МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система планування з елементами гейміфікації»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Ободяк В.К.**

Студента групи ІН.м-92 **Бондаренка В.Ю.**

Суми 2020

РЕФЕРАТ

Записка: 63 стор., 17 рис., 1 додаток, 20 джерел.

Об’єкт дослідження — процес планування з елементами гейміфікації.

Мета роботи — розробити інформаційну систему планування з елементами гейміфікації.

Методи дослідження — в процесі дослідження були використані такі технології як React Native, Styled components, Typescript, Javascript, Node, Express, Sequelize, Postman, Visual Studio Code.

Результати — створена інформаційна система з елементами гейміфікації, розроблений та стилізований макет додатку, розроблена база даних, інформаційна система протестована.

REACT NATIVE, STYLED COMPONENTS, TYPESCRIPT, NODE, EXPRESS, SEQUELIZE, POSTMAN, DOCKER

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *Інформатика*

Затверджую:

Зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТОВІ

Бондаренку Вадиму Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система планування з елементами гейміфікації*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Інформаційний огляд. 2) Постановка задачі. 3) Вибір методів вирішення. 4) Розробка веб-додатку. 5) Додавання елементів гейміфікації.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

_____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

п/п	Назва етапів кваліфікаційної магістерської роботи	Термін виконання проекту (роботи)	Примітка
.			
.			
.			
.			

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

ЗМІСТ

ВСТУП

Планування майбутніх дій, створення планів для виконання – важливий і незамінний процес в процесі життя людини. Думки про те, що потрібно робити далі, спіткають усіх: від бізнесмена, медпрацівника, актора до звичайного школяра, незалежно від життєвого статусу чи професії. Тому планування є невід'ємною складовою кожної людини [1].

На сьогоднішній день існує безліч технологічних рішень та методів для вирішення проблеми Планування є динамічним процесом і потребує постійного доступу до нього з метою визначення точки в плані та його коректування. Раніше використовували блокноти або записні книжки. В наш час більш доцільно використовувати телефон для більш швидкого доступу до плану. Смартфони дозволяють завантажити потрібний додаток і використовувати його у зручний для вас час [2].

Мобільний додаток — програмне забезпечення, спрямоване для роботи на смартфонах та планшетах. Доволі 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, інформацію про біржовий ринок, тощо. З кожним роком концепція мініатюрного функціонального комп'ютера набирала попит, а наявність необхідного рівня розвитку технологій та 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].

У зв'язку з великою кількістю додатків, які направлені на вирішення одного і того самого питання, постає проблема підтримки інтересу користувача до конкретного додатку.

В останні роки мобільні додатки стали все більше використовувати досвіт з ігрової індустрії, а саме використання елементів гейміфікації в додатках, в більшості це імплементація досягнень за певну активність, ігрової валюти, тощо.

Тому об’єктом дослідження є елементи гейміфікації, які виконують функції планування задач для виконання. Предметом дослідження є мобільний додаток планування особистого плану дня з елементами гейміфікації. Метою магістерської роботи є реалізація проєкту, що передбачає розробку модулів планування з елементами гейміфікації. 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.

Актуальність 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].

Гейміфікація - це використання ігрових підходів, які широко поширюються в комп'ютерних іграх, для неігрових процесів, що дозволяє підвищити волевиявлення учасників у вирішенні складних завдань, використання продуктів, послуг, посилення лояльності клієнтів [7].

Основними принципами гейміфікації є:

- принцип мотивації. Аудиторію потрібно мотивувати до взаємодії, адже ми любимо гри не просто так, а за результат. Людині приємно самому чогось досягти. Необхідно підібрати винагороду за інтерактив. Це може бути подарунок, купон на знижку, визнання;
- принцип несподіваних заохочень. Якщо ви плануєтся заохочення матеріальними призами, то можна використовувати емоційні прийоми. Пропонуйте нестандартні опитування і тести з непередбачуваним фіналом. Подібні прийоми викликають цікавість і позитивні емоції;
- принцип статусу. Кожна людина прагне до певного статусу і хоче стати кращим у своїй діяльності. Йому не просто приємно підвищити

самооцінку, а й продемонструвати свої успіхи іншим. Саме тому в багатьох іграх і додатках навчального характеру відображається прогрес користувача і його друзів [8].

Основними елементами гейміфікації є:

- бали;
- лідерборди;
- статуси;
- рейтинги;
- бейджи;
- нагороди / досягнення;
- рівні;
- аватари [9].

1.2 Аналіз проєкту в порівнянні з подібними

Існує безліч мобільних додатків для планування, з різним дизайном та специфікою. При цьому більшість із них має типовий вигляд – календаря. Хоча, при цьому існує також немало і більш наглядних додатків у вигляді діаграми Ганта [10].

Торгівельний простір мобільних застосунків у наш час неосяжно великий, з усіма аналогами ознайомитись неможливо, тому розглянемо і порівняємо можливості найбільш популярних мобільних додатків для App Store та Google Play [11].

Додаток Google Calendar

Google Calendar - це служба управління календарем і плануванням часу, розроблена Google.

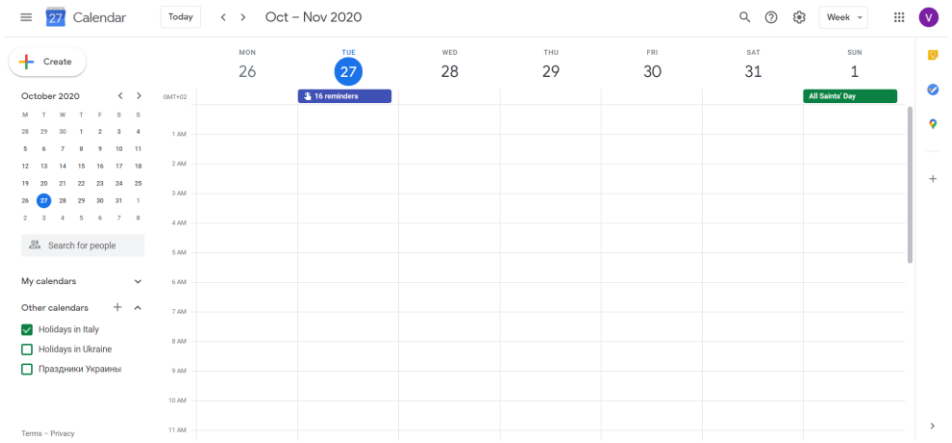

Рисунок 1.1 - Зображення інтерфейсу додатку «Google Calendar»

Переваги планувальника:

- безкоштовний;
- створення задач на різні періоди, та можливості встановлення дуплікації задач на певний період;
- синхронізація між всіма девайсами, у яких був здійснений вхід через один і той самий обліковий запис;
- різні варіанти відображення (на день, неділю, місяць);
- можливість створення різних типів задач;
- широкі можливості в налаштуванні різних параметрів;
- широкий вибір локалізації сервісу;
- кроссплатформенність;
- можливість вибору теми додатку (світла, темна).

Недоліки планувальника:

- перевантажений інтерфейс, занадто багато функціоналу, для створення задачі;
- проблемне відображення задач, паралельних у часі.

Також одним із важливих плюсів є те що даний сервіс має велику кількість інтеграцій з іншими сервісами, що робить його ще більш зручним у користуванні [12].

Додаток Dododo

Даний сервіс намагається замінити планувальник, трекер звичок, нагадування, примітки та допомогти користувачу зберегти всю важливу інформацію в зручному додатку.

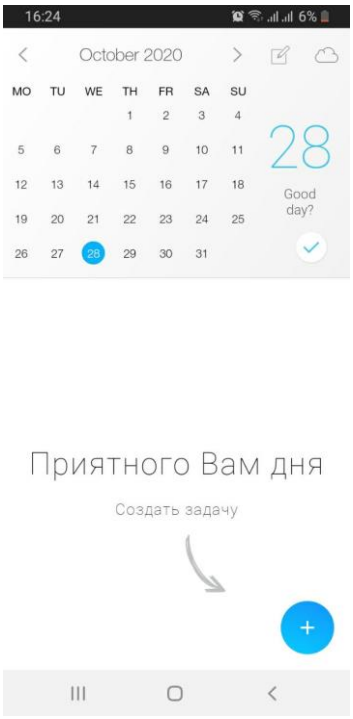

Рисунок 1.2 - Зображення інтерфейсу додатку «Dododo»

Переваги планувальника Dododo:

- синхронізація з Google Calendar;
- зручний перегляд планів по конкретному дню та механізм додавання нових задач;
- можливість додавання заміток на конкретний день;
- приємний дизайн у світлих тонах.

Недоліки:

- має 7 днів пробного періоду, а далі – вибір тарифного плану від 2$ до 6$ за міс.;

- доступний лише на Android;
- занадто малий шрифт додатку, без можливості до його зміни в налаштуваннях;
- навігація реалізована лише на +-1 день, в разі якщо необхідно перейти на конкретний день – виникають труднощі;
- звук оповіщення занадто короткий;
- відсутня можливість додавання повторюваних задач.

В цілому має весь необхідний функціонал для швидкого створення задач та їх перегляду. Підійде для планування в короткостроковій перспективі – неділю, місяць.

Основними недоліками сервісу, є те що він створений виключно на Android та є платним сервісом [13].

Додаток Focuster

Сервіс дозволяє автоматично планувати список справ у календарі, допомагаючи користувачу зберігати фокус, визначати пріоритети завдань та досягати найважливіших цілей щодня.

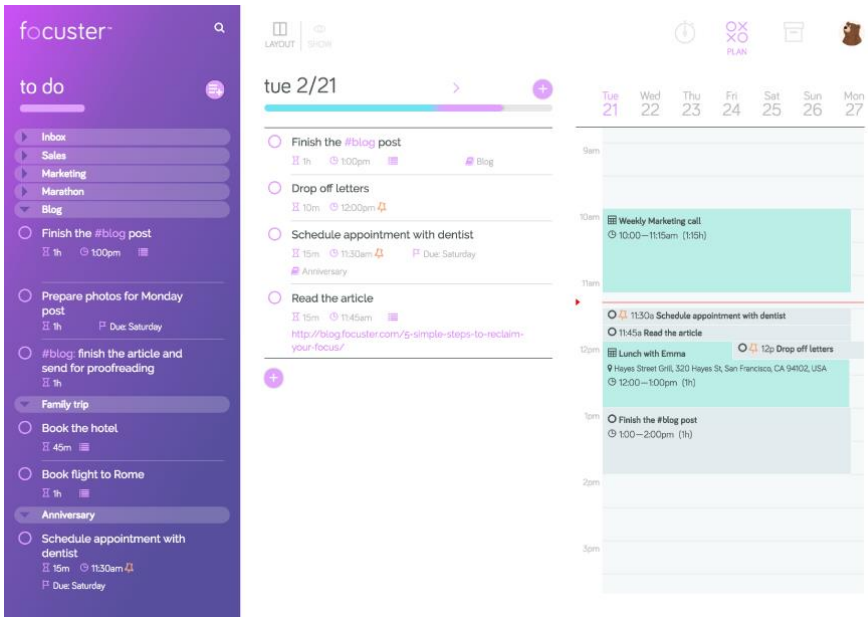

Рисунок 1.3 - Зображення інтерфейсу додатку «Focuster»

Переваги планувальника Focuster:

- синхронізація з Google Calendar;
- алгоритм, який автоматично додає задачу в незапланований час;
- зручна реалізація стеку задач – блоку з задачами без конкретного часу виконання, які можна перенести в основний таймлайн з автоматичним введенням часу для задачі;
- велика кількість інтеграцій зі сторонніми сервісами;
- адаптивний інтерфейс;
- приємний дизайн у світлих тонах.

Недоліки:

- не має можливості створення суб-задач;
- має 14 днів пробного періоду, а далі – вибір тарифного плану від 10$ до 20$ за міс.;
- доступний лише на Web;
- баги, зі створенням задач.

Даний додаток є доволі зручним при створенні плану на день, завдяки алгоритму, який додає задачі у вільний від інших задач час [14].

Основним недоліком є доволі висока ціна за базовий тариф – 10$ в порівнянні з конкурентами, та його реалізація лише у вигляді Web-сервісу, хоча розробники планують реалізацію на Android та Ios, але на сьогоднішній день даний сервіс є доволі нішовим сервісом, через те що користувач з більшою вірогідністю завантажить мобільний додаток для планування, чим буде заходити в мобільний браузер для планування свого дня [15].

Додаток CloudCal Calendar Agenda Planner Organizer To Do

CloudCal - це єдина безкоштовна програма календаря, яка консолідує всі календарі Google і Microsoft в одному щомісячному перегляді, що миттєвий огляд доступності, а також подій та завдань Evernote, Meetup та Eventbrite.

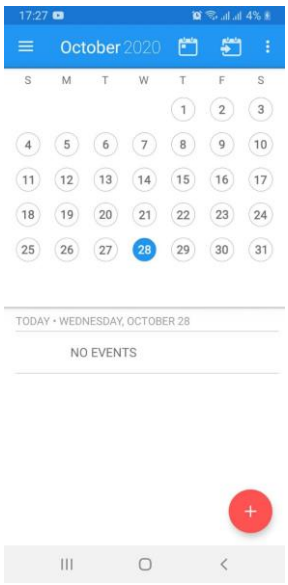

Рисунок 1.4 - Зображення інтерфейсу додатку
«CloudCal Calendar Agenda Planner Organizer To Do»

Переваги планувальника:

- можливість користування безкоштовно;
- різні варіанти відображення (на день, неділю, місяць);
- синхронізація з різними календарями (Google Calendar, Samsung Calendar);
- зручна реалізація опису основного функціоналу та як ним необхідно користуватись, при першому старті додатку;
- зручна навігація по дням;
- зручна кнопка повернення до сьогоднішнього дня;
- приємний дизайн у світлих тонах.

Недоліки:

- не має можливості створення суб-задач;
- не досить інтуїтивно зрозуміле редагування задач;
- не можливо встановити точний час для задачі (наприклад 10:43), а лише інтервали часу кратні 15 хв. (10:15, 10:30, 10:45).

Сервіс має преміум версію за 100 грн., яка дозволить прикріпляти файли до задач, додавати синхронізацію з evernote та містить багато інших додаткових функцій.

При цьому сервіс доступний лише на платформі Android. Також користувачами зазначається збільшення часу виходу нових обновлень сервісу та все більша затримка у вирішенні проблем з сервісом [16].

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 додатку			
	Google Calendar	Dododo	Focuster	CloudCal Calendar Agenda Planner Organizer To Do
Кросплатформенність	✓	-	-	-
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	-	-	-	-

Після аналізу таблиці 1.1 можна зробити висновок, що жоден з розглянутих мобільних додатків не має абсолютно всі представлені характеристики.

При цьому жоден з аналізованих додатків не має елементів гейміфікації, тому оптимальним рішенням у цьому випадку є розробка власного мобільного додатку, який буде володіти всіма виділеними функціями.

1.3. Мета роботи

Проаналізувавши предметну область, застосування інформаційних технологій для вирішення поставленої задачі, огляду та порівняння мобільних застосунків-аналогів, можна сформулювати мету роботи.

Мета роботи − реалізація модулів мобільного додатку, що передбачає розробку фунціоналу, який дозволить планувати свій день, з одним із елементів гейміфікації – досягненнями.

Розроблюваний мобільний додаток надасть можливість користуватися функціоналом інтерфейсів, описаних нижче:

Сторінка планування на день

● перемикання між днями, як + 1 - 1 день, так і на вибір;

● модальне вікно, 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, як приклад можна зробити щось типу стека завдань або таймлайну.

Сама задача повинна виглядати на графіку и вигляді прямокутника.

Створення та редагування завдання

Являє собою модальне вікно. На цій сторінці повинні бути:

● назва;

● час початку завдання;

● час закінчення завдання.

При цьому можливість створення завдань паралельних в часі повинна бути. Прикріплення файлів до задачі не передбачено.

Створення та редагування завдання

Являє собою окрему сторінку, на які можна:

- переглянути всі наявні досягнення;
- переглянути досягнення, які вже отримані;
- переглянути вимоги до досягнень, які ще не були отримані.

Також необхідна реалізація механізму показу досягнень, за певну активність користувача.

Сторінка профілю користувача

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

● відображення email користувача;

● відображення ім’я профіля;

● можливість видалення аккаунту.

До задач проєкту можна віднести: визначення основних функціональних вимог продукту, поведінки і інтерфейсів додатку, вибір інструменту та методу реалізації, проєктування модулів та реалізація додатку, що забезпечує виконання запланованого функціоналу. Спроєктований і реалізований продукт у вигляді модулів буде являти частину функціоналу мобільного додатку.

1.4. Постановка задачі

Проаналізувавши дослідження сфери роботи та подібні проєкти, були виявлені такі задачі для досягнення поставленої мети:

1. Вибрати інструменти реалізації.
2. Розробити модуль планування.
3. Розробити профіль користувача.
4. Додати елементи гейміфікації.

2 ВИБІР МЕТОДІВ ТА ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ

2.1 Вибір методу реалізації

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

В процесі реалізації використовувався 1-недільний спринт, з метою коректування плану розробки у зв’язку з новими даними щодо стану розробки сервісу.

Спринт - Scrum - це каркас розробки, з використанням якого люди можуть вирішувати з'являються проблеми, при цьому продуктивно і виробляючи продукти найвищої значущості. Рекомендовано 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 [17].

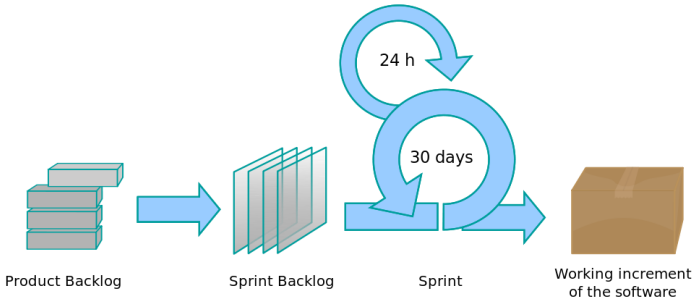

Рисунок 2.1 – Життєвий цикл одного спринту

Як видно з рисунку, життєвий цикл методології Scrum складається з підготовчих етапів для спринту і завершальних етапів. На кожному цьому етапі відбувається певна подія [18].

2.2 Вибір програмних засобів

Для створення мобільного додатку існує доволі багато інструментів та технологій, тому необхідно виділити основні технології, та обрати оптимальний варіант. Порівняльний аналіз інструментів наведений в таблиці 2.1.

Таблиця 2.1. Порівняльний аналіз існуючих інструментів

Х-ка	Назва технології			
	React Native	Flutter	Swift	Kotlin
Продуктивність	Низька	Висока	Висока	Висока
Кросплатформенність	Android / Ios	Android / IOS	IOS	Android
Складність розробки	Невисокий, вимагає знань js.	Високий, вимагає знань Dart	Високий, вимагає знань Objective C	Високий, вимагає знань Java / Scala.
Стан розвитку екосистеми, бібліотеки	Розвинута платформа, та екосистема	Молода платформа, готових рішень доволі мало	Розвинута платформа та екосистема	Розвинута платформа та екосистема

Відповідно до таблиці інструментом для реалізації frontend складової обрано React Native. Це зумовлено:

- простотою переходу з React фреймворку для Web;
- високим станом розвитку екосистеми, наявності готових рішень, що прискорить процес розробки;
- одночасно реалізацією як Android так і для IOS;
- порівняно низьку заробітну плату для розробників, в разі якщо буде вирішено продовжити розробку сервісу.

React Native – це інструмент для розробки мобільних додатків з відкритим вихідним кодом, створений Facebook. Дозволяє створювати додатки для Android і iOS, використовуючи фреймворк. Являє собою потужний кросплатформенний інструмент для розробки мобільних додатків, зі можливістю до швидкої розробки [19].

React Native надає компоненти для тексту, зображень, введення з клавіатури, гортання списків, індикатора виконання, анімації, буфера обміну, посилань і т.п. Ці компоненти значно прискорюють процес розробки додатків, а функція «Hot Reloading» також економить багато часу, оскільки дозволяє перезавантажити програму без повторної компіляції всього коду, що прискорює процес створення збірки у 2-3 рази в порівнянні з Android Studio.

Інструментом для реалізації backend складової обрано бібліотеку для NodeJs - Express.

Express - це мінімалістичний та гнучкий веб-фреймворк для доданого Node.js, що забезпечує загальний набір функцій для мобільних та веб-додатків.

Маючи у своєму розпорядженні безліч службових методів HTTP та проміжних функцій (middlewares), можна швидко створювати API [20].

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ

Реалізація магістерського дослідження складалася з тьох спринтів, що в свою чергу мають декілька етапів (рис.3.1).

Рисунок 3.1 – Кроки реалізації проекту

3.1 Модуль планування

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

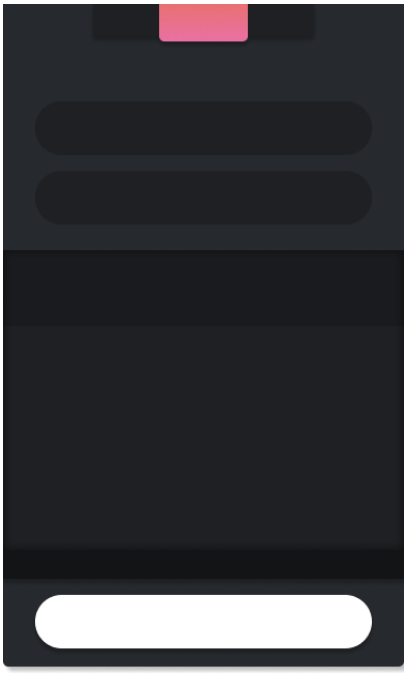

Рисунок 3.2 – Розмежування сторінки таймлайну на області та створення блоків

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

Рисунок 3.3 – Додаткові елементи, шаблон тексту та іконок сторінки таймлайну

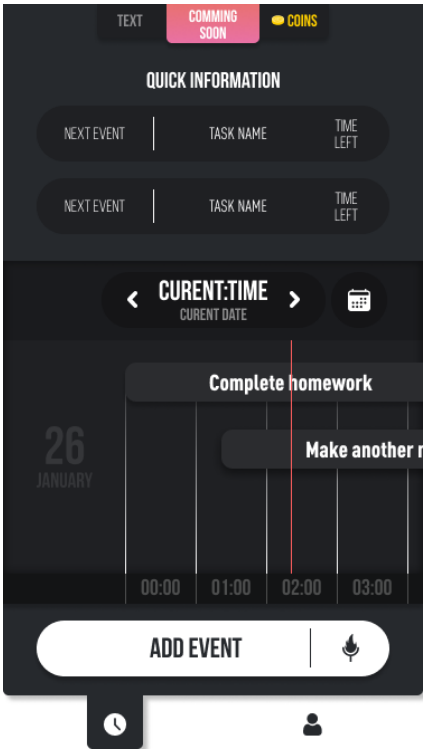

Рисунок 3.4 – Фінальний вигляд екрану перегляду створених задач

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

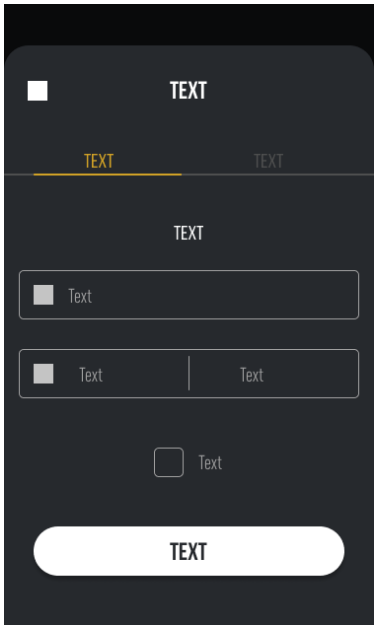

Рисунок 3.5 – Розмежування модального вікна створення задач на області та додання функціональних елементів, шаблону тексту та іконок

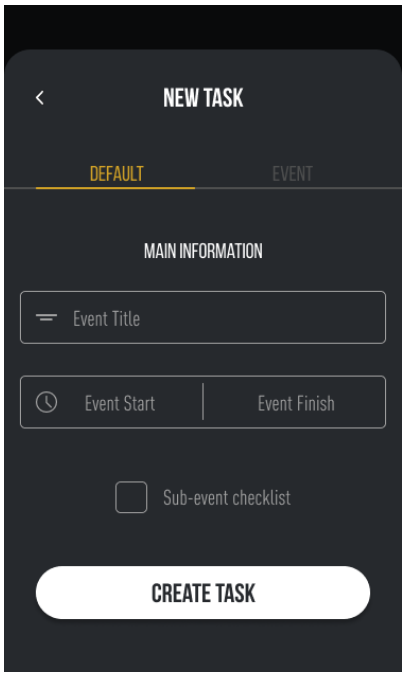

Рисунок 3.6 – Фінальний вигляд модального вікна створення задач

Далі слідує програмна реалізація сторінки планування згідно з функціональними вимогами. Основні файли, де реалізована логіка та функціональний принцип роботи модуля планування, представлені в табл. 3.1.

Таблиця 3.1. 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 напряму 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 контролер 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 параметрів 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 суб-задачами, 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.

3.2 Модуль 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.

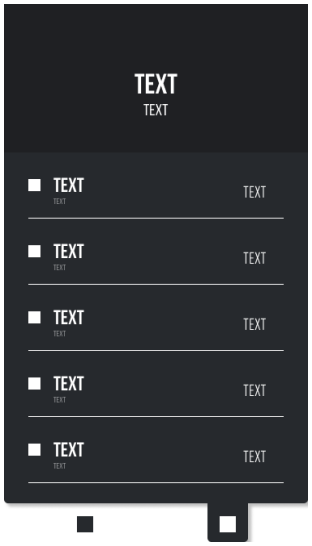

Рисунок 3.7 – Додаткові елементи, шаблон тексту та іконок

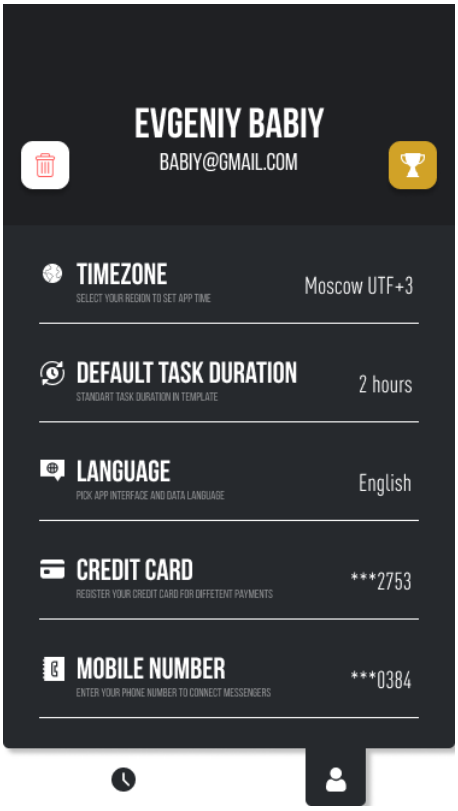

Рисунок 3.8 – Фінальний вигляд сторінки профілю

Основні файли, де реалізована логіка та функціональний принцип роботи модуля профіля користувача представлені в табл. 3.2.

Таблиця 3.2. 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 userReducer.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.

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.

3.3 Додавання елементів гейміфікації

Було обрано серед наявних елементів геймфікації створити досягнення для користувача за певну активність. Відповідно необхідно створити сторінку досягнень, де будуть відображені всі досягнення, які користувач уже отримав та може отримати за певну активність.

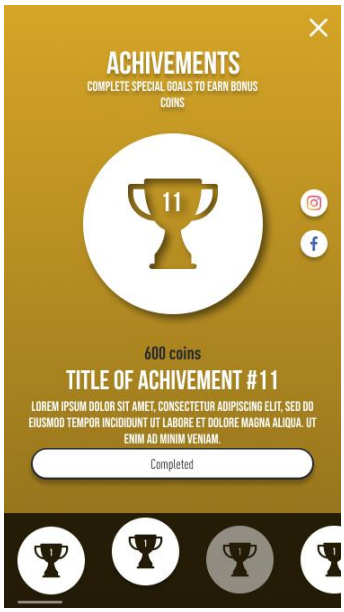

Рисунок 3.9 – Фінальний вигляд сторінки досягнень

Основні файли, де реалізована логіка та функціональний принцип роботи модуля досягнень користувача представлені в табл. 3.3.

Таблиця 3.3. Основні файли реалізації модуля досягнень

Назва файлу	**Функції**
AchievementsView.tsx (компонент досягнень)	Являє собою компонент, який включає в себе суб-компоненти які відповідають за відображення досягнень.
achievementsReducer.ts (reducer користувача)	Являє собою частину комбінованого reducer'а. Відповідає за збереження даних досягнень в глобальному сторі.
achievementsSagas.ts (саги для досягнень)	Являє собою набором побічних ефектів для роботи з даними досягнень.
AchievementsController.ts (API конроллер досягнень)	Являє собою набір методів, для обробки запитів з оновлення даних досягнень.

Відображення сторінки досягнень. Файл AchievementsView.tsx.

В даному файлі імпортуються:

- Хук useEffect з пакету react, який дозволяє створювати побічні ефекти для в функціональному компонеті;
- Хук useState з пакету react, який дозволяє створювати state в функціональному компоненті;
- Хук useMemo з пакету react, який дозволяє мемоїзувати змінну;
- Хук useDispatch з пакету react, який повертає фунцію яка може емітувати події для redux-store;
- Хук useSelector з пакету react, який дозволяє отримати частину redux-store;
- Функція styled для стилізації елементів сторінки планування;
- Компонент AchievementInfo, який відповідає за відображення досягнення у розгорнутому вигляді;

- Компонент AchievementList, який відповідає за відображення слайдера зі списком всіх досягненням, та навігації по ним.

Даний компонент є компонентом-контейнером, який включає в себе 2 основних компоненти – AchievementInfo та AchievementList.

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

Даний файл має 6 експортів:

- STATE_KEY – константа, яка зберігає State;
- State – інтерфейс, який буде використаний при декларації глобального типу всього reduxStore (IRootState);
- getAchievements – функція для отримання всіх наявних досягнень;
- reducer – експортується за замовчуванням і являє собою регулятор, в якому в залежності від типу події модифікуються досягнення.

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.

Побічні ефекти для досягнень. Файл achievementsSagas.ts.

Даний файл містить основну функцію-генератор achievementsSagas, яка за допомогою методу ALL комбінує всі запити, що стосуються отримання досягнень та їх параметрів. Дана функція експортується за замочуванням, та буде підключена та запущена за допомогою методу fork в кореневому файлі rootSagas.ts. Сама функція має 1 побічний ефект, а саме fetchAchievementsSaga – функція-генератор, яка відправить GET запит на API з отриманням всіх наявних досягнень, а потім оновить redux-state.

Обробка запитів. Файл AchievementsController.ts.

Даний контролер необхідний для оброблення запитів з оновлення та отримання досягнень. Він обслуговує тільки модель Achievement.

Відповідно, даний контролер імпортує модель Achievement, яка являє собою модель досягнень.

В файлі описано клас AchievementsController, що використовує наступні методи:

- fetchAllAchievements – даний метод повертає всі створені досягнення;
- getAchievement – даний метод повертає одне досягнення, базуючись на параметрі achievementId, який задекларований в routes досягнень;

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

3.4 Тестування сервісу

В ході розробки інформаційної системи, завершальним етапом розробки є її тестування. В ході тестування отримано працездатну інформаційну систему з трьома модулями:

- модуль планування;
- модуль профілю;
- модуль досягнень.

Нижче наведений готовий вигляд інтерфейсу кожного із модулів.

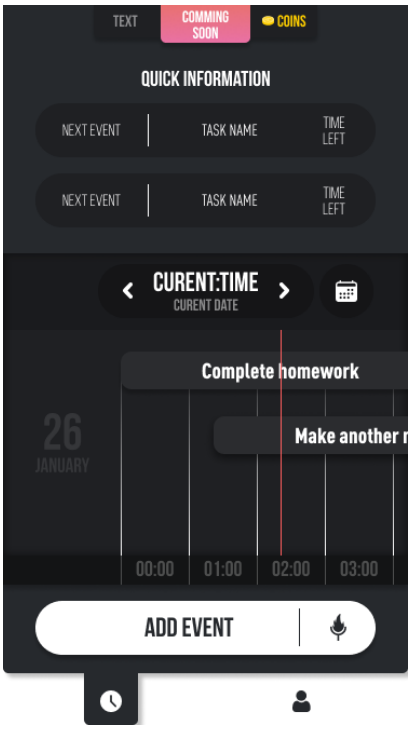

Рисунок 3.10 – Фінальний вигляд екрану перегляду створених задач

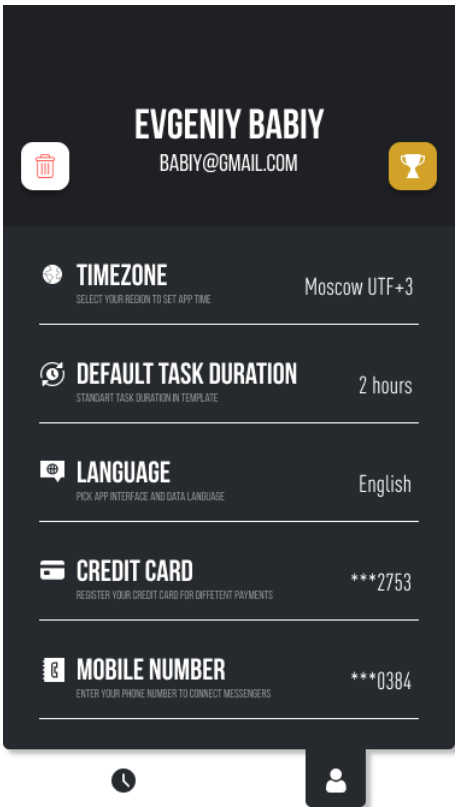

Рисунок 3.11 – Фінальний вигляд сторінки профілю

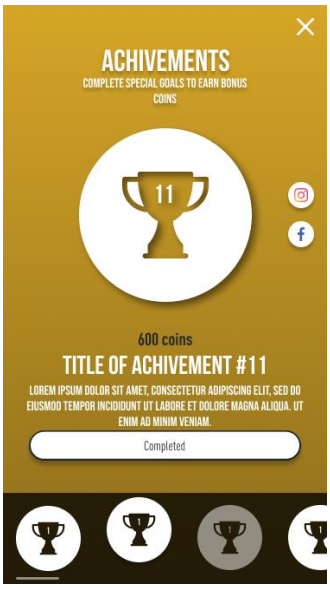

Рисунок 3.12 – Фінальний вигляд сторінки досягнень

Таким чином, результати тестування інформаційної системи показали її надійну роботу.

ВИСНОВКИ

В ході дослідження були виконані такі роботи:

1. Проаналізовані подібні сервіси для планування, і виявлено необхідність додавання елементів гейміфікації в інформаційну систему планування. Також було 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.
2. Для розробки інформаційної системи була обрана методологія scrum, відповідно до неї розробка проєкту була поділена спринти. На етапі постановки задачі описано головну мету проєкту та задачі, які необхідно виконати для досягнення поставленої мети. Сформульовані функціональні вимоги до модулів мобільного додатку.
3. В ході аналізу інструментів для розробки мобільного додатку було проаналізовано методи створення мобільних додатків та обраний інструментарій для виконання поставлених задач. Були обрані технології JavaScript, а саме фреймворк React Native та Node.JS як сучасні та продуктивні засоби для реалізації мобільних додатків.
4. В результаті виконання дипломної роботи було створено мобільний додаток для планування з елементами гейміфікації.

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. 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.

Модуль з досягненнями містить у собі сторінку досягнень, та логіку збереження прогресу досягнень.

Практичне значення розроблених модулів полягає у тому, що вони є необхідними функціональними частинами для комерційного мобільного додатку, призначеного для самоорганізації та планування власної роботи користувачами. При цьому модуль досягнень збільшує зацікавленість користувача в даному сервісі, і збільшує тривалість роботи з ним.

СПИСОК ЛІТЕРАТУРИ

1. Хан Д. Планирование и контроль: концепция контроллинга: пер. с нем. — М.: Финансы и статистика, 1997. — 800с.

2. USwitch. Full story about mobile phone development – https://www.uswitch.com/mobiles/guides/history-of-mobile-phones/.

3. 2018 – год мобильных приложений в МФО – https://minfin.com.ua/ua/2018/01/30/32121301/

4. QAinfo. Мобільний додаток – вносимо розуміння у значення терміну – https://www.quality-assurance-group.com/mobilnyj-dodatok-vnosymo-rozuminnya-u-znachennya-terminu/.

5. ОсвітаUA. Менеджмент – https://ru.osvita.ua/vnz/reports/management/15423/.

6. Koloro. Тайм-менеджмент: принципи управління своїм часом – https://koloro.ua/ua/blog/menedzhment/tajm-menedzhment-principy-upravleniya-svoim-vremenem.html/.

7. Визначення гейміфікації – https://vsetreningi.ru/schools/geymifikaciya/.

8. Элементы геймификации в бизнесе – https://active-vision.ru/blog/geymifikatsii-v-biznese/.

9. Что такое геймификация? Определение советы – https://sendpulse.ua/ru/support/glossary/gamification/.

10. InvoDigital. Топ-100 популярних додатків світу – https://invodigital.com/blog/article/1416-top-100-samykh-populyarnykh-mobilnykh-prilozhenij-v-mire/.

11. Gagadget. App Store & Google Play – https://gagadget.com/17030-google-play-ili-app-store/.

12. Live Business. Google Calendar – https://www.livebusiness.ru/tool/337/.

13. Dododo. Офіційна сторінка в Google Play – https://play.google.com/store/apps/details?id=idea.dododo.app&hl=en_US/.

14. Focuster. Офіційний сайт – https://www.focuster.com/.

15. Focuster Review For 2020: Features, Pricing & Competitors – https://www.screwtheninetofive.com/focuster-review/.

16. CloudCal Calendar Agenda Planner Organizer To Do. Офіційна сторінка в Google Play – https://play.google.com/store/apps/details?id=net.cloudcal.cal &hl=en_US/.

17. Scrum. Що таке Scrum – https://ru.scrum-time.com/infobase/scrum-sprint.php/.

18. QA evolution. Scrum методологія розробки – https://qaevolution.ru/ metodologiya-menedzhment/scrum/.

19. React Native learning book – https://www.oreilly.com/library/ view/learning-react-native/9781491929049/ ch01.html/.

20. Express official website – https://expressjs.com/ru/.

ДОДАТОК

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. authActions.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.to
ken}
```

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.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.model.ts

```
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 задач. 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;
```

Модель user.model.ts

```
import {
  DataType,
  PrimaryKey,
  AutoIncrement,
  Table,
  Column,
  Model,
  HasMany,
  CreatedAt,
  UpdatedAt,
} from 'sequelize-typescript';

import { Task } from './task.model';
import UserHasAchievement from './userHasAchievement.model';

export interface IUser {
  id: number;
  name: string;
  email: string;
  credits: number;
  firstWeekDay: string;
  timezone: string;
  googleUserId: string;
  stripeCustomerId: string | null;
  stripeSubscriptionId: string;
  telegramLink: string;
  whatsAppLink: string;
  lastCheckedVersion: string | null;
  createdAt: Date;
  updatedAt: Date;
  achievementIds: { achievementId: string }[];
}

@Table
export class User extends Model<User> {
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;

  @Column
  name: string;

  @Column
  email: string;
```

```
  @Column
  credits: number;

  @Column
  timezone: string;

  @Column
  firstWeekDay: string;

  @Column
  googleUserId: string;

  @Column
  stripeCustomerId: string | null;

  @Column
  stripeSubscriptionId: string;

  @Column
  telegramLink: string;

  @Column
  whatsAppLink: string;

  @Column
  lastCheckedVersion: string | null;

  @CreatedAt
  @Column({ field: 'createdAt', type: DataType.DATE })
  createdAt: Date;

  @UpdatedAt
  @Column({ field: 'updatedAt', type: DataType.DATE })
  updatedAt: Date;

  // Associations
  @HasMany(() => Task, 'userId')
  tasks: Task[];

  @HasMany(() => UserHasAchievement, 'userId')
  achievementIds: { achievementId: string }[];

  public static readonly TableName: string = 'Users';
}
```

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;
```

Модель achievement.model.ts

```
import {
  AutoIncrement,
  Column,
  PrimaryKey,
  Table,
  Model,
  Unique,
} from 'sequelize-typescript';

export enum AchievementTypes {
  activity = 'ACTIVITY',
  tasks = 'TASKS',
}

export interface IAchievement {
  id: number;
  title: string;
  description: string;
  image: string;
  type: AchievementTypes;
  credits: number;
}

@Table({ createdAt: false, updatedAt: false })
export class Achievement extends Model<Achievement> {
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;

  @Unique
  @Column
  title: string;

  @Column
  description: string;

  @Column
  image: string;

  @Column
  type: string;

  @Column
  credits: number;

  public static readonly TableName: string = 'Achievements';
}
```

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;
```