UML-діаграми зумовлено тим, що недостатньо перерахувати й показати усі необхідні функції, важливо продемонструвати їх послідовність, рух

відповідальності за різні функції та хронології роботи кожного з учасників процесу функціонування системи.

На основі виконання основної функції додатку, а саме створення задач для планування, було сформовано UML-діаграму послідовності додатку:

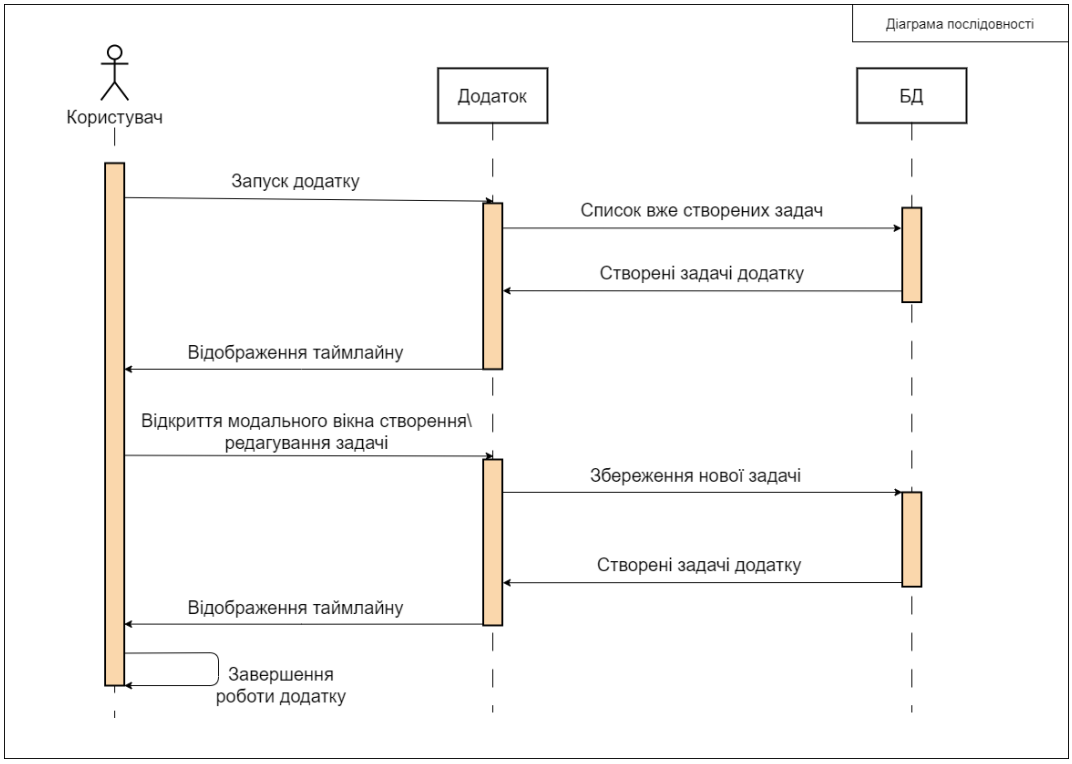


Рисунок 3.5 – Діаграма послідовності системи

Потік діяльності процесу:

- Користувач запускає додаток;
- Додаток дає запит на список вже створених задач;
- БД передає інформацію про створені задачі;
- На основі отриманого списку задач додаток формує таймлайн;
- Користувач створює або редагує задачу у модальному вікні й зберігає його;
- Додаток відсилає створені чи оновлені дані до БД;
- БД відсилає оновлений список задач;
- Додаток оновлює таймлайн із задачами;
- Користувач завершує роботу з додатком.

3.4 Структура використовуваних даних додатку

Для коректного функціонування мобільного додатку і використання всіх можливостей програмного продукту задіяно базу даних (БД), де будуть зберігатися дані. Розглянемо використані таблиці з їх полями.

– Таблиця TASK містить у собі поле ID задачі та користувача типу integer та поля назви, дати початку та кінця задачі, виду та статусу задачі типу varchar. Дана таблиця призначена для даних задач, зберігає їх параметри та пов’язана з таблицею USER шляхом зв’язку з полем users id.

– Таблиця USER має поле id типу integer та поля ім’я, пошти, кредитів задач, часового поясу (поточного часу користувача), поточної дати, токену для авторизації при запуску додатку, контролю версії, дати створення та видалення акаунту з типами varchar та float. Дана таблиця призначена для даних користувача, які його ідентифікують та зберігають персональну інформацію. З таблицею пов’язані таблиці USER та USERHASACHIEVEMENT з допомогою поля users id.

– Таблиця USERHASACHIEVEMENT має поле id типу integer та поля user id та achievement id типів integer. Дана таблиця призначена для зв’язку користувача з певним досягненням у додатку, тобто формує список досягнень користувача. З таблицею пов’язані таблиці USER та ACHIEVEMENT з допомогою поля users id та achievements id.

– Таблиця ACHIEVEMENT має поле id типу integer та поля назви, типу та кількості кредитів винагороди з типами varchar та integer. Дана таблиця призначена для зберігання даних досягнень, що є одним із атрибутів користувача, але більш складним, ніж інші. Пов’язана з таблицею USERHASACHIEVEMENT з допомогою поля achievements id.

– Таблиця SUBTASK має поле id типу integer та поля назви, статусу виконання з типами varchar та tinyinteger. Дана таблиця призначена для сутності субзадача, яка є одним із атрибутів задачі, що дозволяє сегментувати основну задачу. Пов’язана з таблицею TASK з допомогою поля tasks id.

На основі виділених таблиць та їх атрибутів можемо зробити діаграму типу «сутність-зв'язок». ER-діаграма мобільного додатка представлена на рис. 3.6.

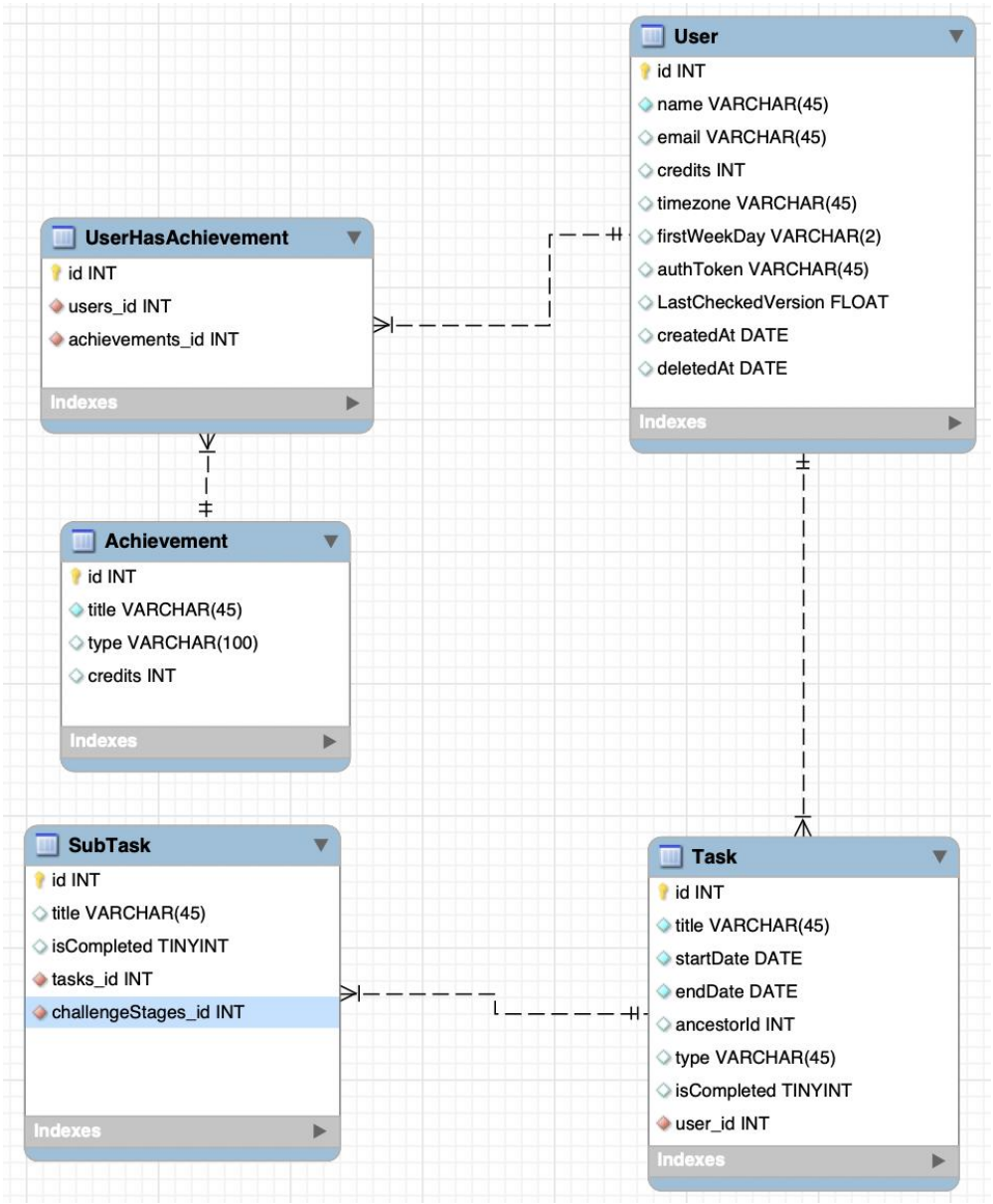


Рисунок 3.6 – ER-діаграма архітектури даних додатку

3.5 Проєктування інтерфейсу користувача

Інтерфейс користувача (User Interface, UI) — засіб зручної та комфортної функціональної взаємодії користувача з інформаційною системою, що має графічну обгортку. UI є сукупністю засобів для обробки та відбиття інформації, оптимально пристосованих для ефективної взаємодії з користувачем [22].

User Interface має важливе значення у проектування й в подальшому використанні додатку. Він забезпечує контекстне візуальне враження для користувача, тому це клопіткий і відповідальний етап роботи.

Для демонстрації глобальних і основних закономірностей будь-якої графічної структури використовують ескіз. Ескіз - це зображення тимчасового характеру, що містить зображення керуючих елементів зображення на основі яких в подальшому будуються інші. Його виконують без точного дотримання масштабу та розмірів, але з приблизним дотриманням пропорцій. Також у ескізі не має сенсу колір та тіні. Ескізи служать для вираження концепції та розкриття її суті у наборі складових частин [23].

Для мобільного додатку було використано зонування робочої області, в результаті чого створено ескізи головного екрану, профілю, модального вікна створення задач, що зображені на рис. 3.7 - 3.9.

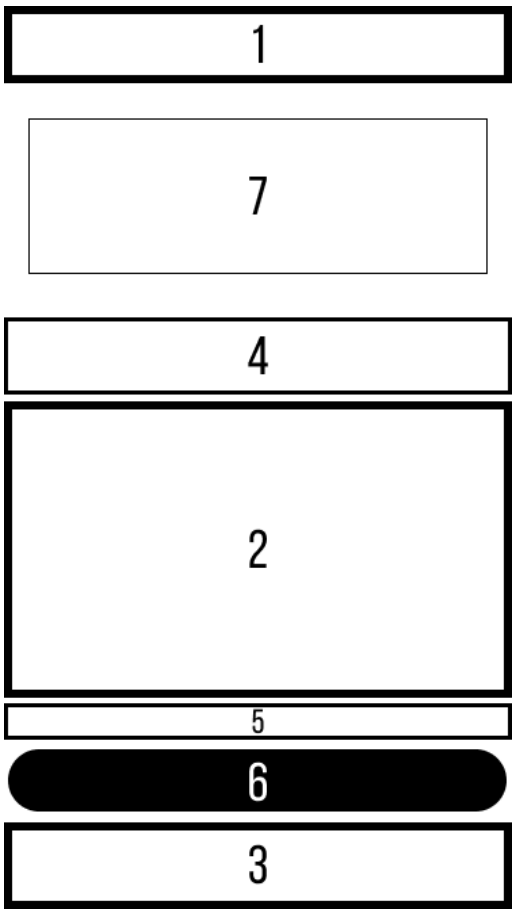


Рисунок 3.7 – Ескіз головного екрану з таймлайном

Функціональні зони робочої області:

1) Додаткова навігаційна область або хедер;
2) Візуальна область таймлайну;
3) Основна навігаційна зона додатку;
4) Функціональна зона таймлайну;
5) Зона для шкали таймлайну;
6) Зона для кнопки виконання головної функції даного екрану;
7) Зона для додаткової інформації та контенту.

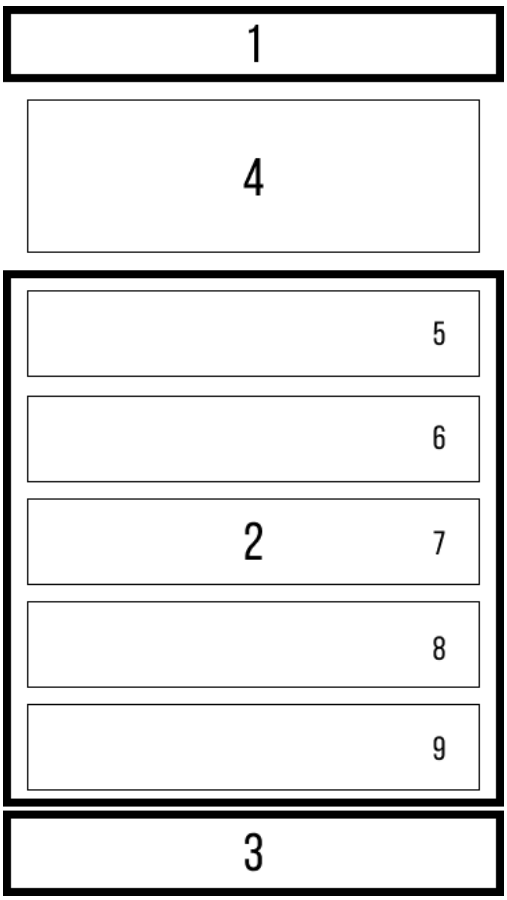


Рисунок 3.8 – Ескіз профілю користувача

Функціональні зони робочої області:

1) Додаткова навігаційна область або хедер;
2) Область параметрів профілю;
3) Основна навігаційна зона додатку;
4) Акцентована інформація профілю;

5-9) Параметри профілю.

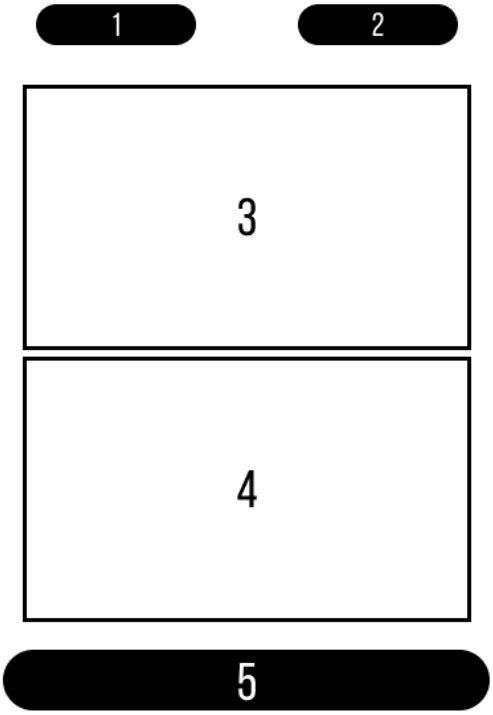


Рисунок 3.9 – Ескіз модального вікна створення задач

Функціональні зони робочої області:

1-2) Зона для кнопок обрання типу створеної задачі;

3) Область основних параметрів задачі;

4) Область додаткових параметрів задачі;

5) Зона для кнопки створення задачі.

На основі продемонстрованих ескізів будуватиметься макет дизайну усіх екранів та вікон мобільного додатку.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

Реалізація дипломного проекту складалася з двух спринтів, що в свою чергу мають декілька етапів (рис.4.1).

Рисунок 4.1 – Кроки реалізації проекту

4.1 Спринт розробки модуля планування

Створення будь-якого продукту починається з дизайну. Орієнтуючись на створений раніше ескіз, було зроблено макет інтерфейсу екрану планування та модального вікна створення задачі.

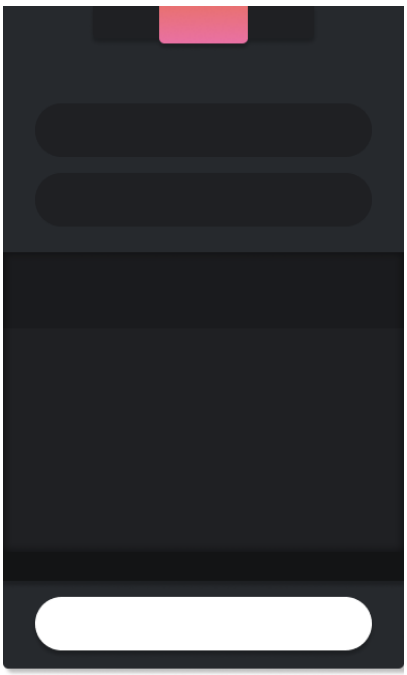


Рисунок 4.2 – Розмежування сторінки таймлайну на області та створення блоків

Після розмежування та додання блоків заповнимо робочу область функціональним текстом та контентом, а також позначимо місця для зображень-іконок.

Рисунок 4.3 – Додаткові елементи, шаблон тексту та іконок сторінки таймлайну

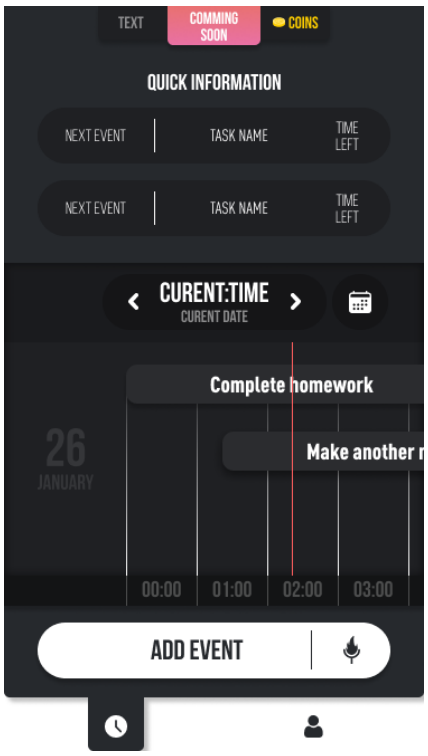


Рисунок 4.4 – Фінальний вигляд екрану перегляду створених задач

Після завершення дизайну сторінки таймлайну переходимо до дизайну модального вікна для створення та редагування задач.

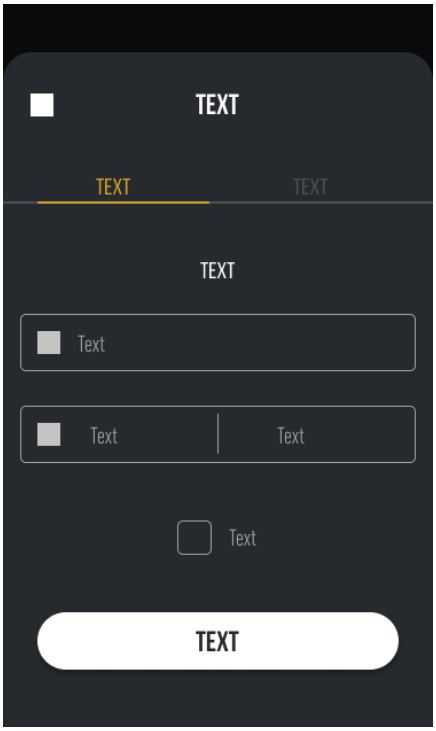


Рисунок 4.5 – Розмежування модального вікна створення задач на області та додання функціональних елементів, шаблону тексту та іконок

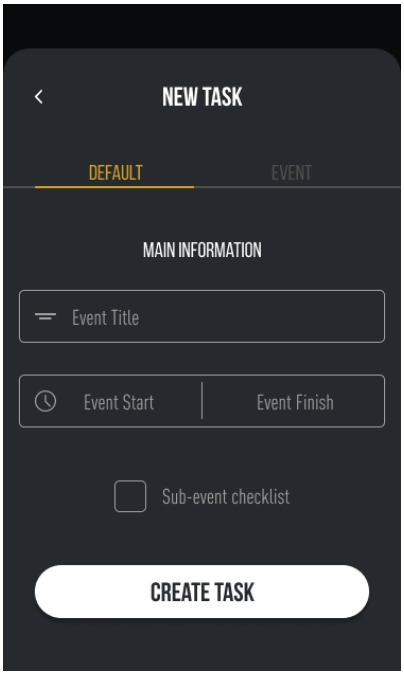


Рисунок 4.6 – Фінальний вигляд модального вікна створення задач

Далі слідує програмна реалізація сторінки планування згідно з функціональними вимогами. Основні файли, де реалізована логіка та функціональний принцип роботи модуля планування, представлені в табл. 4.1.

Таблиця 4.1. Основні файли реалізації модуля планування

Назва файлу	**Функції**
TimelineView.tsx (компонент з таймлайном)	Являє собою компонент, який включає в себе всі суб-компоненти та методи, необхідні для відображення тайлмайну. Включає в себе: - компонент timeline; - модальне вікно зі створенням нової задачі.
taskReducer.ts (reducer для задач)	Являє собою частину комбінованого reducer'а. Відповідає за збереження даних задач в глобальному сторі (redux-state).
taskSagas.ts (саги для задач)	Являє собою серверним реалізатором запитів створення та отримання інформації про задачі.
taskController.ts (API конроллер для задач)	Являє собою набір методів для обробки запитів для роботи з задачами.

Реалізація уявної прямої часу, взаємодія з часом та створення елементів таймлайну. Файл TimelineView.tsx.

В даному файлі імпортуються:

- Змінні format, addDays, isToday, які використовуються у функціях для роботи з датами;
- Функція “Darken” для відображення статусу задач;
- Компоненти інтерфейсу сторінки з пакету react-native;
- Функція styled для стилізації елементів сторінки планування;
- Пакет react-native-svg-transformer для роботи з зображеннями SVG-формату;
- Основний компонент timeline, який включає в себе інтерактивну зону, де розміщені задачі;
- Функції use для створення подій, що будуть оброблені reducer`ом для взаємодії з хранилищем даних.

Реалізовано запит useDispatch, який повертає функцію, за допомогою якої можна створювати події для redux-store. Також іде ініціалізація змінної timerRef, яка буде використана для збереження часового інтервалу між задачами. Також використовується запит useSelector, який повертає дані про задачу з task reducer’a. При зміні значення в redux-store дана змінна також буде оновлена, що спричинить оновлення компоненту.

Створено та визначено роботу з часом та датами за допомогою задання змінних curenttime, formattedDate, timeRef, selectedDay. Усі вони формують уявну лінію часу, на яких буде розміщено задачі з певною тривалістю, часом початку та кінця. Було створено функцію, яка відповідає за оновлення часу. Дана функція, при ініціалізації, створює таймер, який спрацьовує кожну хвилину та оновлює теперішній час. При зміні сторінки, перед знищенням компоненту (componentWillUnmount), спрацьовує функція, яка видаляє інтервал.

Також реалізовано такі компоненти відображення процесів модулю планування:

- ControlledDataContainer – відображає теперіщній час на таймлайні;

- Timeline – є основним компонентом сторінки планування, відображає саму робочу область таймлайну, де розміщені задачі;
- AddEventContainer – стилізована кнопка створення задачі;
- TaskModal – модальне вікно для створення задачі;
- ChangeViewControlledContainer – компонент відображення статусу задачі за допомогою альфа каналу (параметру непрозорості елементу)

Отримання функціональних даних про задачі від сховища. Файл taskReducer.ts.

Даний файл має 6 експортів:

- STATE_KEY – константа, яка зберігає State;
- State - інтерфейс, який буде використаний при декларації глобального типу всього reduxStore (IRootState);
- getTasks – функція для отримання списку всіх створених задач;
- getDateTasks – функція для отримання задач, по конкретному дню. Дана функція вміщує в собі параметр, а саме дату, в залежності від якої і повертаються задачі певної дати;
- getSelectedDay – функція для отримання певного обраного користувачем дня з task reducer'a;
- reducer – експортується за замовчуванням і являє собою регулятор, в якому в залежності від типу події модифікуються параметри задачі.

В даному файлі налаштовано функції, які отримують певний набір даних в залежності від дій користувача. Запит на отримання інформації йде не на пряму від користувача, а від додатку, що потребує її для подальшого функціонування.

Запис та запит даних задач. Файл taskSagas.ts.

Даний файл містить основну функцію-генератор taskSagas, яка за допомогою методу ALL комбінує всі запити, що стосуються створення та отримання задач та їх параметрів. Дана функція експортується за замочуванням, та буде підключена та

запущена за допомогою методу fork в кореневому файлі rootSagas.ts. Сама функція реалізовує 2 процеси:

- createTaskSaga – функція-генератор, яка при події CREATE_TASK, відправить POST запит на API зі створенням задачі, а потім, в разі успішного виконання, оновить redux-state;
- fetchTasksSaga – функція-генератор, яка відправить GET запит на API з отриманням задач по конкретному дню, а потім оновить redux-state.

Обробка запитів. Файл taskController.ts.

Даний конролер необхідний для оброблення запитів з оновлення задач. Він обслуговує тільки модель Task та формує запити в рамках однієї задачі, не змінюючи значення внутрішнього параметру або не керуючи одразу списком задач.

З допомогою команди import, в файл імпортовано основні моделі, що стосуються модулю планування:

- Task – модель задачі з усіма його параметрами;
- SubTask – модель суб-задач, які прив’язанні до однієї конкретної задачі;
- TaskTypes – тип задачі, від якої може змінюватись набір параметрів задачі;
- TUser – модель користувача, до якого прив’язаний його список створених задач.

В файлі описано клас TaskController, що використовує наступні методи:

- fetchTasks – даний метод повертає всі задачі, які належать користувачу. Користувач доступний за рахунок використання passport middleware, яка додає до контексту парамметрів користувача. Також в разі, якщо присутній параметр date, то задачі будуть повернуті лише для конкретного дня, який вказаний в як значення дати;
- fetchTask – даний метод повертає одну задачу з субзадачами, базуючись на параметрі taskId, який задекларований в routes задач;
- createTask – даний метод створює задачу, на основі даних, які приходять в тілі запиту, при цьому в разі, якщо користувач, зазначений в задачі, не

співпадає з авторизованим користувачем, то задача не буде створена і буде повернуте повідомлення як помилка запиту.

- updateTask – даний метод оновлює певну задачу, базуючись на даних в тілі запиту;
- deleteTask – даний метод видаляє задачу, базуючись на її ідентифікаторі та приналежності до авторизованого користувача.

Повний список файлів та коду програмної реалізації модуля планування наведено в додатку Б.

4.2 Спринт розробки модуля профілю користувача

Профіль користувача вміщує в себе одну функціональну сторінку з персональними параметрами користувача. Їх можна переглядати та змінювати деякі з них.

Спочатку розмежуємо сторінку профілю на області, на яких буде розміщуватися функціональна інформація інтерфейсу та контент сторінки.

Розмежування дає нам змогу заповнити робочу область функціональним текстом та контентом, а також позначити місця для зображень-іконок.

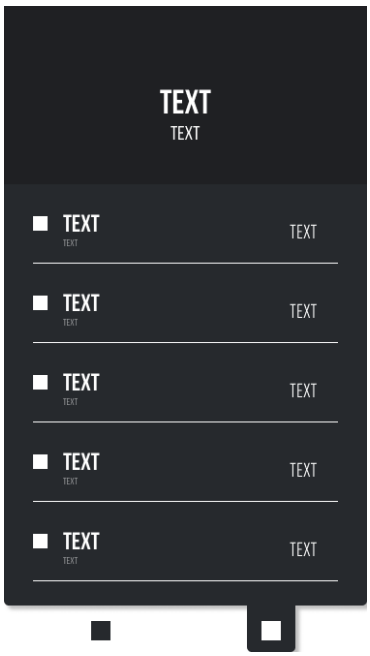


Рисунок 4.7 – Додаткові елементи, шаблон тексту та іконок

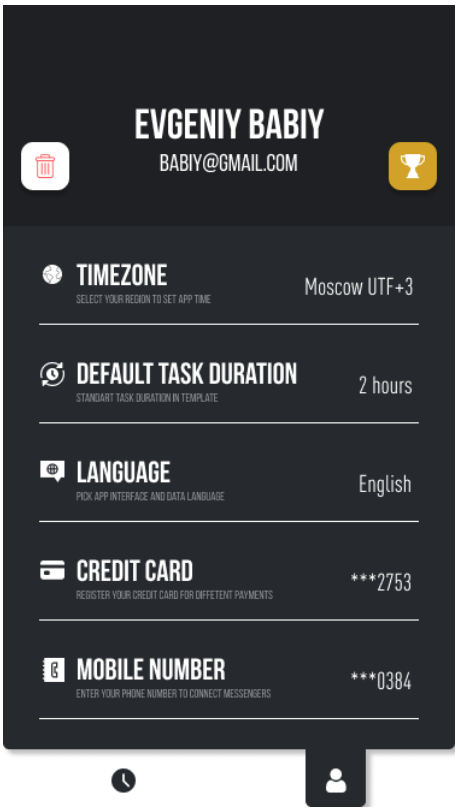


Рисунок 4.8 – Фінальний вигляд сторінки профілю

Основні файли, де реалізована логіка та функціональний принцип роботи модуля профіля користувача представлені в табл. 4.2.

Таблиця 4.2. Основні файли реалізації модуля ідентифікації користувача

Назва файлу	Функції
userProfileView.tsx (компонент профілю користувача)	Являє собою компонент, який включає в себе суб-компоненти, для роботи з профілем. Має масив з конфігурацією елементів форми.
userReducer.ts (reducer користувача)	Являє собою частину комбінованого reducer'а. Відповідає за збереження даних користувача в глобальному сторі.
userController.ts (API конроллер користувача)	Являє собою набір методів, для обробки запитів з оновлення даних користувача.
user.model.ts (модель бази даних)	Являє собою sequelize-typescript модель користувача. Містить в собі набір параметрів та атрибутів, які повинні бути у авторизованого користувача.

Реалізація профілю користувача. Файл userProfileView.tsx.

Даний файл необхідний для відображення профілю користувача. Містить змінну items, яка являє собою об'єктний масив з елементами форми. Містить 7 елементів:

- елемент ім'я – імя авторизованого користувача;
- елемент адреси – почтова адреса користувача при авторизації;
- елемент таймзони – для визначення часового поясу користувача;
- елемент вибору тривалості задачі за замовчуванням – мінімальний час задачі;
- елемент мультимовності – вибору мови інтерфейсу користувача;
- елемент з платіжною інформацією – інформація про кредитну картку користувача;
- елемент з інформацією про мобільний номер – мобільний номер користувача.

Кожен елемент має свій окремий компонент, та назву. При оновленні інформації, за допомогою методу масиву MAP відбувається ітерація та реформування елементів форми.

Отримання функціональних даних про користувача від сховища. Файл userProfileView.tsx.

Містить у собі 5 експортів:

- STATE_KEY - константа, яка зберігає частини state;
- STATE – interface, який буде використаний при декларації глобального типу всього reduxStore (IRootState);
- getUser – функція для отримання даних користувача;
- getCompletedAchievements – функція для отримання ідентифікаторів виконаних досягнень по користувачу;
- reducer - експортується за замовчуванням, являє собою регулятор, в якому в залежності від типу події модифікується user.

Інтерфейс State користувача містить в собі набір параметрів, які слугують для шаблону заповнення інформації при декларації типу користувача reduxStore. Складається з наступних елементів:

- Id – унікальний ідентифікаційний номер користувача;
- Email – авторизована електронна почта користувача;
- Credits – ігрова валюта користувача;
- LastCheckedVersion – остання встановлена версія;
- stribeSubscriptionId – ідентифікаційний номер підписки;
- timezone – часовий пояс;
- achievementIds – список ідентифікаторів досягнень користувача;
- CreatedAt – дата створення користувача;
- UpdatedAt - остання активність користувача.

Обробка запитів. Файл userController.tsx

Даний конролер необхідний для оброблення запитів з оновлення задач. Він містить в собі клас UserController, в якому описані наступні методи управління користувачем:

- fetchUsers – даний метод повертає всіх користувачів;

- fetchUser – даний метод повертає одного користувача, базуючись на параметрі userId, який задекларований в route. При цьому повертається список параметрів користувача, в якого поле achievementsId являє собою масив із ідентифікаторів;
- updateUser – даний метод оновлює користувача, базуючись на даних тіла запиту. При цьому даний метод перевіряє чи авторизований користувач, чи співпадають ідентифікатори авторизованого користувача та користувача, якого планується оновити;

Властивості користувача. Файл user.model.ts

Даний файл має 2 імпорти:

- IUser – являє собою interface, який буде використовуватись і різних частинах API;
- клас User – являє собою клас в якому заходятся властивості (properties) й декоратори, які модифікують поля моделі. Даний клас являє собою модель користувача, яка буде використана, в конфігурації sequelize-typescript.

Модель має поля користувача, до яких застосовані певні властивості:

- id – має декоратори AutoIncrement, PrimaryKey, Column, тип string;
- name - має декоратор Column, тип string;
- email - має декоратор Column, тип string;
- credits - має декоратор Column, тип string | number;
- timezone - має декоратор Column, тип string;
- firstWeekDay має декоратор Column, тип string;
- googleUserId - має декоратор Column, тип string;
- stripeCustomerId - має декоратор Column, тип string | null;
- stripeSubscriptionId - має декоратор Column, тип string;
- lastCheckedVersion - має декоратор Column, тип string | null;
- createdAt - має декоратор Column, CreatedAt, тип Date;
- updatedAt - має декоратори Column, UpdatedAt, тип Date.

Також дана модель має зв’язок HasMany (одна сутність може мати зв'язок з декількома) в базі даних з моделями Task та Achievements.

Повний список файлів та коду програмної реалізації модуля ідентифікації користувача наведено в додатку Б.

В результаті реалізації було створено два функціональні модулі – модуль планування та ідентифікації.

Планування відбувається шляхом створення задач з параметрами та відображення їх на таймлайні.

Ідентифікація користувача відбувається шляхом авторизації, відображення особистого профілю та можливість задання його параметрів.

Потік даних модулів керується з допомогою запитів. Усі дані про користувачів та їх задачі зберігаються в сховищі redux store.

ВИСНОВКИ

В результаті дипломної роботи було виконано проєкт розробки модулів планування та ідентифікації користувача для мобільного додатку “PlannIT”.

Проведено аналіз предметної області, де було визначено основні поняття, сучасні тренди сфери та розглянуто декілька найпопулярніших додатків-аналогів. Проаналізувавши їх за критеріями, було сформовано нефункціональні вимоги до характеристик майбутніх модулів додатку.

На етапі постановки задачі описано головну мету проєкту та задачі, які необхідно виконати для досягнення поставленої мети. Сформульовані функціональні вимоги до модулів мобільного додатку.

Після формування вимог було проаналізовано методи створення мобільних додатків та обраний інструментарій для виконання поставлених задач. Були обрані технології Java Script, а саме фреймворк React Native та Node.JS як сучасні та продуктивні засоби для реалізації мобільних додатків.

Проведено структурно-функціональне моделювання проєкту, розроблено діаграму IDEF0 та її декомпозиції, діаграми варіантів використання, послідовностей. Також наведено ER-діаграму бази даних, яка зберігає необхідні дані для роботи модулів планування та ідентифікації.

Розроблено ескізи інтерфейсу користувача для майбутнього додатку з урахуванням вимог та сучасних тенденцій UI/UX.

Керуючись обраною методологією спринтів та використовуючи обрані інструменти, у редакторі програмного коду було реалізовано модулі мобільного додатку-планувальника «PlanIT». Модуль планування вміщує в себе відображення створених задач, створення, редагування та видалення задач додатку. Модуль ідентифікації користувача містить авторизацію та персональний профіль з набором даних.

Практичне значення розроблених модулів полягає у тому, що вони є необхідними функціональними частинами для комерційного мобільного додатку “PlannIT”, призначеного для самоорганізації та планування власної роботи користувачами.

СПИСОК ДЖЕРЕЛ

1. Визначення процесу планування: – [Електронний ресурс] URL: https://www.yourarticlelibrary.com/management/planning-management/what-is-planning/99766
2. USwitch. Full story about mobile phone development. – [Електронний ресурс] URL: https://www.uswitch.com/mobiles/guides/history-of-mobile-phones/#:~:text=Mobile%20phones%20were%20invented%20as,really%20mobile%20phones%20at%20all.&text=Motorola%2C%20on%203%20April%201973,the%20first%20handheld%20mobile%20phone.
3. IPkey. Що таке мобільний додаток. – [Електронний ресурс] URL: http://ipkey.com.ua/uk/faq/984-application.html.
4. QAinfo. Мобільний додаток – вносимо розуміння у значення терміну – [Електронний ресурс] URL: https://www.quality-assurance-group.com/mobilnyj-dodatok-vnosymo-rozuminnya-u-znachennya-terminu/
5. ОсвітаUA. Менеджмент. – [Електронний ресурс] URL: https://ru.osvita.ua/vnz/reports/management/15423/
6. Koloro. Тайм-менеджмент: принципи управління своїм часом – [Електронний ресурс] URL: https://koloro.ua/ua/blog/menedzhment/tajm-menedzhment-principy-upravleniya-svoim-vremenem.html
7. MFlow. Що таке діаграма Ганта – [Електронний ресурс] URL: http://monetary-flow.com/shto-take-dagrama-ganta-ta-yak-vona-vikoristovutysya-u-bznes-planuvann/
8. InvoDigital. Топ-100 популярних додатків світу. – [Електронний ресурс] URL: https://invodigital.com/blog/article/1416-top-100-samykh-populyarnykh-mobilnykh-prilozhenij-v-mire
9. Gagadget. App Store & Google Play. – [Електронний ресурс] URL: https://gagadget.com/17030-google-play-ili-app-store/#:~:text=%D0%A3

%20Google%20%E2%80%93%20%D1%8D%D1%82%D0%BE%20Google%20Play,%2C%20%D0%BA%D0%BD%D0%B8%D0%B3%D0%B8%2C%20%D1%84%D0%B8%D0%BB%D1%8C%D0%BC%D1%8B%20%D0%B8%20%D1%81%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D1%8B.

10. Android Mobile Review. Характеристика та огляд додатку Any.DO – [Електронний ресурс] URL: http://android.mobile-review.com/market/6732/
11. Zoom. Кращі додатки-планувальники на мобільні пристрої. – [Електронний ресурс] URL: https://zoom.cnews.ru/publication/item/62452
12. Remember the milk. Офіційна веб-сторінка. – [Електронний ресурс] URL: https://www.rememberthemilk.com/
13. PCMAG. Trello app review – [Електронний ресурс] URL: https://www.pcmag.com/reviews/trello
14. Live Business. Google Calendar – [Електронний ресурс] URL: https://www.livebusiness.ru/tool/337/
15. PDM та PLM системи – [Електронний ресурс] Режим доступу: http://asapcg.com/press-center/articles/pdm-i-plm-sistemy/
16. Ініціалізація проекту, його обґрунтування та планування – [Електронний ресурс] Режим доступу: https://studfiles.net/preview/5483482/
17. React Native learning book. – [Електронний ресурс] URL: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html IPkey.
18. W3schools. Node.JS introduction. – [Електронний ресурс] URL: https://www.w3schools.com/nodejs/nodejs_intro.asp
19. Node.JS. Трендовий інструмент 2019 року. – [Електронний ресурс] URL: http://suhorukov.com/news_akademy/nodejs-samye-trendovye-instrumenty-na-2019-god#:~:text=%D0%92%202018%20%D0%B3%D0%BE%D0%B4%D1%83%20%D0%BF%D0%BE%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC,%D1%82%D0%BE%D0%BF%2D4%20%D0%B2%20%D1%81%D0%B2%D0%BE%D0%B5%D0%B9%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8.

20. ScienceDirect. Automation of strategy using – [Електронний ресурс] URL: https://www.sciencedirect.com/science/article/pii/S2214716015000111
21. KPI. Проектування програмного забезпечення засобами UML: – [Електронний ресурс] URL: http://mmsa.kpi.ua/sites/default/files/disciplines/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0%20%D1%96%20%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC/didkovska_m_v_testing_lecture_3.pdf
22. Indeed. Career Guide. What is a User Interface? Definition, Types and User Interface Example : – [Електронний ресурс] URL: https://www.indeed.com/career-advice/career-development/user-interface#:~: text=The%20user%20interface%20(UI) %20is,to %20receive%20maximum %20desired%20outcome.
23. UkrMort. Ескіз проти малювання: – [Електронний ресурс] URL: https://uk.mort-sure.com/blog/difference-between-sketching-and-drawing/
24. Наукове мислення: Підходи до побудови ієрархічної структури робіт проекту (wbs) та обґрунтування вибору на прикладі проекту створення сокового бару "dnipro & the juice" – [Електронний ресурс] URL: http://naukam.triada.in.ua/index.php/konferentsiji/45-p-yatnadtsyata-vseukrajinska-praktichno-piznavalna-internet-konferentsiya/306-pidkhodi-do-pobudovi-ierarkhichnoji-strukturi-robit-proektu-wbs-ta-obgruntuvannya-viboru-na-prikladi-proektu-stvorennya-sokovogo-baru-dnipro-the-juice
25. Структуризація проекту: OBS – [Електронний ресурс] URL: https://www.oa.edu.ua/download/Lektsija_5.PDF
26. Buklib. Поєднання структур проєкту – [Електронний ресурс] URL: https://buklib.net/books/24519/
27. ItWeek. PDM-система – [Електронний ресурс] URL: https://www.itweek.ru/industrial/article/detail.php?ID=59500
28. APM. Gantt Chart – [Електронний ресурс] URL: https://www.apm.org.uk/resources/find-a-resource/gantt-chart/

29. Бізнес-планування, сутність і поняття ризиків. – [Електронний ресурс] URL: https://library.if.ua/book/19/1573.html#:~:text=%D0%A0%D0%B8 %D0%B7%D0%B8%D0%BA%20%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%20%E2%80%94%20%D1%86%D0%B5%20%D1%81%D1%82%D1%83%D0%BF%D1%96%D0%BD%D1%8C%20%D0%BD%D0%B5%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B8,%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%20%D0%BD%D0%B5%D1%81%D0%BF%D1%80%D0%B8%D1%8F%D1%82%D0%BB%D0%B8%D0%B2%D0%B8%D1%85%20%D1%81%D0%B8%D1%82%D1%83%D0%B0%D1%86%D1%96%D0%B9%20%D1%96%20%D0%BD%D0%B0%D1%81%D0%BB%D1%96%D0%B4%D0%BA%D1%96%D0%B2.
30. Оцінка ризиків проєкту – [Електронний ресурс] URL: http://www.ukr.vipreshebnik.ru/upr-rizik/1220-otsinka-rizikiv-proektu.html
31. Навчальні матеріали онлайн. Характеристика проектних ризиків. – [Електронний ресурс] URL: https://pidru4niki.com/12560607/investuvannya/harakteristika_proektnih_rizikiv

ДОДАТОК А. ПЛАНУВАННЯ РОБІТ

1 Ідентифікація мети ІТ-проекту

Деталізація мети методом SMART. Продуктом дипломного проекту є реалізовані функціональні модулі додатку-планувальника. Результати деталізації методом SMART розміщені у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Створити функціональні модулі мобільного додатку-планувальника.
Measurable (вимірювана)	Результатом роботи проекту є оцінка замовника.
Achievable (досяжна)	Розробка модулів здійснюється за допомогою фреймворків React Native та Node.JS.
Relevant (реалістична)	У наявності є всі необхідні технічні та програмні засоби. Розробники достатньо кваліфіковані для виконання поставлених задач.
Time-framed (обмежена у часі)	Ціль має часове обмеження. Робота повинна бути виконана у терміни, що були оговорені замовником проекту. Проект повинен бути виконаний згідно з календарним планом.

2 Планування змісту структури робіт ІТ-проекту

WBS проекту (вона ж Work Breakdown Structure або ICP, Ієрархічна Структура Робіт) - це розбиття проекту на конкретні результати, які повинні бути досягнуті для досягнення цілей проекту. Важливо розуміти, що в WBS збираються саме результати робіт, а не завдання, які потрібно виконати для отримання цих результатів. WBS є ієрархічною та інкрементною декомпозицією проекту у фази, кінцеві результати та пакети робіт. Вона є ієрархічною структурою, що показує подальший розподіл необхідних для виконання мети зусиль; наприклад, програма, проект чи договір [24].

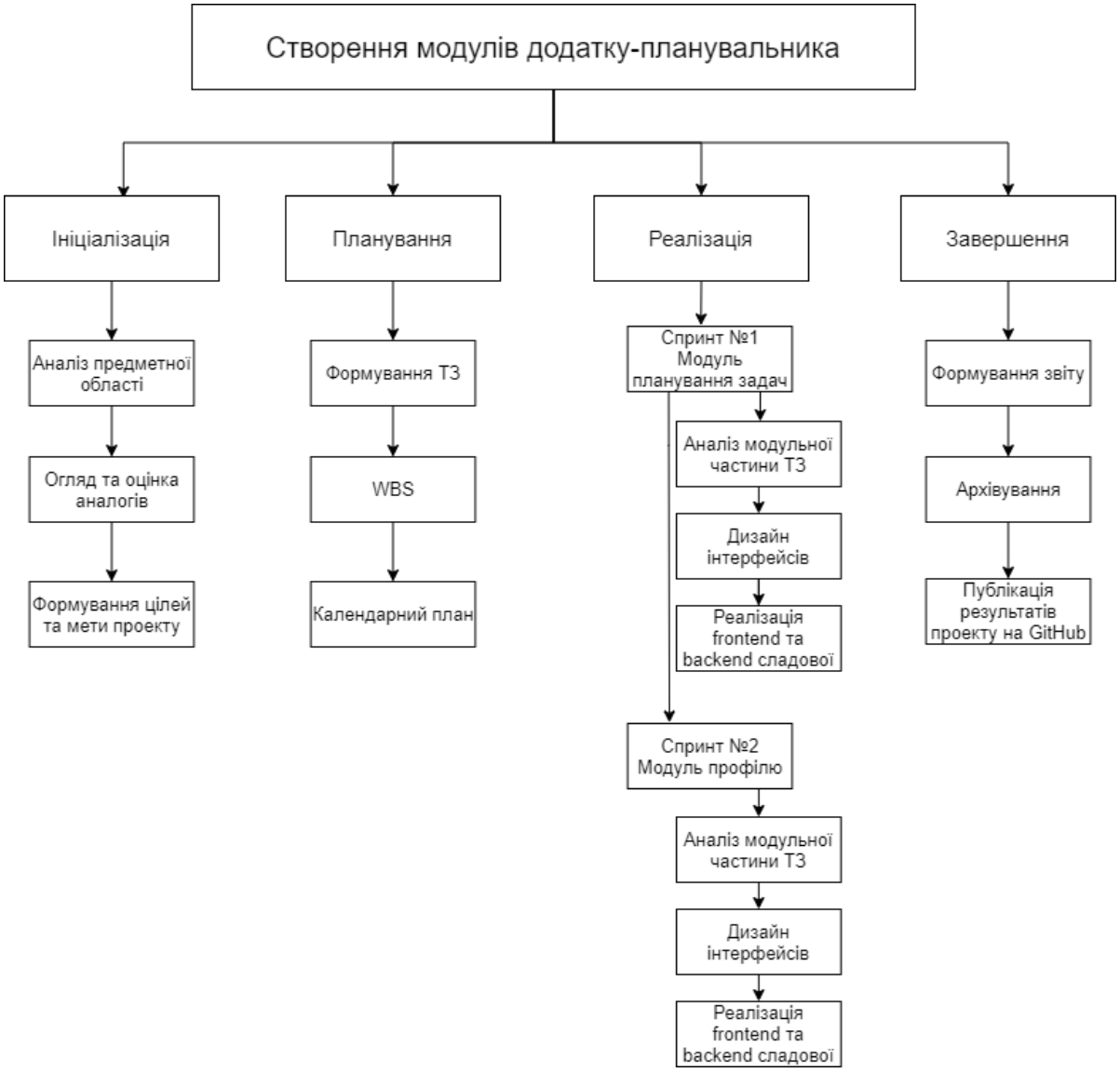


Рисунок А.1 - WBS-структура етап ініціалізації проекту

Ця структура стосується тільки внутрішньої організаційної структури проекту і не зачіпає відносин проектних груп чи учасників з батьківськими організаціями. Будується OBS аналогічно робочій структурі, а саме:

– на першому рівні відображається організаційна структура як єдиний елемент;

– на другому і нижчих рівнях триває поділ структури на основні організаційні елементи.

Цей процес повторюється до найнижчого рівня – базових робочих груп (змішаних цільових або функціональних), а при реалізації малих проектів – до окремих виконавців [25].

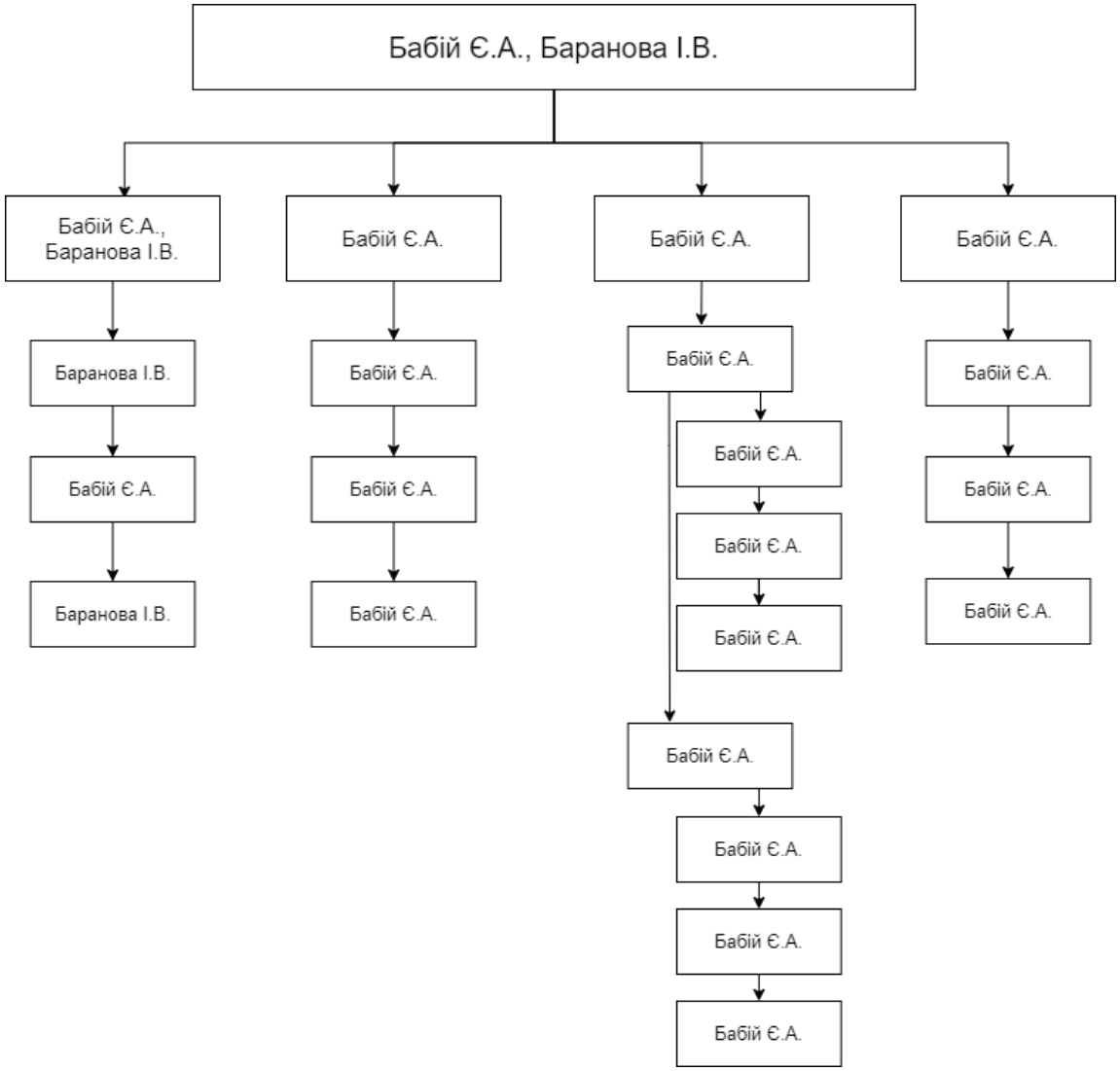


Рисунок А.2 - OBS-структура етапу ініціалізації проекту

На підставі OBS та WBS структур будується матриця відповідальності проекту. Матриця відповідальності закріплює за кожною елементарною роботою виконавця. Використовується для контролю відповідності розподілу ролей цілям проекту.

На верхньому рівні розподіляються ролі та відповідальність по елементах OBS, на нижньому – фази проекту [26]. Приклад матриці відповідальності наведений у табл. А.2.

Таблиця А.2 – Матриця відповідальності

Фази	Виконавці	
	Бабій Є.А.	Баранова І.В.
Аналіз предметної області		✓
Огляд та оцінка аналогів	✓	
Формування цілей та мети проекту		✓
Формування ТЗ		✓
WBS-структура	✓	
Календарний план проекту	✓	
Спринт №1	✓	
Спринт №2	✓	
Формування звіту	✓	
Архівування	✓	
Публікація результатів на GitHub	✓	

PDM-система (англ. Product Data Management - система управління даними про виріб) - організаційно-технічна система, що забезпечує управління всією

інформацією про виріб. При цьому, вироби можуть розглядатися як різні складні технічні об'єкти [27].

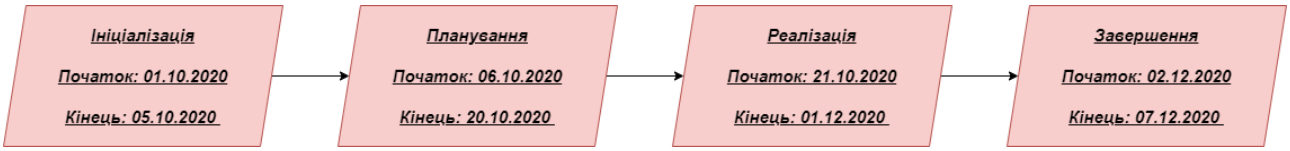


Рисунок А.3- PDM-мережа проекту у згорнутому до основних фаз вигляді

3 Побудова календарного графіку виконання ІТ - проекту

Діаграма Ганта (Gantt chart, стрічкова діаграма, графік Ганта) - це популярний тип стовпчастих діаграм, який використовується для ілюстрації плану, графіка робіт проекту. Є одним з методів планування проектів [28].

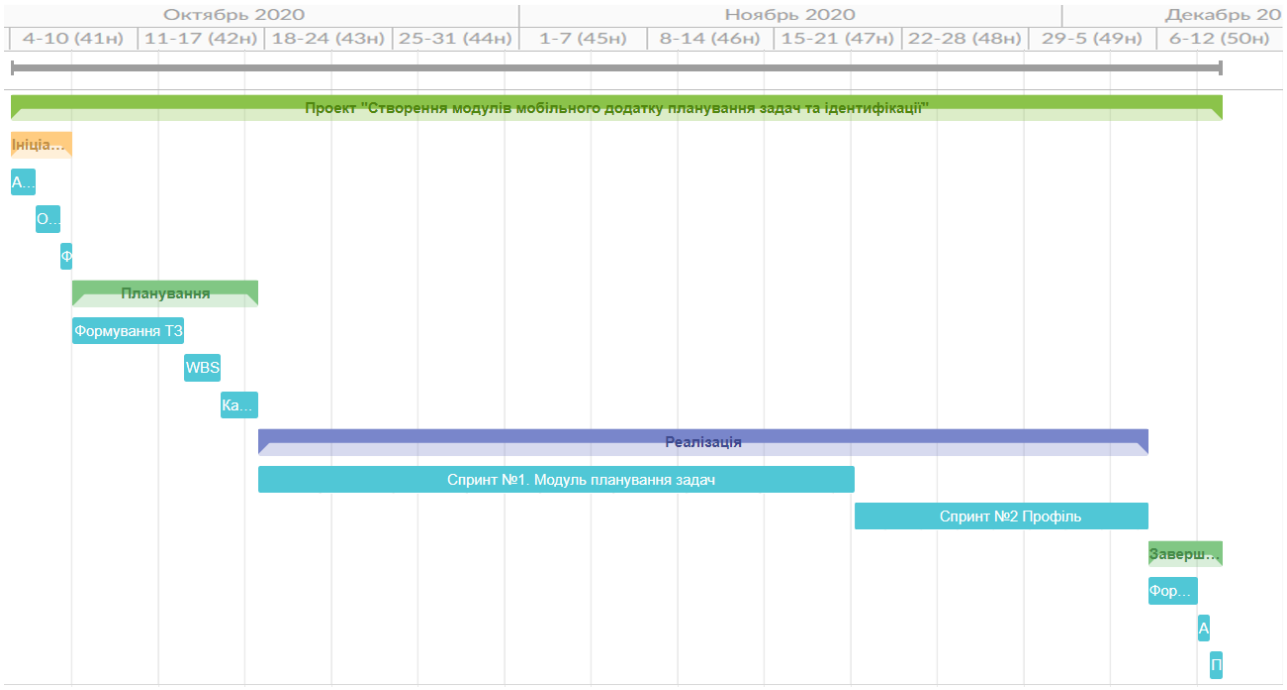


Рисунок А.4 - Діаграма Ганта проекту в розгорнутому вигляді

Задача	Начало	Завершение	Длительность
	01.10.2020	07.12.2020	67д
Проект "Створення модулів мобільного додатку планування...	01.10.2020	07.10.2020	67д
Ініціалізація	01.10.2020	05.10.2020	5д
Аналіз пред. обл.	01.10.2020	02.10.2020	2д
Огляд та оцінка аналогів	03.10.2020	04.10.2020	2д
Формування цілі та мети	05.10.2020	05.10.2020	1д
Планування	06.10.2020	20.10.2020	15д
Формування ТЗ	06.10.2020	14.10.2020	9д
WBS	15.10.2020	17.10.2020	3д
Календарний план	18.10.2020	20.10.2020	3д
Реалізація	21.10.2020	01.12.2020	40д
Спринт №1. Модуль планування задач	21.10.2020	16.11.2020	26д
Спринт №2 Профіль	17.11.2020	01.12.2020	14д
Завершення	02.12.2020	07.12.2020	6д
Формування звіту	02.12.2020	05.12.2020	4д
Архівація	06.12.2020	06.12.2020	1д
Публікація на GitHub	07.12.2020	07.12.2020	1д

Рисунок А.5 − Список робіт для побудови діаграми Ганта

4 Управління ризиками проекту

Ризик – це ймовірнісна подія, яка у випадку своєї появи негативно або позитивно впливає на проект. Управління ризиком – це процес реагування на події та зміни ризиків у процесі виконання проекту [29].

Моніторинг ризиків включає контроль ризиків протягом усього життєвого циклу проекту. Якісний моніторинг ризиків забезпечує управління інформацією, яка допомагає приймати ефективні рішення до настання ризикових подій.

Найбільш розповсюдженою характеристикою ризику є загроза або небезпека виникнення невдач у тій чи іншій діяльності, небезпека виникнення несприятливих

наслідків, змін зовнішнього середовища, які можуть викликати втрати ресурсів, збитки, а також небезпеку, від якої слід застрахуватися [30].

Процес управління ризиками включає в себе такі пункти:

1) Ідентифікація ризиків (виявлення ризиків);
2) Оцінювання ризиків (оцінка ймовірності та впливу);
3) Заходи реагування на ризики;
4) Моніторинг ризиків.

В процесі аналізу для визначення числових значень ймовірності виникнення ступеня впливу, зазвичай застосовується метод експертних оцінок.

На їх основі визначається ранг ризику, як потенційний вплив ризику на проект, який оцінюється як добуток ймовірності виникнення та ступеню впливу [31].

Шкала оцінки ризику може відповідати емпіричній шкалі оцінки ризику:

5 балів - критичний ризик (0,81 - 1);

4 бали - максимальний ризик (0,61 - 0,8);

3 бали - високий ризик (0,41 - 0,6);

2 бали - нормальний ризик (0,31 - 0,4);

1 бал - малий ризик (0 - 0,3).

де R = P * L, де R – ранг ризику; P – ймовірність виникнення; L – ступінь впливу.

Розглянуті ризики проєкту з оцінкою рангу ризику наведено в таблиці А.3

Аналізуючи ризики за ймовірністю виникнення та впливу, можемо їх віднести до тих, що мають низький ранг ризику. Усі продемонстровані ризики проекту – незначні та виправдані.

Таблиця А.3 – Ризики проекту

№	Об’єкт ризику	Ризик	P	L	R
1.	Час	Недотримання виконавцем та замовником календарних строків проекту	0,1	0,5	0,05
2.	Час	Зміна пріоритету проекту замовником	0,1	0,3	0,03
3.	Технологія	Недостовірна інформація про характеристики базового програмно-апаратного комплексу замовника або його значуща зміна в ході реалізації проекту	0,1	0,8	0,08
4.	Якість	Невідповідність системи задачам , грубі помилки в алгоритмах процесів, критичні збої системи	0,1	0,9	0,09
5.	Бюджет	Виникнення незапланованих робіт по проекту	0,4	0,2	0,08
6.	Трудові ресурси та їх кваліфікація	Неможливість участі в запланованих роботах по проекту необхідних співробітників зі сторони замовника та виконавця у зв’язку з відпусткою, відрядженням та ін.	0,1	0,4	0,04
7.	Інтеграція	Недостовірна інформація про зовнішні системи, з яких передбачена взаємодія в рамках проекту	0,2	0,5	0,1
8.	Ринок	Розширення функціональних характеристик програмних продуктів, що вже використовуються замовником в рамках цілей проекту	0,5	0,5	0,25
9.	Ринок	Зміна вимог замовника, економічні зміни, посилення конкуренції, втрата позицій на ринку;	0,3	0,3	0,09
10.	Бюджет	Зриви планів робіт; невірна стратегію;некваліфікований персонал; переплати за роботу/матеріали; неузгодження частин проекту; невірний кошторис.	0,1	0,8	0,08

ДОДАТОК Б.
ЛІСТИНГ КОДУ

Авторизація. App.tsx

```
import { WEB_CLIENT_ID } from '@env';
import React from 'react';
import { PersistGate } from 'redux-persist/integration/react';
import 'react-native-gesture-handler';
import { GoogleSignin } from 'react-native-google-signin/index';
import { Provider } from 'react-redux';
import { ThemeProvider } from 'styled-components/native';

// initialize localization
import '~/localization/i18n';
import Main from '~/Main';
import configureStore from '~/store/configureStore';
import theme from '~/theme/theme';
const { store, persistor } = configureStore();
GoogleSignin.configure({
  webClientId: WEB_CLIENT_ID,
});
const App = () => (
  <ThemeProvider theme={theme}>
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
        <Main />
      </PersistGate>
    </Provider>
  </ThemeProvider>
);
export default App;
```

Роути. AppRoutes.tsx

```
const TimelineStackNavigation = () => (
  <TimelineStack.Navigator initialRouteName="Timeline">
    <TimelineStack.Screen
      options={{
        header: (props: StackHeaderProps) => <Header {...props} />,
      }}
      name="Timeline"
      component={TimelineView}
    />
  </TimelineStack.Navigator>
);

const UserProfileStackNavigation = () => (
```

```
  <UserProfileStack.Navigator initialRouteName="UserProfile">
    <UserProfileStack.Screen
      options={{
        header: (props: StackHeaderProps) => <Header {...props} />,
      }}
      name="UserProfile"
      component={UserProfileView}
    />
  </UserProfileStack.Navigator>
);
const ScreensWithTab = () => (
  <AppTab.Navigator tabBar={(props) => <CustomTabBar {...props} />}>
    <AppTab.Screen name="TimelineTab" component={TimelineStackNavigation} />
    <AppTab.Screen name="ProfileTab" component={UserProfileStackNavigation} />
  </AppTab.Navigator>
);
export const MainAppNavigator = (
  <DefaultStack.Navigator
    screenOptions={{
      headerShown: false,
    }}>
    <AppTab.Screen name="MainScreens" component={ScreensWithTab} />
    <DefaultStack.Screen name="Achievements" component={AchievementsView} />
  </DefaultStack.Navigator>
);
export const AppRouter = ({
  children,
}: {
  children: JSX.Element | JSX.Element[];
}) => <NavigationContainer>{children}</NavigationContainer>;
```

Сторінка входу. LoginView.tsx

```
const LoginView = () => {
  const { signIn, signOut } = useAuthHook();
  return (
    <LoginContainer>
      <GoogleLogo />
      <SignInButton
        onPress={signIn}
        style={{
          shadowColor: '#000',
          shadowOffset: {
            width: 0,
            height: 7,
          },
          shadowOpacity: 0.8,
          shadowRadius: 5.51,
          elevation: 15,
```

```
        }}>
        <Description>Sign In</Description>
      </SignInButton>
      <SignOutButton
        onPress={signOut}
        style={{
          shadowColor: '#000',
          shadowOffset: {
            width: 0,
            height: 7,
          },
          shadowOpacity: 0.8,
          shadowRadius: 5.51,
          elevation: 15,
        }}>
        <Description>Sign out (dev only)</Description>
      </SignOutButton>
    </LoginContainer>
  );
};
const LoginContainer = styled.View`
  flex: 1;
  align-items: center;
  justify-content: center;
  background-color: ${THEME.colors.appBackground};
`;
const SignInButton = styled.TouchableOpacity`
```

Робота с авторизацією. authHook.tsx

```
import { useCallback } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { logout, signInAction } from '~/store/actions/authActions';
export const useAuthHook = () => {
  const dispatch = useDispatch();
  const touchId = useSelector((state: RootState) => state.auth.touchId);
  const token = useSelector((state: RootState) => state.auth.token);
  const signIn = useCallback(() => dispatch(signInAction()), [dispatch]);
  const signOut = useCallback(() => dispatch(logout()), [dispatch]);
  return { token, touchId, signIn, signOut }; };
```

Reducer для зберігання даних авторизації в redux store. authReducer.tsx

```
const namespace = 'AUTH';
export const AUTH = `${namespace}/AUTH`;
export const LOGOUT = `${namespace}/LOGOUT`;
export const LOGOUT_REQUEST = `${namespace}/LOGOUT_REQUEST`;
export const SIGN_IN_GOOGLE = `${namespace}/SIGN_IN_GOOGLE`;
export const SIGN_IN_GOOGLE_SUCCESSFUL = `${namespace}/SIGN_IN_GOOGLE_SUCCESSFUL`;
export const TOUCH_ID_CHECK = `${namespace}/TOUCH_ID_CHECK`;
export const TOUCH_ID_DATA = `${namespace}/TOUCH_ID_DATA`;
```

```
export const logout = () => ({
  type: LOGOUT_REQUEST,});
```

Action для зберігання даних авторизації в redux store. authAction.tsx

```
export const signInAction = () => ({
  type: SIGN_IN_GOOGLE,
});
export const touchIdAction = () => ({
  type: TOUCH_ID_CHECK,
});
} from '~/store/actions/authActions';
export const STATE_KEY = 'auth';
export interface State {
  token: string | null;
  touchId: boolean;
}
const initialState: State = {
  token: null,
  touchId: false,
};
const AuthReducer = (
  state: State = initialState,
  action: DefaultAction<Partial<State>>,
) => {
  switch (action.type) {
    case SIGN_IN_GOOGLE_SUCCESSFUL: {
      return {
        ...state,
        token: action.payload?.token ?? null,
      };
    }
    case TOUCH_ID_DATA: {
      return {
        ...state,
        touchId: action.payload as boolean,
      };
    }
    case LOGOUT: {
      return initialState;
    }
    default: {
      return state;
    }
  }
};
export const getToken = (state: RootState) => state[STATE_KEY]?.token;
export default AuthReducer;
```

Sagas для зберігання даних авторизації в redux store. authSagas.tsx

```
import axios from '~/configs/axios';
import AsyncStorage from '@react-native-community/async-storage';
import { GoogleSignin, statusCodes } from 'react-native-google-signin';
import TouchID from 'react-native-touch-id';
import { all, call, fork, put, select, takeLatest } from 'redux-saga/effects';

import {
  LOGOUT,
  LOGOUT_REQUEST,
  SIGN_IN_GOOGLE,
  SIGN_IN_GOOGLE_SUCCESSFUL,
  TOUCH_ID_CHECK,
  TOUCH_ID_DATA,
} from '~/store/actions/authActions';
import { SET_USER } from '~/store/actions/userActions';
import { getToken } from '~/store/reducers/authReducer';

function* signInWithGoogle() {
  try {
    yield GoogleSignin.hasPlayServices();
    const userInfo = yield GoogleSignin.signIn();
    const tokenResponse = yield call(() =>
      axios.post('/auth', {
        idToken: userInfo.idToken,
      }),
    );

    if (!tokenResponse.data.token) {
      throw new Error('Authentication error');
    }
    yield put({
      type: SIGN_IN_GOOGLE_SUCCESSFUL,
      payload: tokenResponse.data,
    });
    yield put({
      type: SET_USER,
      payload: tokenResponse.data.user,
    });
    yield fork(saveToken, tokenResponse.data.token);
    axios.defaults.headers.common.Authorization = `Bearer ${tokenResponse.data.token}
```

Авторизація на серверній частині. app.tsx

```
import express from 'express';
import cors from 'cors';
import passport from 'passport';
import path from 'path';
import bearerStrategy from 'config/passport';
import dbService from 'services/database.service';
import Routes from './routes';
const environment = process.env.NODE_ENV;
```

```
const app = express();
dbService(environment, false).start();
passport.use('bearer', bearerStrategy);
app.use('/assets', express.static(path.join(__dirname, 'assets')));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
// allow cross origin requests
app.use(cors());
Routes.routes(app);
export default app;
```

Роути для авторизації. Auth.routes.tsx

```
import { Router } from 'express';
import AuthController from 'controllers/AuthController';
const authRoutes = Router();
authRoutes.post('', AuthController.auth);
authRoutes.get('/logout', AuthController.logout);
export default authRoutes;
```

Паспорт пакет для захисту даних. Passport.tsx

```
import { Strategy as BearerStrategy } from 'passport-http-bearer';
import { User } from 'database/models/user.model';
import RedisService from 'services/redis.service';

const strategy = new BearerStrategy(async (token, done) => {
  const googleId = await RedisService.client.getAsync(token);
  const user = await User.findOne({
    where: {
      googleUserId: googleId,
    },
    attributes: {
      exclude: ['googleUserId'],
    },
  });
  return done(null, user, { scope: 'all' });
});

export default strategy;
```

Контроллер авторизаційний. AuthController.tsx

```
class AuthController {
  public static async auth(
    request: Request,
    response: Response,
```

```
  ): Promise<Response<void>> {
    const { idToken, token } = request.body;

    const storedIdToken = token
      ? await RedisService.client.getAsync(token)
      : null;

    if (storedIdToken) {
      const user = await User.findOne({
        where: {
          googleUserId: storedIdToken,
        },
        attributes: {
          exclude: ['googleUserId'],
        },
        include: [
          {
            model: UserHasAchievement,
            attributes: ['achievementId'],
          },
        ],});
```

Сторінка таймлайна. TimelineView.tsx

```
const TimelineView = () => {
  const dispatch = useDispatch();

  const timerRef = useRef<number>();
  const selectedDay = useSelector(getSelectedDay);

  const [modalVisible, setModalVisible] = useState(false);
  const [currentTime, setCurrentTime] = useState(new Date());
  const [isShowCurrentLine, setIsShowCurrentLine] = useState(
    isToday(selectedDay),
  );

  const formattedDate = useMemo(() => format(selectedDay, 'dd MMMM yyyy'), [
    selectedDay,
  ]);

  const formattedCurrentTime = useMemo(() => format(currentTime, 'HH:mm'), [
    currentTime,
  ]);
  useEffect(() => {
    timerRef.current = setInterval(
      () => setCurrentTime(new Date()),
      millisecondsInMinute,
    );
    return () => clearInterval(timerRef.current);
  }, []);
  useEffect(() => {
```

```
    setIsShowCurrentLine(isToday(selectedDay));
  }, [selectedDay]);

  const onDayNavigation = useCallback(
    (isNextDay: boolean) => {
      dispatch(setSelectedDay(addDays(selectedDay, isNextDay ? 1 : -1)));
    },
    [dispatch, selectedDay],
  );
  return (
    <AppContainer>
      <ViewContainer>
        <QuickInfoContainer>
          <Title>Quick information</Title>
          <NextEventSchedule>
```

Компонент таймлайну. Timeline.tsx

```
const Timeline = ({ isShowCurrentLine, currentTime, selectedDay }: Props) => {
  const dispatch = useDispatch();
  const dayTasks = useSelector((state: RootState) =>
    getDateTasks(state, selectedDay),
  );

  useEffect(() => {
    dispatch(fetchTasks(selectedDay));
  }, [selectedDay, dispatch]);

  const isTasksAvailable = useMemo(() => dayTasks.length > 0, [
    dayTasks.length,
  ]);

  return (
    <Container>
      <ScrollView scrollEnabled={isTasksAvailable} horizontal>
        <TimelineDate selectedDay={selectedDay} />
        <ActiveZone>
          <TaskZone
            isShowCurrentLine={isShowCurrentLine}
            currentTime={currentTime}
            tasks={dayTasks}
          />
          <HoursContainer>
            {hours.map((hour: string, index: number) => (
              <Hour isLast={index === hours.length - 1} key={hour}>
                <HourText>{hour}</HourText>
              </Hour>
            ))}
          </HoursContainer>
        </ActiveZone>
```

```
      </ScrollView>
    </Container>
  );};
```

Action, reducer, safas файли таймлайну.

```
import { TaskCreate } from '~/interfaces';

const namespace = 'TASKS';

export const FETCH_TASKS = `${namespace}/FETCH_TASKS`;
export const FETCH_TASKS_SUCCESSFUL = `${namespace}/FETCH_TASKS_SUCCESSFUL`;
export const CREATE_TASK = `${namespace}/CREATE_TASK`;
export const CREATE_TASK_SUCCESSFUL = `${namespace}/CREATE_TASK_SUCCESSFUL`;
export const SET_SELECTED_DAY = `${namespace}/SET_SELECTED_DAY`;
export const fetchTasks = (date: Date) => ({
  type: FETCH_TASKS,
  payload: date,
});
export const createTask = (task: { task: TaskCreate; subTasks: string[] }) => ({
  type: CREATE_TASK,
  payload: task,
});
export const setSelectedDay = (selectedDay: Date) => ({
  type: SET_SELECTED_DAY,
  payload: selectedDay,
});
import { Task, DefaultAction } from '~/interfaces';
import {
  CREATE_TASK_SUCCESSFUL,
  FETCH_TASKS_SUCCESSFUL,
  SET_SELECTED_DAY,
} from '~/store/actions/taskActions';
export const STATE_KEY = 'tasks';

export interface State {
  selectedDay: Date;
  tasks: Record<string, Task[]>;
}

const initialState = {
  tasks: {},
  selectedDay: new Date(),
};
const TaskReducer = (
  state: State = initialState,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  action: DefaultAction<any>,
) => {
  switch (action.type) {
    case SET_SELECTED_DAY: {
```

```
      return {
        ...state,
        selectedDay: action.payload,
      };
    }
    case CREATE_TASK_SUCCESSFUL: {
      const dayKey = state.selectedDay.toDateString();
      const selectedDayTasks = state.tasks[dayKey] ?? [];
      return {
        ...state,
        tasks: {
          ...state.tasks,
          [dayKey]: selectedDayTasks.concat(action.payload),
        },
      };
    }
    case FETCH_TASKS_SUCCESSFUL:
      return { ...state, tasks: { ...state.tasks, ...action.payload } };
    default: {
      return state;
    }
  }
};
export const getTasks = (state: RootState) => state[STATE_KEY].tasks;
export const getDateTasks = (state: RootState, date: Date) =>
  state[STATE_KEY].tasks[date.toDateString()] ?? [];
export const getSelectedDay = (state: RootState) =>
  state[STATE_KEY].selectedDay;
export default TaskReducer;
import { all, call, put, takeLatest } from 'redux-saga/effects';
import { DefaultAction, TaskCreate } from '~/interfaces';
import axios from '~/configs/axios';
import {
  CREATE_TASK,
  CREATE_TASK_SUCCESSFUL,
  FETCH_TASKS,
  FETCH_TASKS_SUCCESSFUL,
} from '../actions/taskActions';
function* createTaskSaga({
  payload,
}: DefaultAction<{
  task: TaskCreate;
  subTasks: string[];
}>) {
  const { data: task } = yield call(() => axios.post('/tasks', payload));
  yield put({
    type: CREATE_TASK_SUCCESSFUL,
    payload: task,
  });
}
function* fetchTasksSaga({ payload }: DefaultAction<Date>) {
```

```
  const { data: tasks } = yield call(() =>
    axios.get('/tasks', { params: { date: payload } }),
  );
  yield put({
    type: FETCH_TASKS_SUCCESSFUL,
    payload: {
      [payload.toDateString()]: tasks,
    }, });}
function* tasksSagas() {
  yield all([
    takeLatest(CREATE_TASK, createTaskSaga),
    takeLatest(FETCH_TASKS, fetchTasksSaga),
  ]);}
export default tasksSagas;
```

Роути для backend роботи з таймлайном. Task.routes.tsx

```
import { Router } from 'express';
import passport from 'passport';
import TaskController from 'controllers/TaskController';

const taskRoutes = Router();
// mark all routes as protected
taskRoutes.use(passport.authenticate('bearer', { session: false }));
taskRoutes.get('/', TaskController.fetchTasks);
taskRoutes.post('/', TaskController.createTask);
taskRoutes.get('/:taskId', TaskController.fetchTask);
taskRoutes.patch('/:taskId', TaskController.updateTask);
taskRoutes.delete('/:taskId', TaskController.deleteTask);
export default taskRoutes;
```

Модель Task.task.model.tsx

```
export interface ITask {
  id: number;
  title: string;
  startDate: Date;
  endDate?: Date;
  ancestorId?: number;
  type: TaskTypes;
  isCompleted: boolean;
  userId: number; }
@Table({ createdAt: false, updatedAt: false })
export class Task extends Model<Task> {
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;
  @Column
  title: string;
```

```
  @Column
  startDate: Date;
  @Column
  endDate?: Date | null;
  @Column
  type: TaskTypes;
  @Column
  isCompleted: boolean;

  @AllowNull
  @Column
  ancestorId?: number;
  @BelongsTo(() => User, 'userId')
  user: User;
  @HasMany(() => SubTask, 'taskId')
  subTasks: SubTask[];
  public static readonly TableName: string = 'Tasks';
}
```

Модель SubTask.subtask.model.tsx

```
import { Task } from './task.model';
export enum SubTaskTypes {
  task = 'TASK',
  challenge = 'CHALLENGE',
}
export interface ISubTask {
  id: number;
  title: string;
  type: SubTaskTypes;
  isCompleted: boolean;
  taskId: number | null;
  challengeStageId: number | null;
}

@Table({ createdAt: false, updatedAt: false })
export class SubTask extends Model<SubTask> {
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;
  @Column
  title: string;
  @Column
  type: SubTaskTypes;
  @Column
  isCompleted: boolean;
  @BelongsTo(() => Task, 'taskId')
  task: Task;
  public static readonly TableName: string = 'SubTasks';
}
```

Контроллер для моделі Task. TaskController.tsx

```
import { SubTask } from 'database/models/subTask.model';
import { Task, TaskTypes } from 'database/models/task.model';
import { IUser } from 'database/models/user.model';

class TaskController {
  public static async fetchTasks(
    request: Request,
    response: Response,
  ): Promise<Response<void>> {
    const requestUser = request.user as IUser;
    try {
      const { date } = request.query;
      const searchDate = date ? new Date(date as string) : null;
      const startDateRange = setHours(setMinutes(searchDate, 0), 0);
      const endDateRange = setHours(setMinutes(searchDate, 59), 23);
      const searchParameters: {
        userId: number;
        type: TaskTypes;
        startDate?: { [Op.between]: string[] };
      } = {
        userId: requestUser.id,
        type: TaskTypes.task,
      };
      if (searchDate) {
        searchParameters.startDate = {
          [Op.between]: [
            startDateRange.toISOString(),
            endDateRange.toISOString(),
          ],
        };
      }
      const tasks = await Task.findAll({
        include: [SubTask],
        where: searchParameters,
      });

      return response.status(200).send(tasks);
    } catch (error) {
      return response.status(404).send(error);
    }
  }
```

Сторінка профілю. UserProfileView.tsx

```
const UserProfileView = () => {
  return (
    <AppContainer>
```

```
      <ListContainer>
        <FlatList
          ListHeaderComponent={<ProfileGeneral />}
          data={items}
          renderItem={({ item }) => <ProfileDetailItem item={item} />}
          keyExtractor={(item) => item.id}
          showsVerticalScrollIndicator={false}
        />
      </ListContainer>
    </AppContainer>
  );
};
const ListContainer = styled(SafeAreaView)`
  flex: 1;`;
export default UserProfileView;
```

Action, reducer для роботи з користувачем.

```
const namespace = 'USER';
export const SET_USER = `${namespace}/SET_USER`;
export const UPDATE_USER = `${namespace}/UPDATE_USER`;
export interface State {
  id: string;
  email: string;
  name: string;
  credits: number;
  firstWeekDay: string;
  lastCheckedVersion: string;
  stripeCustomerId: string | null;
  stripeSubscriptionId: string | null;
  timezone: string;
  telegramLink: string | null;
  whatsAppLink: string | null;
  achievementIds: number[];
  createdAt: Date;
  updatedAt: Date | null;
}
const UserReducer = (
  state: Partial<State> = {},
  action: DefaultAction<{ user?: State }>,
) => {
  switch (action.type) {
    case SET_USER: {
      return (action.payload ?? {}) as State;
    }
    case UPDATE_USER: {
      return {
        ...state,
        ...action.payload?.user,
      };
    }
```

```
    default: {
      return state;
    }
  }
};

export const getUser = (state: RootState) => state[STATE_KEY];

export const getCompletedAchievements = (state: RootState) =>
  state[STATE_KEY].achievementIds ?? [];

export default UserReducer;

import { Router } from 'express';
import passport from 'passport';
import UserController from 'controllers/UserController';
const userRoutes = Router();
// mark all routes as protected
userRoutes.use(passport.authenticate('bearer', { session: false }));
userRoutes.get('', UserController.fetchAllUsers);
userRoutes.get('/:userId', UserController.fetchUser);
userRoutes.patch('/:userId', UserController.updateUser);
export default userRoutes;
```

Контроллер користувача.UserController.tsx

```
class UserController {
  public static async fetchAllUsers(
    _: Request,
    response: Response,
  ): Promise<Response<void>> {
    const users = await User.findAll();
    return response.status(200).send(users);
  }

  public static async fetchUser(
    request: Request,
    response: Response,
  ): Promise<Response<void>> {
    const { userId } = request.params;
    try {
      const user = await User.findByPk(userId, {
        include: [
          {
            model: UserHasAchievement,
            attributes: ['achievementId'],
          },
        ],
      });
      if (!user) {
        throw new Error('User does not exist');
```

```
      }
      // normalize user's achievements
      const normalizedUser = R.evolve(
        {
          achievementIds: (achievements) =>
            achievements.map(
              (achievement: { achievementId: number }) =>
                achievement.achievementId,
            ),
        },
        user.get({ plain: true }),
      );
      return response.status(200).send(normalizedUser);
    } catch (error) {
      return response.status(404).send(error);
    }
  }
```

Сторінка досягнень. AchivementsView.tsx

```
const AchievementsView = () => {
  const dispatch = useDispatch();
  const [
    currentAchievement,
    setCurrentAchievement,
  ] = useState<Achievement | null>(null);
  const achievements = useSelector(getAchievements);
  const completedAchievements = useSelector(getCompletedAchievements);

  const isCompletedCurrentAchievement = useMemo(
    () =>
      currentAchievement
        ? completedAchievements.includes(currentAchievement.id)
        : false,
    [completedAchievements, currentAchievement],
  );
  useEffect(() => {
    dispatch(fetchAchievements());
  }, [dispatch]);
  useEffect(() => {
    if (!currentAchievement && achievements.length > 0) {
      setCurrentAchievement(achievements[0]);
    }
  }, [achievements, currentAchievement]);
  return (
    <Container>
      <AchievementInfo
        achievement={currentAchievement}
        isCompleted={isCompletedCurrentAchievement}
      />
      <AchievementList
```

```
        currentAchievementId={currentAchievement?.id ?? null}
        achievements={achievements}
        completedAchievements={completedAchievements}
        setCurrentAchievement={setCurrentAchievement}
      />
    </Container>
  );
};
const Container = styled(View)`
```

Action, reducer, sagas для досягнень. AchivementsAction.tsx

```
const namespace = 'ACHIEVEMENTS';
export const FETCH_ACHIEVEMENTS = `${namespace}/FETCH_ACHIEVEMENTS`;
export const SET_ACHIEVEMENTS = `${namespace}/SET_ACHIEVEMENTS`;
export const fetchAchievements = () => ({
  type: FETCH_ACHIEVEMENTS,
});
import { DefaultAction, Achievement } from '~/interfaces';
import { SET_ACHIEVEMENTS } from '~/store/actions/achievementsActions';
export const STATE_KEY = 'achievements';
const AchievementsReducer = (
  state: Achievement[] = [],
  action: DefaultAction<Achievement[]>,
) => {
  switch (action.type) {
    case SET_ACHIEVEMENTS: {
      return action.payload;
    }
    default: {
      return state;
    }
  }
};
export const getAchievements = (state: RootState) => state[STATE_KEY];
export default AchievementsReducer;
import { all, call, put, takeLatest } from 'redux-saga/effects';
import axios from '~/configs/axios';
import {
  FETCH_ACHIEVEMENTS,
  SET_ACHIEVEMENTS,
} from '../actions/achievementsActions';
function* fetchAchievementsSaga() {
  const { data: achievements } = yield call(() => axios.get('/achievements'));
  yield put({
    type: SET_ACHIEVEMENTS,
    payload: achievements,
  });
}
function* achievementsSagas() {
  yield all([takeLatest(FETCH_ACHIEVEMENTS, fetchAchievementsSaga)]);
```

```
}
export default achievementsSagas;
```

Роути для досягнень. achievementsRoutes.tsx

```
import { Router } from 'express';
import passport from 'passport';

import AchievementsController from 'controllers/AchievementsController';
const achievementsRoutes = Router();
// mark all routes as protected
achievementsRoutes.use(passport.authenticate('bearer', { session: false }));
achievementsRoutes.get('', AchievementsController.fetchAllAchievements);
achievementsRoutes.get(
  '/:achievementId',
  AchievementsController.getAchievement,
);
export default achievementsRoutes;
```

Контроллер для досягнень. achievementsController.tsx

```
import { Request, Response } from 'express';
import { Achievement } from 'database/models/achievement.model';
class AchievementsController {
  public static async fetchAllAchievements(
    _: Request,
    response: Response,
  ): Promise<Response<Achievement[]>> {
    try {
      const achievements = await Achievement.findAll();
      return response.status(200).send(achievements);
    } catch (error) {
      return response.status(404).send(error);
    }
  }
  public static async getAchievement(
    request: Request,
    response: Response,
  ): Promise<Response<void>> {
    const { achievementId } = request.params;
    try {
      const achievement = await Achievement.findByPk(Number(achievementId));
      if (!achievement) {
        throw new Error('Achievement does not exist');
      }
      return response.status(200).send(achievement);
    } catch (error) {
      return response.status(404).send(error);
    }  }  }
export default AchievementsController;
```