МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з _____Компоненти програмної інженерії_____
(назва дисципліни)

на тему:_____Мобільний застосунок для контролю особистого часу_____

Студента __1___ курсу, групи __ІП-13____
Нещерета Віталія Олександровича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник _____асистент_Ахаладзе І. Е._____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

_____ доц, к.т.н. Лісовиченко О. І.
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ асистент Ахаладзе А. Е.
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2024 рік

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *інформатики та програмної інженерії*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
_____ *Максим Головченко*
(підпис)
“___”_____2024 р.

З А В Д А Н Н Я
НА КУРСОВУ РОБОТУ СТУДЕНТУ

Нещерету Віталію Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Мобільний застосунок для контролю особистого часу »

керівник роботи *Ахаладзе Ілля Елдарійович, асистент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Термін подання студентом роботи «4» *січня 2024 року*

3. Вихідні дані до роботи

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Аналіз якості та тестування програмного забезпечення

4) Впровадження та супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Діаграма варіантів використань

2) BPMN модель фільтрації завдань

3) BPMN модель оновлення стану Pomodoro таймера

4) Схема архітектури застосунку

6. Консультанти розділів роботи

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

7. Дата видачі завдання «12» *жовтня 2023 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Примітка
1.	*Вивчення рекомендованої літератури*	*до 10.10.2023*	
2.	*Аналіз існуючих методів розв'язання задачі*	*10.10.2023 - 12.10.2023*	
3.	*Постановка та формалізація задачі*	*13.10.2023 - 14.10.2023*	
4.	*Аналіз вимог до програмного забезпечення*	*14.10.2023- 15.10.2023*	
5.	*Алгоритмізація задачі*	*16.10.2023- 17.10.2023*	
6.	*Моделювання програмного забезпечення*	*18.10.2023- 19.10.2023*	
7.	*Обґрунтування використовуваних технічних засобів*	*20.10.2023- 21.10.2023*	
8.	*Розробка архітектури програмного забезпечення*	*22.10.2023- 25.10.2023*	
9.	*Розробка програмного забезпечення*	*26.10.2023- 18.12.2023*	
10.	*Налагодження програми*	*18.12.2023- 26.12.2023*	
11.	*Виконання графічних документів*	*26.12.2023- 27.12.2023*	
12.	*Оформлення пояснювальної записки*	*27.12.2023- 03.01.2023*	
13.	*Подання КР на перевірку*	*04.01.2023*	
14.	*Захист КР*	*06.01.2024*	

Студент _____ Віталій НЕЩЕРЕТ
(підпис)

Керівник _____ Ілля АХАЛАДЗЕ
(підпис)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка курсової роботи складається з 4 розділів, містить 9 рисунків, 46 таблиць, 15 джерел. Обсяг основної частини складає 51 сторінку.

Мета. Розробка ставить перед собою досягнення наступних цілей:

– розробити функціонал для детального керування завданнями та категоріями;

– реалізувати синхронізацію даних користувача між пристроями за допомогою спільного облікового запису;

– покращення структурування завдань за рахунок допрацювання фільтрації;

– вирішення проблеми фокусування за допомогою впровадження Pomodoro таймера.

У розділі аналізу вимог були поставлені поставлені вимоги для програмного забезпечення.

У розділі моделювання програмного забезпечення було описано архітектуру програмного забезпечення та алгоритми вирішення прикладних задач.

У розділі аналіз якості були описані основні тест кейси, та стани системи після проведення тестування.

У розділі впровадження та супровід було описано процеси розгортання програмного забезпечення.

КЛЮЧОВІ СЛОВА: ТАЙМ-МЕНЕДЖМЕНТ, ANDROID, POMODORO, КОНТРОЛЬ ЧАСОМ

Пояснювальна записка
до курсової роботи

на тему: Мобільний застосунок для контролю особистого часу

КПІ.ІП-1324.045490.01.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
SDK	–	Software development kit
IT	–	Інформаційні технології
ОС	–	Операційна система.
БД	–	База даних.

ВСТУП

У сучасному світі, де людина постійно перебуває в оточенні інформації та завдань, контроль особистого часу є надзвичайно важливим. Це дозволяє ефективніше використовувати свій час, досягати поставлених цілей та уникати стресу.

Сучасні тенденції вказують на зростання інтересу до рішень, спрямованих на підвищення продуктивності та ефективного використання часу. Незважаючи на попит, багато існуючих рішень мають обмежені функції або не дозволяють користувачам детально контролювати свій час.

Мобільні прострої, які завжди під рукою, дозволяють створювати більш зучні та функціональні програми для управління часом, що значною мірою вирішує проблему та дозволяє тримати всі свої плани під рукою.

Розроблене мною рішеня базується на кращих практиках та інтегрує в себе передові технології для надання користувачам унікального інструменту, який сприяє більш ефективному управлінню їхніми завданнями та часом. Розробка відкриває багато можливостей для ефективного управління часом та завданнями, створюючи платформу, яка враховує потреби сучасного користувача в багатозадачності, організації та досягненні цілей.

Додаток має широкий спектр застосувань і придатний для використання як в особистому, так і професійному житті. Від студентів і фрілансерів до бізнесменів та професіоналів будь-якої галузі – усі можуть скористатися перевагами детального контролю над своїм часом та завданнями.

У наш час, коли ефективне управління часом стає ключовим фактором успіху, мобільний додаток є відмінним інструментом для тих, хто прагне досягти високої продуктивності та збалансованого стилю життя.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Тайм-менеджмент є систематичним підходом до ефективного використання часу з метою досягнення поставлених цілей та завдань. Це важливий інструмент у сучасному житті, оскільки сучасне суспільство характеризується високим темпом життя, підвищеною конкуренцією та постійним потоком інформації. Відправним пунктом тайм-менеджменту є усвідомлення пріоритетів та раціональне розподілення часу між різними аспектами життя [1].

В умовах постійних викликів та можливостей, вміння ефективно володіти своїм часом стає ключовою компетенцією. Тайм-менеджмент сприяє підвищенню продуктивності та зниженню ризиків втрати часу на неважливі або непродуктивні справи. Відповідальне використання часу дозволяє досягати поставлених завдань, підтримує психічне здоров'я та сприяє досягненню балансу між професійним та особистим життям.

Одним з найважливіших викликів у керуванні особистим часом є ефективне управління власними пріоритетами. Здатність визначати найважливіші завдання та концентруватися на них в умовах великої кількості різноманітних обов'язків визначає успішність в досягненні поставлених цілей.

Також важливим аспектом є балансування між професійним та особистим життям [2]. Сучасна реальність часто ставить перед людьми завдання поєднувати високоінтенсивну професійну діяльність з утриманням стабільних відносин та особистим розвитком. Відповідальність за кілька сфер життя ставить людей перед необхідністю виваженості у розподілі часу.

Для полегшення цих задач можна використовувати візуалізацію завдань та планів, що відіграє ключову роль у розумінні та досягненні цілей, оскільки вона надає конкретну форму та образ предметам або ідеям. Візуальні зображення допомагають аналізувати та виявляти зв'язки між різними елементами, що сприяє кращому усвідомленню стратегії досягнення поставленого завдання [3].

Письмове фіксування планів, у свою чергу, відіграє роль структуруючого елемента, який допомагає забезпечити послідовність дій. Записані на папері чи

електронному носії плани надають конкретну форму ідеям та дозволяють їм зберігатися в систематизованому вигляді. Цей процес створює основу для оцінки та контролю за власною продуктивністю, що робить його важливим інструментом у досягненні успіху.

Крім того, важливо не лише спланувати час, а й дотримуватись плану та не порушувати власну стратегію. Концентрація грає визначальну роль у здатності виконувати завдання та досягати поставлених цілей. Здатність уважно фокусуватися на важливому завданні та утримувати фокус протягом тривалого періоду часу сприятиме підвищенню продуктивності та якісному виконанню робіт.

Один з ефективних методів поліпшення концентрації є використання методу Pomodoro. Цей підхід базується на ідеї поділу робочого часу на короткі інтенсивні періоди роботи по 25 хвилин, які називають "помідорами", короткими перервами по 5 хвилин та довгими перервами по 15 хвилин. Така система підтримує режим концентрованої праці, уникаючи втоми та забезпечуючи ефективність у виконанні завдань [4].

Pomodoro техніка дозволяє працювати у фокусованих інтервалах, що сприяє уникненню відволікань та забезпечує регулярні періоди відновлення енергії. Вона може бути особливо корисною в умовах великої кількості завдань або надмірної інформації, де важливо зберігати високий рівень концентрації. Такий підхід сприяє оптимізації часу та підвищенню ефективності роботи, допомагаючи досягати поставлених цілей з великою точністю та результативністю.

Автоматизація керування часом стає важливою складовою сучасного підходу до ефективного планування та відстеження завдань. Високий темп сучасного життя та постійний потік інформації вимагають не тільки ретельного планування, але й використання інструментів, які дозволяють автоматизувати цей процес. Різноманітні програми та додатки створюють можливість створювати розклади, нагадувати про важливі події та відстежувати часові рамки завдань.

Останні тенденції в розробці програм для керування часом свідчать про постійне прагненя до вдосконалення та розширення функціональності. Це дозволяє користувачам зручно об'єднувати планування робочого та особистого часу в єдиному інтерфейсі. Ледь не головною функцією є можливість встановлення пріоритетів та

категоризації завдань. Програми для керування часом дозволяють визначити важливість та терміни виконання завдань, щоб забезпечити оптимальне використання часу та досягнення ключових цілей. Деякі програми також пропонують аналітичні засоби для вивчення звичок та управління часом, сприяючи подальшій оптимізації робочого процесу [5].

У підсумку можна визначити важливість тайм-менеджменту як критичного фактора для досягнення успіху та гармонії у сучасному житті. В умовах високого темпу, постійних викликів та обсягу інформації, ефективне управління часом стає стратегічною навичкою. Відповідне планування та використання інструментів автоматизації надають можливість не лише раціонально розподіляти час, але і зосереджувати увагу на найважливіших завданнях.

1.2 Змістовний опис і аналіз предметної області

На практиці, мобільні додатки для контролю часу використовують різноманітні технічні реалізації для надання користувачам зручності та ефективності. Більшість з них пропонують інтуїтивно зрозумілі інструменти для створення, редагування та видалення завдань. Користувач може швидко додавати нові завдання, присвоювати їм терміни виконання та встановлювати пріоритети. Це робить процес планування більш організованим та контрольованим. Такий підхід сприяє візуалізації кількості роботи та допомагає відслідковувати прогрес.

Незважаючи на переваги і реалізації базових практик для тайм менеджменту, існують певні недоліки в сучасних мобільних додатках. Одним із найбільших недоліків є загальність підходу: багато програм пропонують універсальний функціонал, який може бути неефективним для ситуацій з реального життя. Технічні реалізації дозволяють скласти загальний план без конкретики та можливості розділення на різні сфери життя.

Додатковим недоліком є неможливість дійсно детального, похвилинного контролю часу: хоч користувач може створити план дій, додаток не завжди забезпечує ефективний механізм для сфокусування користувача на виконанні цього плану. Відсутність чіткого механізму контролю та підтримки концентрації може призводити до розсіювання уваги та зниження продуктивності [6].

Хоча існуючі мобільні додатки для тайм-менеджменту пропонують ряд технічних рішень для полегшення організації часу, їхні недоліки стають бар'єром для максимальної ефективності. Розуміння цих обмежень дозволить подальше вдосконалення і розвиток програм, спрямованих на ефективне управління часом користувачів.

1.3 Аналіз існуючих технологій та успішних IT-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільного застосунку для контролю особистого часу AnyTime. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

У контексті розробки мобільного додатка для контролю часу виникає кілька ключових проблем: необхідно визначити оптимальну платформу для розробки, сервіс для збереження даних, підхід до синхронізації даних між пристроями залежно від профілю користувача та навіть паттерн проєктування. Аналізуючи динаміку ринку та вподобання цільової аудиторії, варто зробити виважений вибір.

Розглядаючи різні платформи, можна виділити кілька найпопулярніших: Android, IOS, Windows Phone. Для розроблюваного застосунку важлива розповсюдженість операційнної системи, гнучкість, підтрмка та можливості для розробки.

Android володіє значною часткою ринку мобільних пристроїв, що дозволяє покрити широкий спектр пристроїв, на яких буде доступний розроблений додаток, підвищуючи його доступність. Відкритий характер операційної системи дозволить використати різноманітні API та інтегрувати різноманітні функціональні можливості в додаток. А за рахунок щорічних оновлень опеаційної системи та довготривалої підтримки старих версій, додаток розроблений для Android буде зберігати актуальність.

Під час вибору методу збереження даних користувача та їх синхронізації між пристроями важливо ретельно порівняти два розповсюджені підходи: використання

Room [7] для локального збереження та Firebase [8] для хмарної синхронізації. Room, як локальна база даних, забезпечує швидкий та ефективний доступ до інформації, що зберігається на пристрої. З іншого боку, Firebase відзначається безкоштовними квотами, що робить його привабливим варіантом для курсової роботи.

Однією з ключових переваг Firebase є можливість зберігання даних в нереляційній базі, що буде корисним для нашого додатку, який під час вдосконалення та розширенння функціоналу може змінювати структуру даних, що необхідно буде зберігати. Додатково, можливість поєднання автентифікації та сховища даних в одному сервісі робить його інтеграцію простішою та ефективнішою. Цей аспект особливо актуальний для доступності даних для авторизованих користувачів між пристоями.

При розробці андроїд додатків, особливо важливим є вибір ефективного паттерну проєктування, який сприятиме зручності управління даними та їх відображенням на інтерфейсі. Два з основних паттернів – MVVM (Model-View-ViewModel) та MVC (Model-View-Controller) – розглядаються як ключові для архітектури Android додатків.

MVC використовує розділення між моделлю, що представляє дані, видом, який відображає інформацію, і контролером, що обробляє взаємодію користувача. Недоліками цього підходу є те, що контролер часто стає перенасиченим та важким для тестування, а також проблеми зі збереженням стану інтерфейсу користувача під час повороту екрану. Натомість, MVVM вирішує ці проблеми, та педставляє більш чітке розділення та забезпечує однонаправленість потоку даних. ViewModel служить посередником між ними, дозволяючи легше тестувати та підтримувати код. Особливо важливим для Android додатків стає збереження стану інтерфейсу під час повороту екрану, і це завдання MVVM розв'язує ефективніше.

Отже, ми обрали основні технічні рішення для розробки мобільного застосунку AnyTime. Операційна система Android обрана як основна платформа з урахуванням широкого користувацького сприйняття та гнучкості для розробки. Для забезпечення надійності збереження даних та їх синхронізації обрано Firebase, який надає ефективний і хмарний механізм для цих цілей. Щодо паттерну проєктування,

вирішено використовувати MVVM для розділення логіки та інтерфейсу, забезпечуючи модульність та зручність в подальшому розвитку та тестуванні коду.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

У контексті розробки Android-застосунку, важливим етапом є вибір інтегрованого середовища розробки (IDE). При розгляді доступних опцій, Android Studio визначається як найбільш оптимальний вибір з урахуванням ряду факторів. Android Studio, розроблена компанією Google, надає розробникам повноцінне середовище з інтуїтивно зрозумілим інтерфейсом та потужними інструментами для відлагодження та розробки додатків [9].

Щодо мови програмування, Kotlin, від JetBrains обирається як перевага над Java через сучасність, безпеку та можливості, які дозволяють розробникам зменшити кількість коду та покращити читабельність. Kotlin є офіційною мовою для розробки Android-додатків та відзначається вищою продуктивністю [10].

У контексті бібліотек та фреймворків, Jetpack виявляється невід'ємною частиною екосистеми Android розробки. Jetpack пропонує багатий набір бібліотек та компонентів, що полегшують розробку, включаючи навігацію, роботу з базами даних, життєвий цикл компонентів та інше. Його активна підтримка Google забезпечує стабільність та оновлення, роблячи Jetpack зручним вибором для побудови надійних та сучасних Android-додатків.

1.3.3 Аналіз відомих програмних продуктів

Todoist [11] — це додаток для управління задачами, який надає користувачам можливість ефективно організовувати свою робочу та особисту діяльність. Зокрема, він пропонує інтуїтивний інтерфейс для створення, призначення та відстеження завдань. Крім того, Todoist дозволяє встановлювати терміни виконання, додавати пріоритети та використовувати хмарне сховище для синхронізації даних між різними пристроями.

TickTick [12] — це також додаток для управління завданнями, який дозволяє користувачам організовувати свій робочий та особистий час. Він пропонує широкий функціонал, включаючи можливість створювати різні списки завдань, встановлювати

нагадування, планувати події та спільно працювати над завданнями в команді. Крім того, TickTick надає аналітику використання часу для підвищення продуктивності.

Для порівняння курсової роботи з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Критерій	**Курсова робота (AnyTime)**	**Todoist**	**TickTick**	**Пояснення**
Можливість додавати завдання з описом, датою та відміткою про виконання	+	+	+	Основний функціонал додатку
Наявність фільтрації або групування завдань за категоріями	+	+	-	Швидкий доступ до категорій забезпечить більш гнучке розділення задач за сферами діяльності
Статистика по завданням	+	+	-	Статистика допоможе відстежувати прогрес та покращить досвід користувача
Pomodoro таймер або інший режим фокусування	+	-	-	Фокусування на виконанні завдань підвищить продуктивність
Безкоштовний доступ до всього функціоналу	+	-	-	Наявність спливаючих повідомлень зменшує бажання користуватись додатком

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є керування завданнями та режимом фокусування, більше функцій можна побачити на рисунку 1.1.

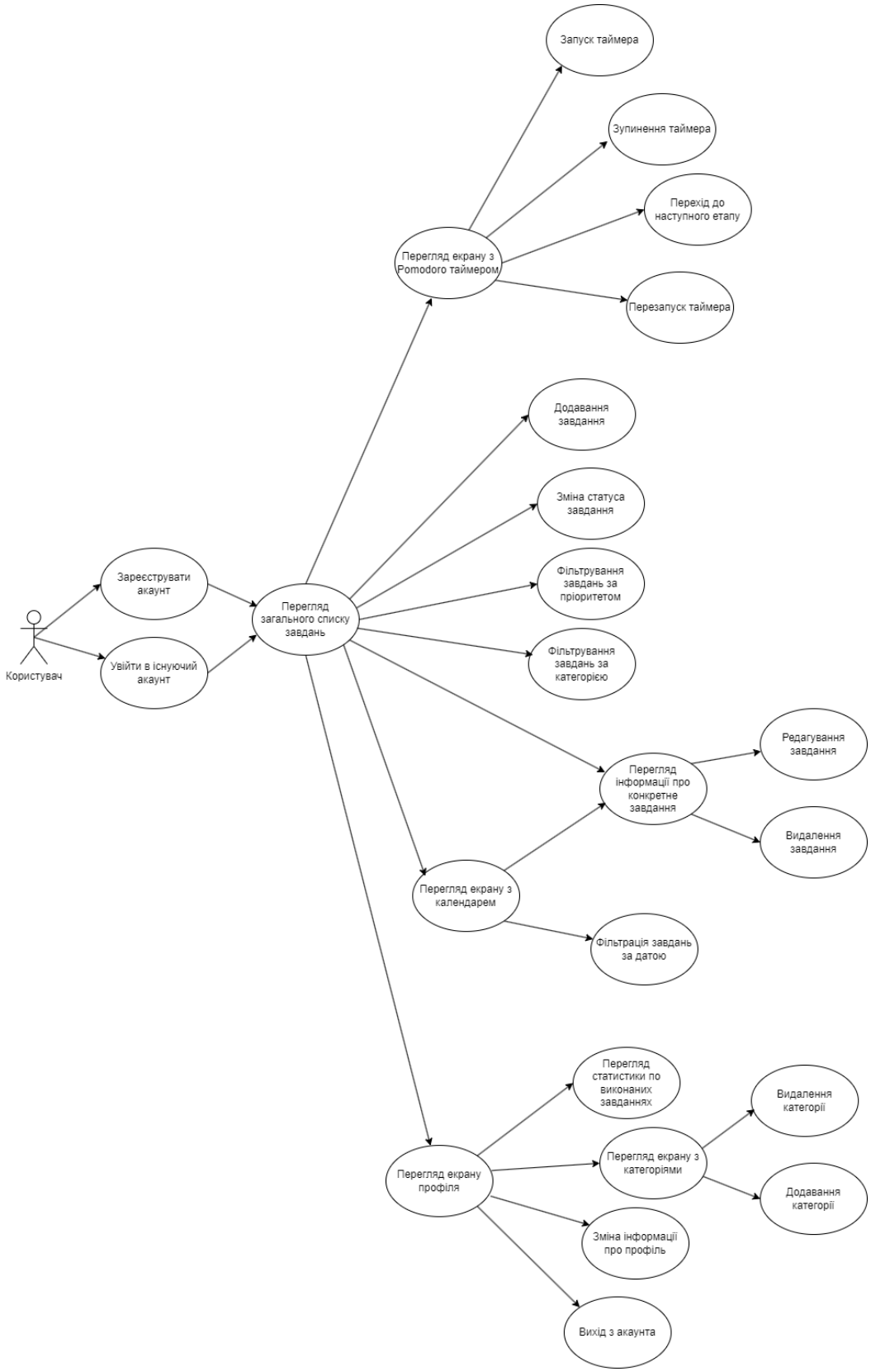

Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 - 1.13 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Створення нового облікового запису
Actors	Користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	Користувач не має облікового запису
Flow of Events	Користувач вводить електронну адресу, пароль та підтвердження паролю або реєструється за допомогою Google акаунту, натиснувши віповідну кнопку
Extension	У випадку введення не коректних даних у відповідних полях з’являється повідомлення про помилку.
Post-Condition	Створеня акаунту користувача, перехід до загального списку завдань

Таблиця 1.3 - Варіант використання UC-2

Use case name	Вхід в акаунт
Use case ID	UC-02
Goals	Увійт в існуючий обліковий запис
Actors	Користувач
Trigger	Користувач бажає зайти у свій акаунт
Pre-conditions	Існуючий акаунт
Flow of Events	Користувач вводить електронну адресу та пароль або здійснює вхід за допомогою Google акаунту, натиснувши віповідну кнопку
Extension	У випадку введення не коректних даних у відповідних полях з’являється повідомлення про помилку.
Post-Condition	Вхід до акаунту, перехід на сторінку входу

Таблиця 1.4 - Варіант використання UC-3

Use case name	Перегляд детальної інформації про завдання
Use case ID	UC-03
Goals	Побачити всю інформацію про завдання включаючи назву, статус, категорію, пріоритет, дату та опис

Actors	Користувач
Trigger	Користувач хоче дізнатись інформацію про завдання
Pre-conditions	Користувач здійснив вхід до акаунту Користувач має принаймні одне завданння.
Flow of Events	Користувач відкриває додаток та натискає на завдання із списку
Extension	-
Post-Condition	Перехід до екрану з інформацією про завдання

Таблиця 1.5 - Варіант використання UC-4

Use case name	Створення нового завдання
Use case ID	UC-04
Goals	Створити завдання
Actors	Користувач
Trigger	Користувач хоче додати завдання до списку
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач натискає на кнопку створення завдання на головному екрані, Користувач вводить інформацію про завдання та натискає «Create»
Extension	У випадку введення не коректних даних на екрані створення завдання у відповідних полях з’являється повідомлення про помилку.
Post-Condition	Завдання створено та відображається у списку

Таблиця 1.6 - Варіант використання UC-5

Use case name	Редагування завдання
Use case ID	UC-05
Goals	Змінити інформацію про завдання
Actors	Користувач
Trigger	Користувач хоче змінити завдання
Pre-conditions	Користувач здійснив вхід до акаунту Користувач має принаймні одне завданння.
Flow of Events	Користувач відкриває додаток та натискає на завдання із списку, Користувач натискає на кнопку «Edit» Користувач змінює інформацію в полях та натискає «Save»
Extension	У випадку введення не коректних даних на екрані створення завдання у відповідних полях з’являється повідомлення про помилку.

Post-Condition	Завдання відредаговано, користувач повертається до головного екрану

Таблиця 1.7 - Варіант використання UC-6

Use case name	Фільтрація завдань за датою
Use case ID	UC-06
Goals	Переглянути невиконані завдання заплановані на конкретну дату
Actors	Користувач
Trigger	Користувач хоче дізнатися завдання, які він ще не виконав сьогодні
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на екран календаря Користувач обирає дату
Extension	-
Post-Condition	Відображається список завдань, які відповідають даті та незавершені

Таблиця 1.8 - Варіант використання UC-7

Use case name	Редагування профілю
Use case ID	UC-07
Goals	Змінити зовнішній вигляд профілю користувача
Actors	Користувач
Trigger	Користувач бажає персоналізувати особисту сторінку
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на сторінку профіля. Користувач натискає кнопку «Edit». Користувач змінює нікнейм у текстовому полі та/або завантажує фото. Користувач натискає кнопку «Save».
Extension	У випадку, якщо користувач не зберіг зміни, повернути всі значення до початкових, після повернення на сторінку профіля
Post-Condition	Збереження та відображення нових даних на сторінці профілю

Таблиця 1.9 - Варіант використання UC-8

Use case name	Вихід з акаунту
Use case ID	UC-08
Goals	Вийти з облікового запису
Actors	Користувач

Trigger	Користувач бажає вийти з акаунту та перейти до екрану входу/реєстрації
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на сторінку профілю. Користувач натискає кнопку «Sign Out». Користувач підтверджує свою дію в діалоговому вікні.
Extension	-
Post-Condition	Вихід з акаунту без збереження даних на пристрої

Таблиця 1.10 - Варіант використання UC-9

Use case name	Додавання категорії
Use case ID	UC-09
Goals	Створити категорію
Actors	Користувач
Trigger	Користувач хоче додати категорію, щоб використовуватии при фільтрації, створенні та редагуванні завдань
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на сторінку профілю. Користувач натискає кнопку «Manage Categories» та переходить до екрану керування категоріями. Користувач натискає кнопку «Add category» Користувач вводить назву категорії та підтверджує створення
Extension	У випадку некоректного введення назви категорії у діалоговому вікні, з’являється напис про помилку
Post-Condition	Створення категорії та відображення її при фільтрації, створенні та редагуванні завдання

Таблиця 1.11 - Варіант використання UC-10

Use case name	Запуск Pomodoro таймера
Use case ID	UC-10
Goals	Запустити відлік таймера
Actors	Користувач
Trigger	Користувач хоче увімкнути режим фокусування
Pre-conditions	Користувач здійснив вхід до акаунту Користувач не має запущеного таймера
Flow of Events	Користувач переходить на сторінку фокусування Користувач натискає кнопку «Start»

Extension	-
Post-Condition	Початок зворотного відліку таймера та продовження його роботи після виходу користувача з додатку.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Перезапуск Pomodoro таймера
Use case ID	UC-11
Goals	Почати спочатку поточний етап роботи таймера
Actors	Користувач
Trigger	Користувач має на меті перезапустити таймер
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на сторінку фокусування Користувач натискає кнопку «Restart»
Extension	-
Post-Condition	Зупинка зворотного відліку таймера та переведення його до початкового значення для активного етапу.

Таблиця 1.13 - Варіант використання UC-12

Use case name	Пропуск етапа Pomodoro таймера
Use case ID	UC-12
Goals	Пропустити поточний етап таймера та перейти до наступного
Actors	Користувач
Trigger	Коистувач має на меті перейти до наступного етапу таймера
Pre-conditions	Користувач здійснив вхід до акаунту
Flow of Events	Користувач переходить на сторінку фокусування Користувач натискає кнопку «Next»
Extension	-
Post-Condition	Початок зворотного відліку таймера з початкового значення для наступного етапу та продовження його роботи після виходу користувача з додатку.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На таблиці 1.14 наведено загальну модель вимог, а в таблицях 1.15 –

1.33 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на таблиці 1.34.

Таблиця 1.14 – Загальна модель вимог

Опис	Назва	Пріоритет	Ризик
Реєстрація	FR-1	Середній	Високий
Авторизація	FR-2	Середній	Високий
Редагування профілю	FR-3	Низький	Середній
Вихід з профілю	FR-4	Середній	Високий
Перегляд списку завдань	FR-5	Високий	Високий
Перегляд детальної інформації про завданння	FR-6	Низький	Низький
Редагування завдання	FR-7	Середній	Середній
Створення завдання	FR-8	Середній	Середній
Перегляд списку категорій	FR-9	Середній	Низький
Додавання категорії	FR-10	Середній	Середній
Видалення категорії	FR-11	Середній	Середній
Фільтрування завдань за пріоритетом	FR-12	Високий	Середній
Фільтрування завдань за категорією	FR-13	Високий	Середній
Фільтрування завдань за датою та активним статусом	FR-14	Високий	Середній
Перегляд статистики по виконаних завданнях	FR-15	Низький	Низький
Початок відліку таймера	FR-16	Високий	Високий
Зупинка відліку таймера	FR-17	Високий	Високий
Перезапуск поточного стану таймера	FR-18	Високий	Середній
Пропуск поточного стану та запуск таймера	FR-19	Високий	Середній

Таблиця 1.15 – Функціональна вимога FR-1

Назва	Реєстрація

Опис	Система повинна надавати можливість реєстрації нового користувача шляхом введення пошти, паролю та підтвердження паролю або за гугл акаунтом.

Таблиця 1.16 – Функціональна вимога FR-2

Назва	Авторизація
Опис	Система повинна надавати можливість авторизації користувача шляхом введення пошти та паролю або за гугл акаунтом.

Таблиця 1.17 – Функціональна вимога FR-3

Назва	Редагування профілю
Опис	Система повинна надавати можливість змінювати юзернейм та фото профіля

Таблиця 1.18 – Функціональна вимога FR-4

Назва	Вихід з профілю
Опис	Система повинна надавати можливість вийти з акаунту.

Таблиця 1.19 – Функціональна вимога FR-5

Назва	Перегляд списку завдань
Опис	Система повинна надавати можливість переглядати весь список завдань

Таблиця 1.20 – Функціональна вимога FR-6

Назва	Перегляд детальної інформації
Опис	Система повинна надавати можливість переглядати всю інформацію про задання, включаючи дату, пріоритет та опис

Таблиця 1.21 – Функціональна вимога FR-7

Назва	Редагування завдання
Опис	Система повинна надавати можливість змінювати завдання, редагуючи будь-яке його поле.

Таблиця 1.22 – Функціональна вимога FR-8

Назва	Створення завдання

Опис	Система повинна надавати можливість створити завдання.

Таблиця 1.23 – Функціональна вимога FR-9

Назва	Перегляд списку категорій
Опис	Система повинна надавати можливість переглянути список збережених користувачем категорій.

Таблиця 1.24 – Функціональна вимога FR-10

Назва	Додавання категорії
Опис	Система повинна надавати можливість додавання категорії.

Таблиця 1.25 – Функціональна вимога FR-11

Назва	Видалення каткегорії
Опис	Система повинна надавати можливість видаляти категорії.

Таблиця 1.26 – Функціональна вимога FR-12

Назва	Фільтрування завдань за пріоритетом
Опис	Система повинна надавати можливість фільтрування завдань за пріоритетом.

Таблиця 1.27 – Функціональна вимога FR-13

Назва	Фільтрування завдань за категорією
Опис	Система повинна надавати можливість фільтрування завдань за категорією.

Таблиця 1.28 – Функціональна вимога FR-14

Назва	Фільтрування завдань за датою та активним статусом
Опис	Система повинна надавати можливість фільтрування завдань за датою та активним статусом.

Таблиця 1.29 – Функціональна вимога FR-15

Назва	Перегляд статистики по виконаних завданнях
Опис	Система повинна надавати можливість переглядати статистику по виконаних завданнях.

Таблиця 1.30 – Функціональна вимога FR-16

Назва	Початок відліку таймера

Опис	Система повинна надавати можливість почати відлік таймеру.

Таблиця 1.31 – Функціональна вимога FR-17

Назва	Зупинка відліку таймера
Опис	Система повинна надавати можливість зупинити відлік таймеру.

Таблиця 1.32 – Функціональна вимога FR-18

Назва	Перезапуск поточного стану таймера
Опис	Система повинна надавати можливість перезапустити таймер.

Таблиця 1.33 – Функціональна вимога FR-19

Назва	Пропуск поточного стану та запуск таймера
Опис	Система повинна надавати можливість пропустити поточний стан таймера.

Таблиця 1.33 – Матриця трасування вимог

	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9	UC 10	UC 11	UC 12
FR-1	+											
FR-2		+										
FR-3							+					
FR-4								+				
FR-5			+									
FR-6			+									
FR-7					+							
FR-8				+								
FR-9									+			
FR-10									+			
FR-11												
FR-12						+						
FR-13												
FR-14						+						
FR-15										+		
FR-16										+		
FR-17												
FR-18											+	
FR-19												+

1.4.2 Розроблення нефункціональних вимог

Розорблене рішення має працювати на пристроях з операційною системою Android версії 8.0 та вище, з об’ємом ОЗП мінімум 1 Гб та стабільним підключенням до Інтерету.

Додаток має забезпечувати безпеку даних шляхом використання захищеного каналу зв'язку для передачі даних користувача. Також, доступ до інформації

користувача в базі даних має бути можливий тільки від його імені. Це дозволить впевнитися, що конфіденційна інформація не потрапить в руки осіб, які не мають доступу до акаунту користувача.

Контроль введення інформації має здійснюватися шляхом валідації текстових полів, що захищатиме систему від некоректних дій коистувача та підтримуватиме точність даних.

Для забезпечення якості додатка використовуватиметься механізм відстеження аварійних ситуацій з Firebase Crashlytics. Цей інструмент дозволяє ефективно моніторити та аналізувати аварійні завершення роботи додатка на пристроях користувачів, щоб оперативно виявляти та усувати можливі несправності. Такий підхід підтримує стабільність та надійність роботи додатка в умовах реального використання.

1.5 Постановка задачі

Основною метою є розробка мобільного застосунку для контролю особистого часу, спрямованого на покращення існуючих рішень та надання розширених можливостей управління завданнями. Головні цілі проєкту можна описати трьома основними аспектами, кожен з яких вирішує конкретну проблему в області ефективного управління часом.

В першу чергу необхідно розробити основний функціонал для контролю завданнями та категоріями і синхронізації їх між пристроями за допомогою спільного облікового запису. Застосунок дозволятиме побудувати загальний план дій та дозволить вільно керувати задачами і відслідковувати їхнє виконання.

Після впровадження базового функціоналу, варто попроацювати над покращення фільтрації завдань. Багато існуючих рішень не надають ефективних інструментів для структурування та фільтрації завдань відповідно до конкретних критеріїв. Розроблений застосунок буде надавати можливості для фільтрації завдань як по категоріям, так і по пріоритету чи даті, що дозволить більш ефективно розмежовувати різні сфери життя та ефективно працювати кожною з них.

Наостанок, буде імплементовано Pomodoro таймер з метою підвищення продуктивності та концентрації. Це дозволить користувачу не лише будвати загальні

плани, а й більш детально керувати часом для підтримки оптимального режиму роботи.

Висновки до розділу

У цьому розділі було проведено аналіз предметної області, пов'язаної із тайм менеджментом та контролем особистого часу. Визначено основні принципи ефективного тайм менеджменту, зокрема акцентовано на необхідності систематичного контролю та планування завдань для досягнення максимальної продуктивності та досягнення поставлених цілей. Досліджено різноманітні готові програмні продукти, які вже існують на ринку, включаючи Todoist та TickTick.

Аналізуючи реалізацію існуючих технічних рішень, виявлено ряд недоліків та обмежень. Проаналізовані програми не забезпечують достатню гнучкість у фільтрації завдань та сприянні фокусування користувача на виконанні поставлених цілей.

Після проведеного аналізу технічних рішень та існуючого програмного забезпечення, визначено стратегію для розробки власного мобільного додатку для контролю особистого часу. Сформульовано функціональні та нефункціональні вимоги, враховуючи виявлені проблеми та недоліки існуючих рішень. Поставлено конкретні завдання, спрямовані на вирішення виявлених проблем та забезпечення оптимального функціоналу та ефективності майбутнього додатку.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

В контексті застосунку для контролю особистого часу, бізнес-процеси визначаються як послідовність дій та операцій, спрямованих на досягнення основних цілей та функцій додатку. Зазначимо кілька ключових бізнес-процесів, які можуть виявитися важливими в рамках розробки нашого додатку:

Опис послідовності створення облікового запису користувача:

- користувач запускає додаток;
- користувач обирає опцію створення нового акаунту або авторизується за допомогою Google OAuth;
- користувач вводить необхідні дані в поля, залежно від обраного способу.

Опис послідовності зміни інформації у профілі користувача:

- користувач переходить на сторінку профіля;
- користувач натискає на кнопку Edit;
- користувач вводить необхідні дані та завантажує файли;
- користувач підтверджує зміни натисканням на кнопку Save.

Опис послідовності додавання нової задачі до списку:

- користувач натискає на кнопку + на головому екрані;
- користувач вводить дані та підтверджує створення завдання.

Опис послідовності дій для фільтрації задач:

- користувач переходить на головну сторінку;
- користувач обирає потрібний пріоритет та/або категорію завдань у відповідних полях.

Опис послідовності дій для видалення категорії:

– користувач переходить на сторінку управління категоріями;

– користувач обирає категорію зі списку та натискає на кнопку Delete а відповідному елементі списку;

Опис послідовності дій для запуску таймера:

– користувач переходить на сторінку фокусування;

– користувач натискає на кнопку запуску таймеру та отримує сповіщення про роботу таймера.

Для кращого опису бізнес процесів програмного забезпечення використаємо BPMN модель найважливіших функцій застосунку: загального опису фільтрації завдань (рисунок 2.1) та опису оновлення стану таймера (рисунок 2.2)

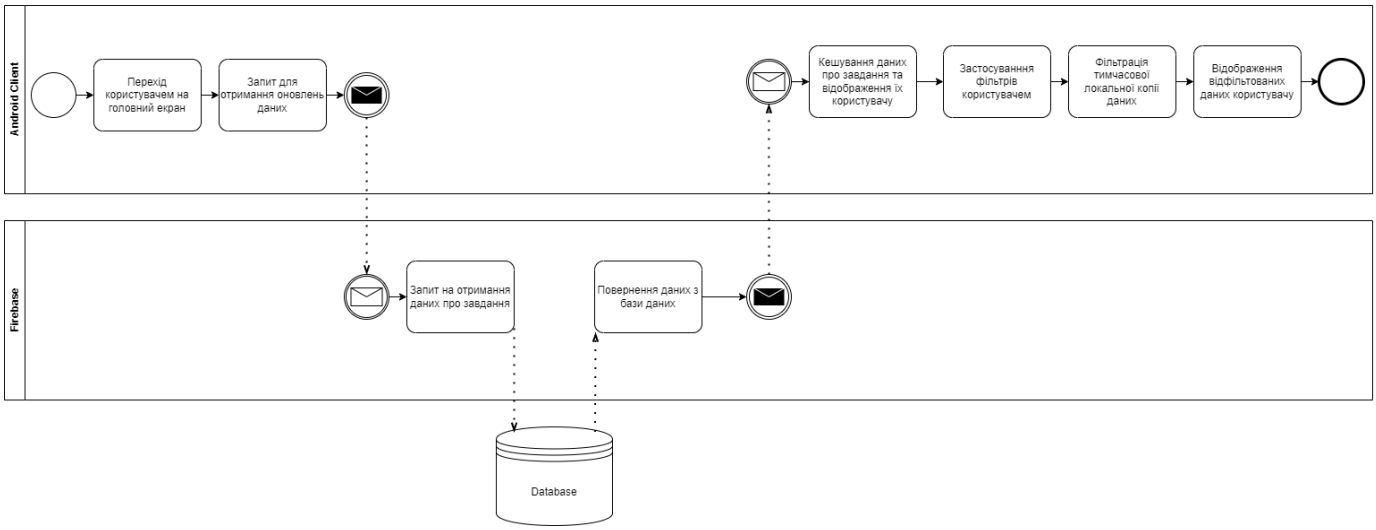

Рисунок 2.1 – BPMN модель фільтрації завдань

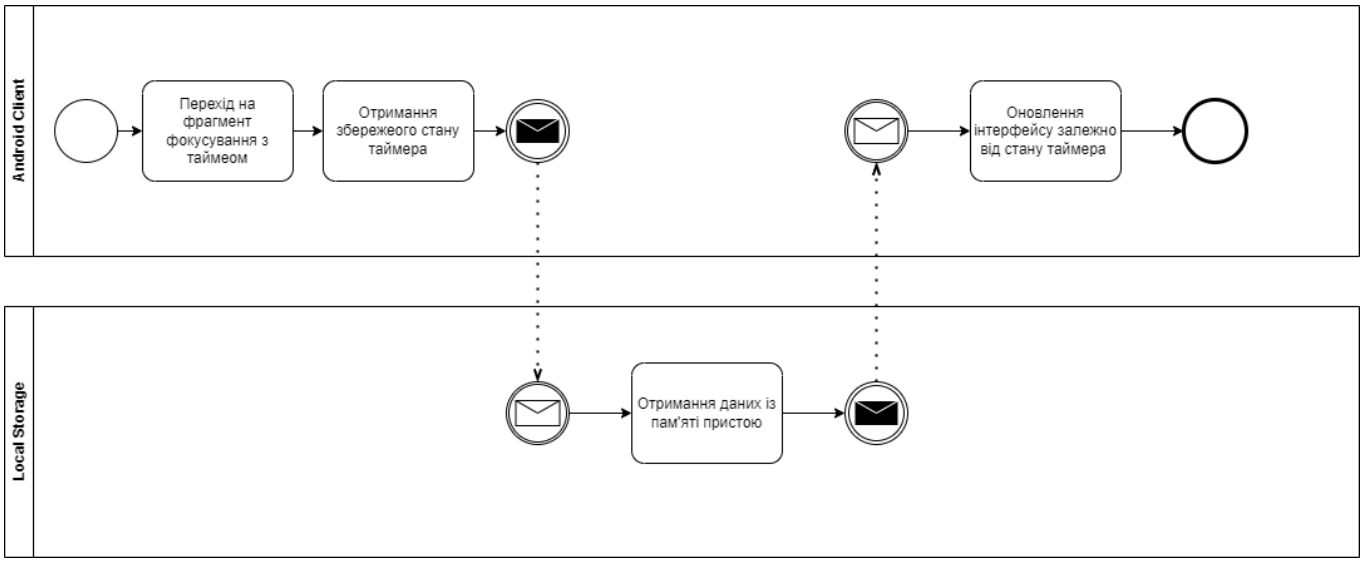

Рисунок 2.2 – BPMN модель оновлення стану Pomodoro таймера

2.2 Архітектура програмного забезпечення

В розробленому Android додатку застосована клієнт-серверна архітектура, що передбачає взаємодію між застосунком та Firebase для виконання запитів на отримання даних та авторизації користувача.

Архітектурно, використовується патерн MVVM (Model-View-ViewModel), що є рекомендованим Google [13] патерном архітектури для розробки Android-додатків, який забезпечує чітке розділення між логікою бізнес-логіки, інтерфейсом користувача та представленням даних.

В основі MVVM лежить концепція односпрямованості даних, що означає, що дані можуть переміщуватися лише в одному напрямку між компонентами патерну. Модель (Model) відповідає за управління даними та бізнес-логікою, Представлення (View) відповідає за відображення інтерфейсу користувача, а В'юМодель (ViewModel) служить посередником, який обробляє дані з Моделі та забезпечує їх відображення у Представленні.

Переваги MVVM включають в себе покращену структурованість та підтримуваність коду. Також цей патерн дозволяє використовувати переваги такої функціональності як StateFLow та SharedFlow в Android, що автоматично взаємодіє з життєвим циклом компонентів Android, таких як Activity та Fragment, спрощуючи управління станом додатку.

Для реалізації ін'єкції залежностей використовується Hilt, який забезпечує автоматичне створення та управління залежностями в різних частинах додатку. Це спрощує процес впровадження та збереження класів. Спрощена діаграма, яка опускає багато деталей, але відображає суть реалізованої архітектури і паттерна, зображена на Рисунку 2.3.

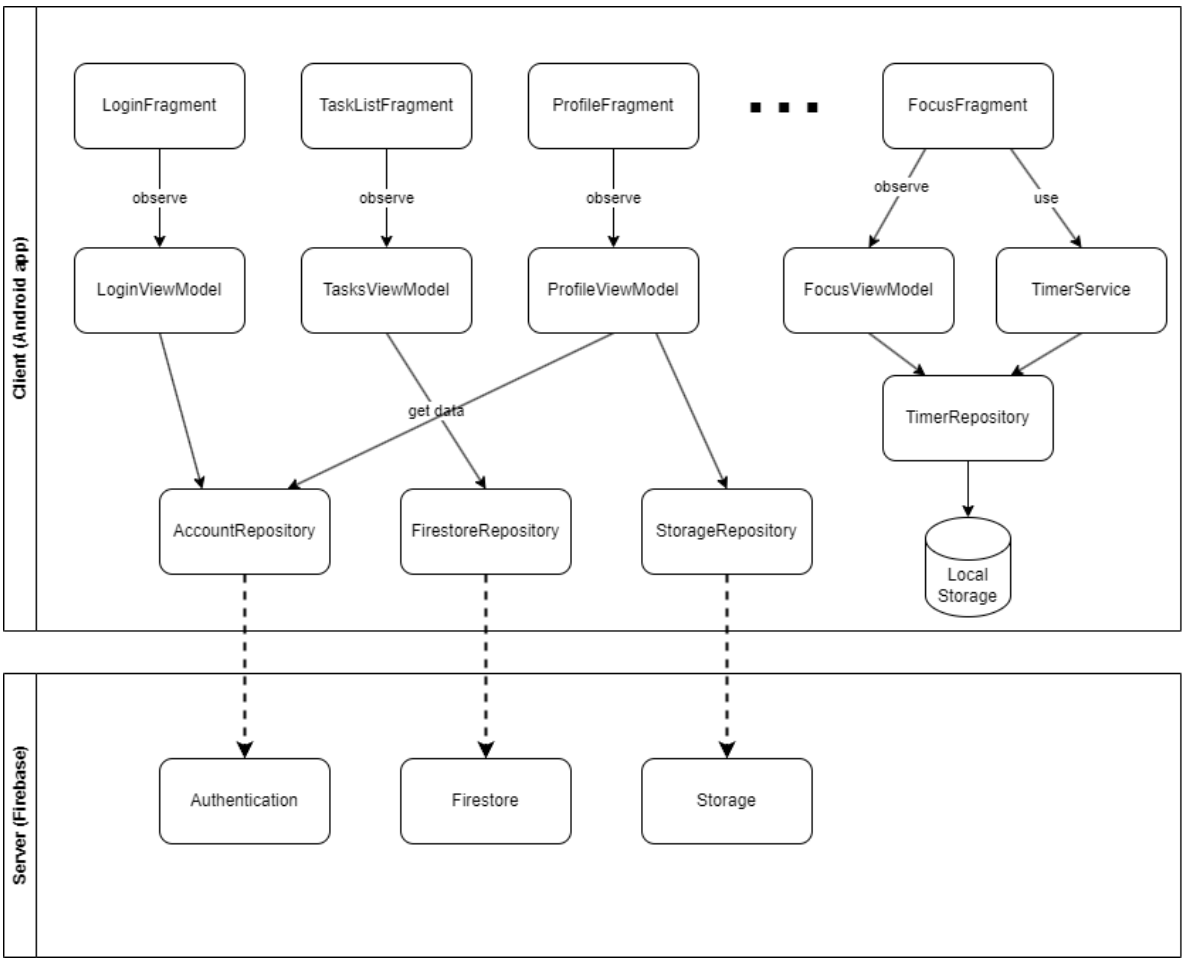

Рисунок 2.2 – Архітектура застосунку

2.3 Конструювання програмного забезпечення

У розробці мого мобільного додатку було здійснено використання ряду ключових бібліотек, спрямованих на покращення якості та функціональності додатку. Однією з основних бібліотек, яку я використав, є Material Design Library, надана Google. Ця бібліотека дозволяє ефективно реалізувати дизайн відповідно до принципів Material Design, забезпечуючи єдність та сучасний вигляд інтерфейсу користувача.

Для завантаження та відображення зображень в додатку була використана бібліотека Coil. Вона надає зручний та продуктивний механізм роботи з

зображеннями, автоматично вирішуючи завдання їхнього завантаження та кешування.

Щодо аутентифікації користувачів, використано Google OAuth бібліотеку, яка спрощує взаємодію з сервісом Google для безпечного та зручного входу користувачів у додаток.

Для реалізації навігаційної структури в додатку використані navigation components. Ця бібліотека надає зручні інструменти для управління фрагментами та активностями, спрощуючи навігацію між різними частинами додатку.

Hilt виступає в ролі бібліотеки для внедрення залежностей і графа залежностей. Вона забезпечує ефективну ін'єкцію залежностей у додатку та допомагає у збереженні чистоти та організації коду за стандартами сучасної розробки Android-додатків.

В моєму мобільному додатку для зберігання та управління даними використовується база даних Firestore, яка є частиною екосистеми Google Cloud та надає гнучкість та масштабованість для роботи з даними в режимі реального часу. Обрана структура бази даних детально відповідає вимогам та функціональності мого додатку, забезпечуючи ефективне зберігання та отримання інформації.

База даних містить основну колекцію "users", в якій кожен документ ідентифікується унікальним id користувача. Це дозволяє впорядковано та ефективно взаємодіяти з інформацією, пов'язаною з кожним конкретним користувачем. У документі користувача присутнє поле "categories", яке є масивом рядкових значень, відображаючи категорії завдань, які користувач створив. Додатково, внутрішнім елементом документа є вкладена колекція "tasks". Кожен документ цієї колекції представляє конкретне завдання та містить поля, що деталізують його характеристики. Зокрема, "completed" вказує на стан завдання (виконане чи ні), "title" містить заголовок завдання, "category" - категорію, "priority" - пріоритет, "date" - дату та "description" - опис. Така структура бази даних дозволяє зручно та ефективно управляти завданнями користувачів, забезпечуючи необхідний рівень нормалізації даних та швидкий доступ до інформації.

Опис документів бази даних наведено у таблицях 2.1 та 2.2.

Таблиця 2.1 – Опис документа user

Таблиця	Назва поля	Тип даних	Опис
user	tasks	Collection	Вкладена колекція документів, структура яких описана у таблиці 2.2
	categories	Array<String>	Список категорій користувача

Таблиця 2.2 – Опис документа task

Таблиця	Назва поля	Тип даних	Опис
task	completed	Boolean	Позначає чи виконано завдання
	title	String	Назва завдання
	category	String	Категорія
	priority	Int	Пріоритет
	date	String	Дата у форматі dd.MM.yyyy
	description	String	Опис завдання, може містити пусте значення

Опис утиліт, що використовувалися у розробці наведено в таблиці 2.3.

Таблиця 2.3 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування

1	Android Studio	Основне інтегроване середовище розробки (IDE) для платформи Android.
2	Firebase Console	Консоль, яка забезпечує зручний інтерфейс для перегляду, внесення змін та відстеження різноманітних аспектів роботи додатку в реальному часі.
3	Git та GitHub	Система контролю версій та система керування віддаленими репозиторіями, які допомогали зберігати зміни в коді та ділитись файлами курсової роботи з керівником.

2.4 Аналіз безпеки даних

Додаток забезпечує високий рівень безпеки передачі даних між Android клієнтом та Firebase, використовуючи захищений канал зв'язку для надійного обміну інформацією користувача. Забезпечено, що доступ до особистої інформації користувача в базі даних можливий лише від його імені, гарантуючи, що конфіденційна інформація залишається виключно під контролем авторизованого користувача.

Крім того, безпека додатка забезпечується контролем введення інформації через валідацію текстових полів. Цей підхід не лише захищає систему від некоректних дій користувача, але й підтримує високу точність та надійність даних, які обробляються в додатку. Такий комплекс заходів сприяє збереженню конфіденційності та безпеки обробки особистої інформації.

Висновки до розділу

У даному розділі було розглянуто процес моделювання програмного забезпечення для застосунку. Основною метою моделювання було визначення

ефективних та оптимальних бізнес-процесів, які забезпечать високий рівень функціональності та зручність користування для кінцевих користувачів.

Архітектурний підхід, обраний при розробці застосунку, ґрунтувався на принципах гнучкості та простоти. Забезпечено розділення бізнес-логіки, інтерфейсу та доступу до даних, що сприяє підтримці та розвитку системи. Архітектура була спроектована таким чином, щоб забезпечити високу продуктивність та ефективність додатку.

У роботі детально розглянуті використані бібліотеки та інструменти, які визначили технічний стек розробки. Обрані бібліотеки сприяють реалізації функціональності та підвищенню продуктивності розробки.

Структура бази даних була розроблена для забезпечення ефективного зберігання та обробки інформації про завдання. Визначено заходи забезпечення конфіденційності, цілісності та доступності інформації. Розроблені заходи мають на меті запобігання несанкціонованому доступу та збереження цілісності важливих даних користувачів.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Оцінка якості коду додатку була проведена за допомогою Detekt [14], інструменту статичного аналізу коду. Аналіз охоплював різні метрики, що дають уявлення про структурні аспекти кодової бази, результат можна побачити на Рисунку 3.1. Додаток складається з 41 класу, розподіленого по 14 пакетах, із загальною кількістю в 101 поле та 132 функції. У звіті про складність вказано 2 291 рядок коду, з яких 1 147 - це логічні рядки коду. Цикломатична складність вимірюється на рівні 271, а когнітивна складність - 151.

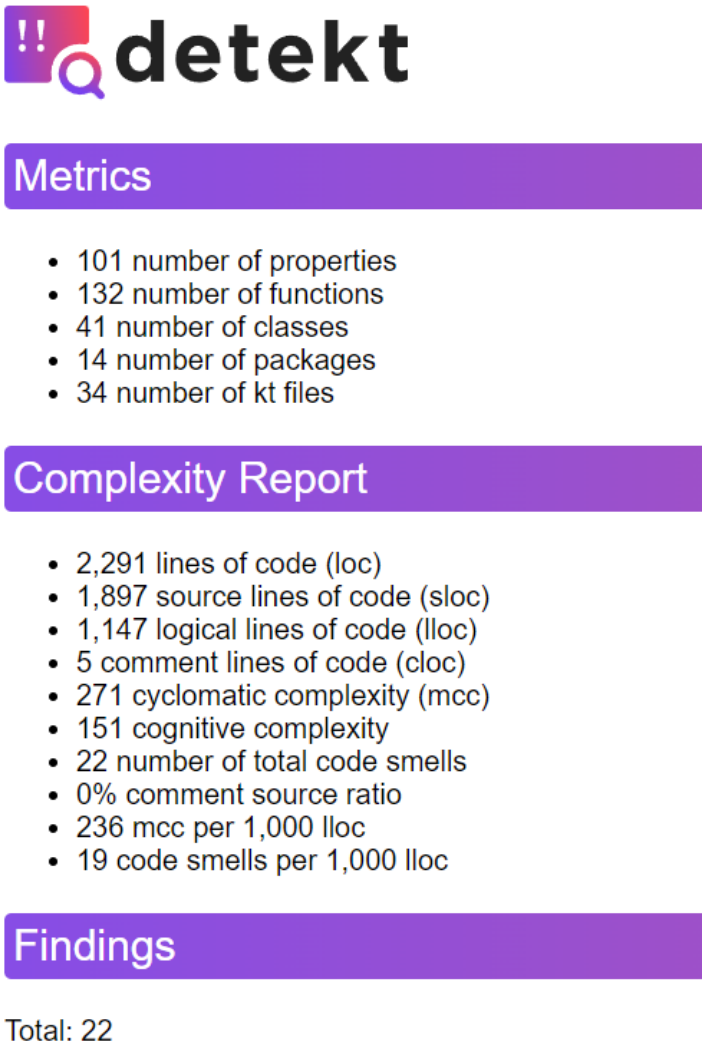

Рисунок 3.1 – Результат аналізу коду

З точки зору code smells, аналіз виявив загалом 22 проблеми. Ці результати охоплюють різні аспекти, включаючи потенційну надмірність коду, проблеми з

підтримкою та відхилення від найкращих практик. Надані зауваження в основному представляють собою рекомендації щодо стилістики, оптимізації та дотримання кращих практик програмування. Вони не є критичними помилками, які призведуть до справжніх несправностей у роботі програми. Проте, аналіз показав потенційні можливості для поліпшення читабельності, підтримки та масштабованості коду.

3.2 Опис процесів тестування

У даному контексті, для тестування основних функцій додатку було вибрано мануальний підхід. Це обрано на основі складності та унікальності функціональності, яку пропонує додаток для контролю особистого часу. Мануальне тестування дозволяє здійснити глибокий та детальний аналіз різноманітних сценаріїв використання, виявити потенційні вразливості та забезпечити високий рівень покриття тестами.

Результати мануального тестування будуть представлені у вигляді тест-кейсів, що детально описують кроки для відтворення конкретного тестового сценарію та очікувані результати. Цей підхід дозволить не лише забезпечити якісне функціонування основних функцій додатку, але й створити підґрунтя для подальших випробувань та вдосконалення продукту. Опис відповідних тестів наведено у таблицях 3.1 – 3.10.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	Натискаємо на кнопку «Create new account». У відповідні поля вводимо: коректну електронну пошту, яка до цього не була зареєстрована в системі, пароль від 8 символів, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискаємо кнопку «Sign Up»

Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на головну сторінку.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на головну сторінку.

Таблиця 3.2 – Тест 2.1

Тест	Зміна статусу завдання на головному екрані
Модуль	Список завдань
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на головній сторінці і має хочаб одне невиконане завдання у списку
Вхідні данні	Відсутні
Опис проведення тесту	Обираємо завдання та натискаємо на чекбокс
Очікуваний результат	Статус завдання змінився на виконане, зміна статусу відобразилася у чекбоксі, завдання у списку перемістилося нижче за всі невиконані, якщо такі є
Фактичний результат	Статус завдання змінився на виконане, зміна статусу відобразилася у чекбоксі, завдання у списку перемістилося нижче за всі невиконані

Таблиця 3.3 – Тест 2.2

Тест	Фільтація завдань за категорією
Модуль	Список завдань
Номер тесту	2.2
Початковий стан системи	Користувач знаходиться на головній сторінці і має хочаб одне невиконане завдання у списку
Вхідні данні	Відсутні

Опис проведення тесту	У відповідному полі у верхній частині екрану замість значення «All categories» обираємо довільну категорію зі списку.
Очікуваний результат	Зі списку зникли усі завдання, що не відповідають обраній категорії
Фактичний результат	Зі списку зникли усі завдання, що не відповідають обраній категорії

Таблиця 3.4 – Тест 3.1

Тест	Робота Pomodoro таймера у фоновому режимі
Модуль	Фокус
Номер тесту	3.1
Початковий стан системи	Користувач знаходиться на екрані фокусування
Вхідні данні	Відсутні
Опис проведення тесту	Запускаємо таймер натисканням на відповідну кнопку та закиваємо додаток
Очікуваний результат	З’явилось беззвучне сповіщення від додатку зі зворотнім відліком таймера, який продовжує працювати
Фактичний результат	З’явилось беззвучне сповіщення від додатку зі зворотнім відліком таймера, який продовжує працювати

Таблиця 3.5 – Тест 3.2

Тест	Автоматичне продовження Pomodoro таймера
Модуль	Фокус
Номер тесту	3.2
Початковий стан системи	Користувач знаходиться на екрані фокусування
Вхідні данні	Відсутні

Опис проведення тесту	Запускаємо таймер натисканням на відповідну кнопку та чекаємо поки закінчиться поточний стан
Очікуваний результат	Додаток надіслав сповіщення про закінчення поточного стану таймера та автоматично почав зворотній відлік наступного стану
Фактичний результат	Додаток надіслав сповіщення про закінчення поточного стану таймера та автоматично почав зворотній відлік наступного стану

Таблиця 3.6 – Тест 4.1

Тест	Видалення завдання з екрану календаря
Модуль	Календар
Номер тесту	4.1
Початковий стан системи	Користувач знаходиться на екрані календаря та має хоч одне незавершене завдання
Вхідні данні	Відсутні
Опис проведення тесту	Обираємо дату, на яку є заплановані невиконані завдання. Натискаємо на елемент списку. У вікні інформації про завдання натискаємо на кнопку «Delete».
Очікуваний результат	Завдання видалилось, користувач повернувся на екран календаря
Фактичний результат	Завдання видалилось, користувач повернувся на екран календаря

Таблиця 3.7 – Тест 5.1

Тест	Видалення всіх категорій
Модуль	Керування категоріями
Номер тесту	5.1

Початковий стан системи	Користувач знаходиться на екрані керування категоріями та має лише одну збережену категорію
Вхідні данні	Відсутні
Опис проведення тесту	Натискаємо кнопку «Delete» біля останньої категорії
Очікуваний результат	Категорія не видалилась, вивелось попередження про неможливість видалити всі категорії
Фактичний результат	Категорія не видалилась, вивелось попередження про неможливість видалити всі категорії

Таблиця 3.8 – Тест 5.2

Тест	Створення вже існуючої категорії
Модуль	Керування категоріями
Номер тесту	5.2
Початковий стан системи	Користувач знаходиться на екрані керування категоріями
Вхідні данні	Назва ісуючої категорії
Опис проведення тесту	Натискаємо кнопку «Add category» та вводимо зарезервовану назву у поле.
Очікуваний результат	Категорія не створилась, вивелось повідомлення про помилку
Фактичний результат	Категорія не створилась, вивелось повідомлення про помилку

Таблиця 3.9 – Тест 6.1

Тест	Перевірка коректності статистики
Модуль	Профіль
Номер тесту	6.1

Початковий стан системи	Користувач знаходиться на екрані профілю
Вхідні данні	Відсутні
Опис проведення тесту	Запам’ятовуємо кількість виконаних та невиконаних завдань. Переходимо на головний екран, натискаємо та чекбокс будь-якого невиконаного завдання. Повертаємось до екрану профіля
Очікуваний результат	Значення виконаних завдань збільшилось на 1, значення невиконаних зменшилось на 1
Фактичний результат	Значення виконаних завдань збільшилось на 1, значення невиконаних зменшилось на 1

Таблиця 3.10 – Тест 6.2

Тест	Вихід з акаунту
Модуль	Профіль
Номер тесту	6.2
Початковий стан системи	Користувач знаходиться на екрані профілю
Вхідні данні	Відсутні
Опис проведення тесту	Натискаємо кнопку «Sign Out» та підтверджуємо свою дію у діалоговому вікні
Очікуваний результат	Користувач вийшов із акаунта та перейшов на екран авторизації
Фактичний результат	Користувач вийшов із акаунта та перейшов на екран авторизації

Висновки до розділу

У цьому розділі було проведено високорівневий аналіз ключових аспектів розробленого додатку для контролю особистого часу. Оцінка якості коду,

використовуючи інструмент Detekt, розкрила важливі аспекти структури та ефективності кодової бази. Зауваження, виявлені під час аналізу, сконцентровані переважно на стилістиці, оптимізації та відповідності кращим практикам програмування.

Мануальне тестування основних функцій додатку виокремило сценарії використання та ідентифікувало потенційні проблеми, сприяючи вдосконаленню якості та надійності програмного продукту. Результати тестів у вигляді тест-кейсів нададуть вичерпну картину відтворення різних сценаріїв та перевірки відповідності функціональності вимогам.

Підтвердження функціональності додатку виявило його високу ефективність та відповідність поставленим завданням. Заходи, вжиті для виправлення зауважень та покращення кодової бази, сприятимуть подальшому розвитку та суттєвому підвищенню якості продукту. В цілому, виконані оцінка якості коду та тестування забезпечили не лише виявлення потенційних проблем, а й визначення конструктивних кроків для подальшого вдосконалення та успішної експлуатації додатку для контролю особистого часу.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Процес розгортання розробленого застосунку для контролю особистого часу досить простий та зручний. Кодова база та готовий apk файл знаходяться у відкритому доступі на GitHub репозиторії проекту (рисунок 4.1). Для установки додатку, користувачу необхідно виконати кілька кроків.

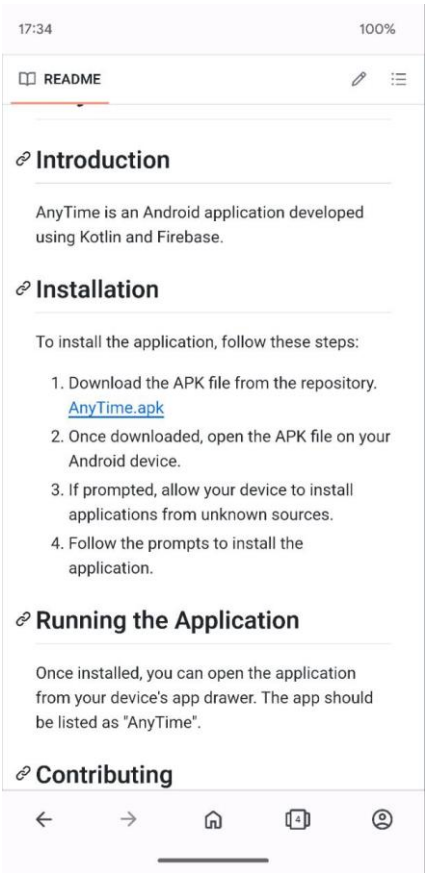

Рисунок 4.1 – Репозиторій з інструкцією по встановленню

У першу чергу, користувач повинен завантажити apk файл із GitHub репозиторію. Це може бути виконано шляхом клонування репозиторію або завантаження окремого файлу на пристрій (рисунок 2.2).

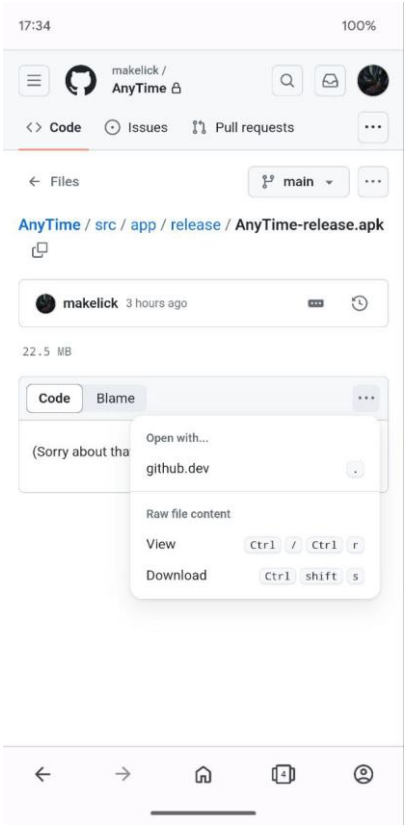

Рисунок 4.2 – Завантаження .apk файлу

Після завантаження та налаштування, користувач може запустити завантажений apk файл та вибрати опцію "Встановити" (рисунок 2.3). На деяких пристроях доведеться надати дозвіл на встановлення додатків з невідомих джерел. Для цього необхідно знайти та увімкнути необхідну функцію у розділі "Налаштування" або "Безпека" пристрою.

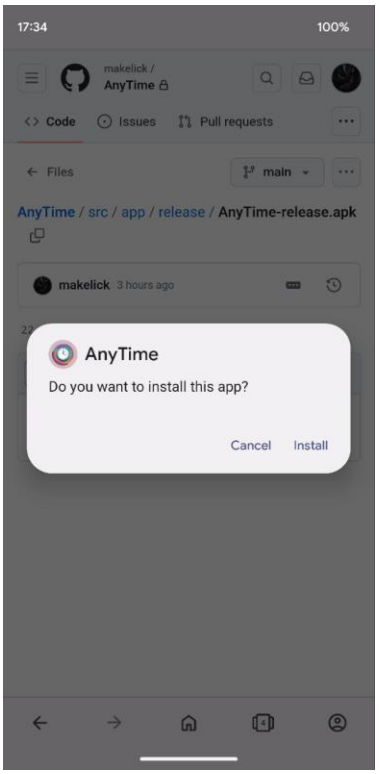

Рисунок 4.3 – Встановлення .apk файлу

Процес встановлення додатку відбудеться автоматично. Після успішного завершення установки та авторизації користувач зможе скористатися всім функціоналом застосунку AnyTime на своєму пристрої (рисунок 2.4).

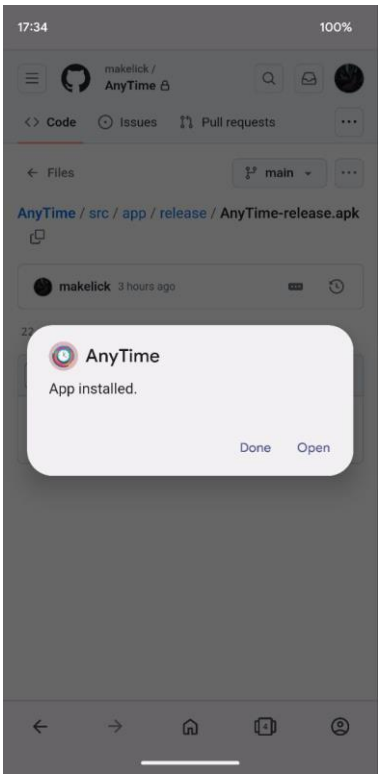

Рисунок 4.4 – Успішне встановлення

4.2 Підтримка програмного забезпечення

Розроблений застосунок для контролю особистого часу розглядається як проект з відкритим вихідним кодом, що надає можливість будь-якій зацікавленій особі редагувати та дороблювати його відповідно до власних потреб. Завантаження та редагування кодової бази є легким завданням: достатньо клонувати весь репозиторій з GitHub та відкрити його в Android Studio. Цей процес надає можливість розробникам отримати доступ до повного коду, щоб вносити зміни, доповнювати функціонал та адаптувати застосунок під свої потреби.

Також важливо відзначити, що застосунок підтримується та відкритий для подальших розширень. Після внесення необхідних змін та допрацювань, розроблений додаток може бути легко завантажений до Google Play Market. Це покращить процес доставки оновлень та зробить застосунок доступним для більшої аудиторії користувачів.

Такий підхід не лише надає можливість спільноті розробників взаємодіяти та спільно працювати над вдосконаленням продукту, але й підтримує довгостроковий розвиток та рост додатку завдяки активній участі громадськості та постійному оновленню його функціональності.

Висновки до розділу

В розділі були визначені ключові аспекти, спрямовані на максимальну зручність та доступність для користувачів та розробників. За допомогою відкритого вихідного коду на GitHub репозиторії, будь-яка зацікавлена особа може легко клонувати та редагувати кодову базу, щоб адаптувати застосунок під свої власні потреби та вимоги. Крім того, будь хто може завантажити та скористатися готовим apk файлом з готовим рішенням.

У розділі були описані чіткі та прості інструкції, що забезпечують ефективний процес встановлення та використання додатку, забезпечуючи кінцевим користувачам легкий доступ до розробленої функціональності.

Загалом, цей розділ визначає ключові аспекти, спрямовані на розширення та підтримку додатку, забезпечуючи зручність для всіх сторін - від розробників до кінцевих користувачів.

ВИСНОВКИ

Розроблений мобільний додаток для контролю особистого часу визначається як результат вдалої реалізації усіх поставлених завдань та цілей наукової роботи. Додаток успішно впроваджує ідею ефективного управління часом, пропонуючи користувачам інтегрований інструмент для кращого контролю над своїм розкладом та завданнями.

Результат курсовою роботи відзначається своєю універсальністю та можливостю використання в різних сферах. Можливості фільтрації завдань та використання Pomodoro таймера надають широкий спектр інструментів для досягнення високої продуктивності та ефективного управління часом.

Розроблений додаток надає конкретні рішення для поліпшення ефективності та організації часу користувачів, що є актуальним у сучасному ритмі життя. Результати наукової роботи підкреслюють важливість розробки інноваційних технологій для поліпшення якості життя та підвищення особистої продуктивності.

Основним інструментом для створення додатку стало середовище Android Studio. Вибір мови програмування Kotlin визначився його прогресивними можливостями та високою сумісністю з платформою Android, що сприяє створенню чистого та ефективного коду.

Для забезпечення потреб додатку у зберіганні та обробці даних було використано Firebase, що дозволило реалізувати нереляційну базу даних та інші функції з високою надійністю та швидкістю. Використання Firebase забезпечило плавну синхронізацію даних між різними пристроями та безпеку інформації.

Розроблений застосунок було протестовано на різних смартфонах з різними характеристиками та версіями Android. Результати тестування підтверджують стабільну та ефективну роботу додатку на різних пристроях, що підкреслює його універсальність та готовність до використання в різних умовах.

У першу чергу, у моєму додатку для контролю особистого часу вперше був реалізований базовий функціонал, такий як створення та відстеження завдань, встановлення та використання Pomodoro таймера для оптимізації робочого часу. Ці

основні елементи надають користувачам зручний інструмент для планування та ефективного використання свого часу.

Під час розвитку додатку були внесені модифікації до функціоналу, що включає в себе поліпшення фільтрації завдань, розширення можливостей редагування та синхронізацію між пристроями. Ці зміни були спрямовані на поліпшення зручності та гнучкості користування додатком, а також на врахування побажань користувачів та оптимізацію робочого процесу.

Функціонал додатку продовжує набирати розвиток, додаючи нові можливості та опції відповідно до вимог сучасного користувача. Додаткові інструменти для аналізу продуктивності, покращені функції сповіщень та інші функції плануються для майбутнього розвитку, щоб надати користувачам ще більше інструментів для ефективного управління часом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Time Management Is About More Than Life Hacks // Harvard Business Review, Erich C. Dierdorff. URL: https://hbr.org/2020/01/time-management-is-about-more-than-life-hacks (дата звернення 29.12.2023).
2) Five Strategies For Improving Work-Life Balance // Forbes, Erik Pham. URL: https://www.forbes.com/sites/forbesbusinesscouncil/2023/02/27/five-strategies-for-improving-work-life-balance/?sh=1c7c7669a745 (дата звернення 29.12.2023).
3) The power of visualization | How to use visualization to achieve your goals // Medium, Amelia Stewards. URL: https://medium.com/@seventysuccesssecrets/the-power-of-visualization-how-to-use-visualization-to-achieve-your-goals-d38a053b2e30 (дата звернення 29.12.2023).
4) Boost Productivity with the Pomodoro Technique: Time Management Mastery // Everhour, Mike Kulakov. URL: https://everhour.com/blog/pomodoro-technique/ (дата звернення 29.12.2023).
5) Time Management and Productivity in the Modern Workplace // Day.io, Julia Neves. URL: https://day.io/blog/time-management-and-productivity-in-the-modern-workplace-an-in-depth-analysis/ (дата звернення 29.12.2023).
6) Why Traditional Time Management Apps Don't Work // Motion blog. URL: https://www.usemotion.com/blog/time-management-apps (дата звернення 29.12.2023).
7) Room // Android Developers. URL: https://developer.android.com/jetpack/androidx/releases/room (дата звернення 29.12.2023).
8) Firebase. URL: https://firebase.google.com (дата звернення 29.12.2023).
9) Android Studio // Android Developers. URL: https://developer.android.com/studio (дата звернення 29.12.2023).
10) Kotlin Programming Language // Kotlin official site blog. URL: https://kotlinlang.org (дата звернення 29.12.2023).
11) Todoist. URL: https://todoist.com/?locale=en (дата звернення 29.12.2023).
12) TickTick. URL: https://ticktick.com/?language=en_us (дата звернення 29.12.2023).

13) Recommendations for Android architecture // Android Developers. URL: https://developer.android.com/topic/architecture/recommendations (дата звернення 29.12.2023).
14) Detekt source code // Github, Detekt. URL: https://github.com/detekt/detekt (дата звернення 29.12.2023).
15) Publish your app // Android Developers. URL: https://developer.android.com/studio/publish (дата звернення 29.12.2023).

ДОДАТОК А

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024 р.

Мобільний застосунок для контролю особистого часу

Технічне завдання

КПІ.ІП-1324.045490.02.91

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Віталій НЕЩЕРЕТ

Київ – 2024

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ КОНТРОЛЮ ОСОБИСТОГО ЧАСУ.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення мобільного застосунку для контролю особистого часу AnyTime, котре використовується для створення задач, відстежування прогресу та призначена для людей, які хочуть покращити своє вміння управляти часом та концентруватися на справах.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Мобільний застосунок для контролю особистого часу був створений з метою підвищення ефективності користувача, оптимізації власного часу та автоматизації управління особистим розкладом, що є завданням курсової роботи.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для відстежування задач, їхньої пріоритетності та планування витрат часу.

Метою розробки є забезпечення користувачам зручного та ефективного інструменту для управління особистим часом, сприяючи підвищенню продуктивності та досягненню особистих цілей.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

1. Відображення фрагмента “Login” (Рисунок 4.1);
   1.1. Забезпечення зміни режиму між Sign In та Sign Up після натискання на текст “Create new account” та “Login to existing account” відповідно;
   1.2. Відображення повідомлення про помилку в разі неуспішної спроби входу;
   1.3. Перехід до фрагмента “TaskList” після успішного входу в акаунт, створення нового акаунта або авторизації за допомогою Google;

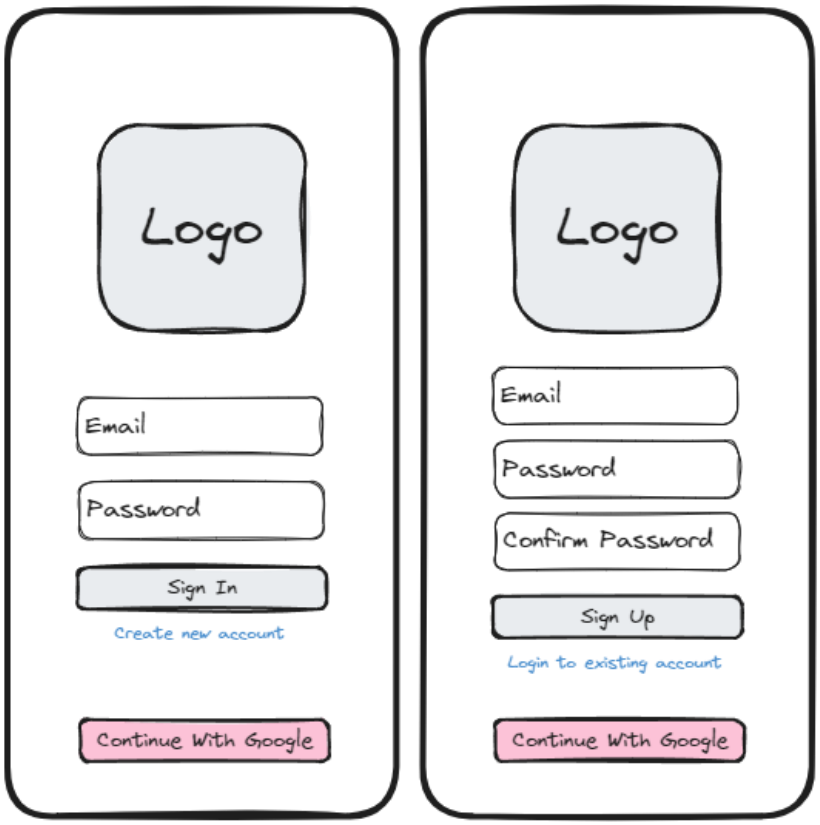

Рисунок 4.1 - Прототипи фрагмента “Login”

2. Відображення фрагмента “TaskList” (Рисунок 4.2);

2.1. Можливість здійснити навігацію до фрагментів “Focus”, “Calendar” та “Profile” за допомогою нижнього меню;

2.2. Реагування інтерфейсу на зміну користувачем фільтрів для Priority та Category;

2.3. Навігація до фрагмента “EditTask” після натискання на кнопку “+”;

2.4. Відкриття фрагмента “TaskInfo” та передача у нього відповідних даних при натисканні на елемент списку;

2.5. Після натискання на чекбокс елемента списку, позначити його виконаним та перемістити вниз;

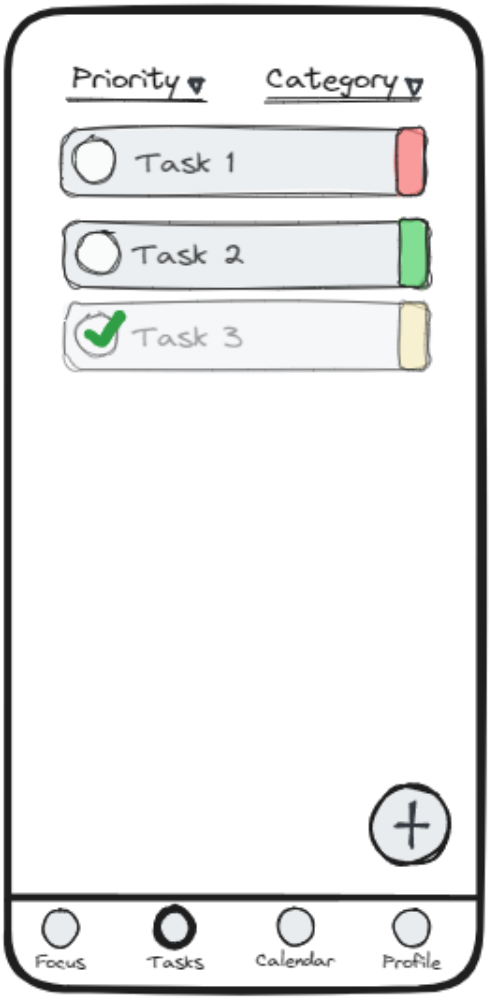

Рисунок 4.2 - Прототип фрагмента “TaskList”

3. Відображення фрагмента “Calendar” (Рисунок 4.3);

3.1. Можливість здійснити навігацію до фрагментів “Focus”, “TaskList” та “Profile” за допомогою нижнього меню;

3.2. Відображення активних завдань залежно від обраної дати;

3.3. Відкриття фрагмента “TaskInfo” та передача у нього відповідних даних при натисканні на елемент списку;

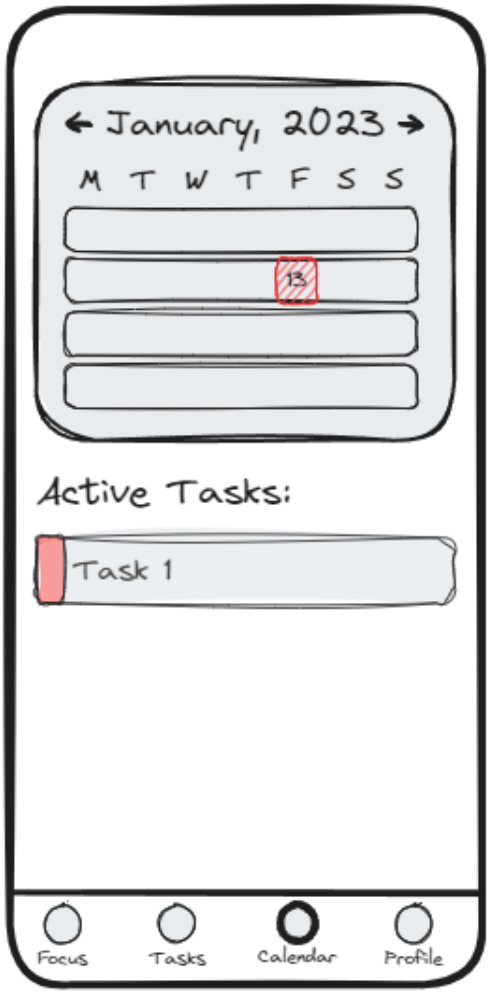

Рисунок 4.3 - Прототип фрагмента "Calendar"

4. Відображення фрагмента “Profile” (Рисунок 4.4);

4.1. Можливість здійснити навігацію до фрагментів “Focus”, “TaskList” та “Calendar” за допомогою нижнього меню;

4.2. Дозволити користувачу відредагувати фото профілю та ім’я після натискання на текст “Edit”;

4.3. Відобразити фрагмент “Categories” після натискання на кнопку “Manage categories”;

4.4. Відображення діалогового вікна про підтвердження виходу (Рисунок 4.5) після натискання на кнопку “Log Out” та навігація до фрагмента “Login” в разі підтвердження користувачем виходу з акаунта;

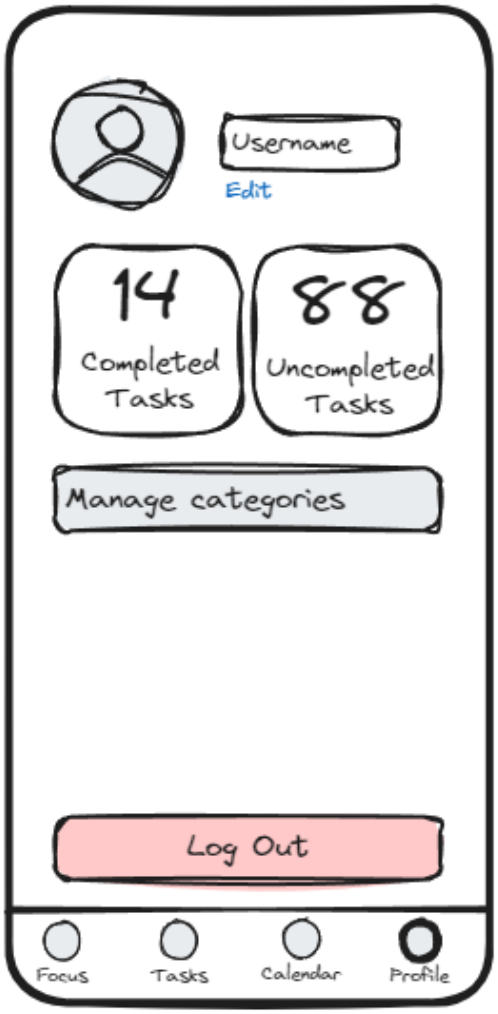

Рисунок 4.4 - Прототип фрагмента "Profile"

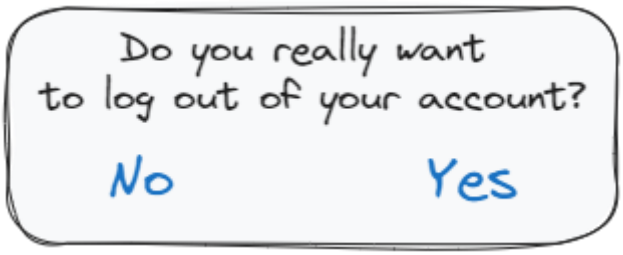

Рисунок 4.5 - Прототип діалогового вікна про підтвердження виходу

5. Відображення фрагмента “Focus” (Рисунок 4.6);
   5.1. Можливість здійснити навігацію до фрагментів “TaskList”, “Calendar” та “Profile” за допомогою нижнього меню;
   5.2. Зміна текстових міток залежно від стану таймера (зупинений, помодоро, коротка пауза, довга пауза);
   5.3. Після натискання на кнопку “Play/Pause” почати або зупинити зворотний відлік;
   5.4. Після натискання на кнопку “Next” відобразити наступний стан таймера;

5.5. Почати зворотний відлік спочатку після активування клавіші “Restart”;

Рисунок 4.6 - Прототип фрагмента “Focus”

6. Відображення фрагмента “TaskInfo” (Рисунок 4.7);

6.1. Повернутися до попереднього фрагмента в результаті кліка по стрілці;

6.2. Після натискання на чекбокс, змінити стан завдання та відобразити зміну;

6.3. Здійснити навігацію до фрагмента “EditTask” після натиснення на кнопку “Edit”;

6.4. Натиснувши на “Delete”, видалити задачу та повернутись до попереднього фрагмента;

Рисунок 4.7 - Прототип фрагмента “TaskInfo”

7. Відображення фрагмента “EditTask” (Рисунок 4.8);
   7.1. Повернутися до попереднього фрагмента в результаті кліка по стрілці;
   7.2. Зміна значень в текстових полях залежно від переданих даних;
   7.3. Збереження/створення завдання та повернення до попереднього фрагмента після натискання на відповідну кнопку;

Рисунок 4.8 - Прототипи фрагмента “EditTask”

8. Відображення фрагмента “Categories” (Рисунок 4.9);
   - 8.1. Повернутися до попереднього фрагмента в результаті кліка по стрілці;
   - 8.2. Після натискання на текст “Add category” відобразити діалогове вікно із запитом імені категорії (Рисунок 4.10) та створити категорію, якщо дія не буде скасована;
   - 8.3. Видалити категорію після натиснення на кнопку “Delete” елемента списку;

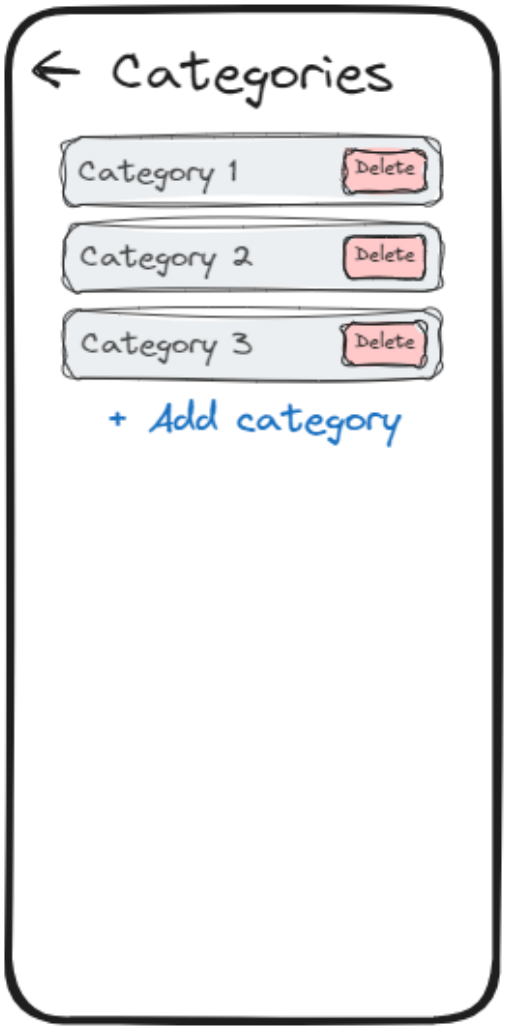

Рисунок 4.9 – Прототип фрагмента “Categories”

Рисунок 4.10 – Прототип діалогового вікна із запитом імені категорії

4.1.2 Для користувача:

1. Можливість авторизуватися та керувати акаунтом в додатку;
   1.1. Можливість створити акаунт за допомогою електронної пошти;
   1.2. Можливість створити акаунт за допомогою акаунта Google;

1.3. Можливість увійти в наявний акаунт за електронною поштою та паролем;

1.4. Можливість увійти в наявний акаунт, що прив’язаний до Google;

1.5. Можливість змінити аватар та нікнейм акаунта;

1.6. Можливість вийти з акаунту;

2. Можливість переглядати та керувати завданнями;

2.1. Можливість переглядати список завдань;

2.2. Можливість переглядати детальну інформацію про завдання;

2.3. Можливість редагувати завдання;

2.3.1. Можливість змінити статус (активне/виконано);

2.3.2. Можливість змінити назву;

2.3.3. Можливість змінити категорію;

2.3.4. Можливість змінити пріоритет;

2.3.5. Можливість змінити дату;

2.3.6. Можливість змінити опис;

2.4. Можливість створити нове завдання;

3. Можливість переглядати та керувати категоріями;

3.1. Можливість переглянути список категорій;

3.2. Можливість додати категорію;

3.3. Можливість видалити категорію;

4. Можливість фільтрувати завдання та переглядати статистику;

4.1. Можливість переглянути завдання відфільтровані за пріоритетом;

4.2. Можливість переглянути завдання відфільтровані за категорією;

4.3. Можливість переглянути активні завдання для обраної дати;

4.4. Можливість побачити статистику за кількістю активних та виконаних завдань;

5. Можливість скористатися режимом фокусування;

5.1. Можливість почати відлік таймера та слідкувати за часом сеансу;

5.2. Можливість поставити таймер на паузу;

5.3. Можливість почати відлік таймера спочатку;

5.4. Можливість змінити стан таймера.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. Відстежувати аварійні завершення роботи додатка на пристроях користувачів.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на мобільних пристроях з операційною системою Android.

Мінімальна конфігурація технічних засобів:

- Android 8;
- об‘єм ОЗП: 1 Гб;
- процесор з тактовою частотою 1.5 ГГц;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

- Android 13;
- об‘єм ОЗП: 4 Гб;
- процесор з тактовою частотою 2.8 ГГц;
- підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати з операційними системами сімейства Android.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати мовою програмування Kotlin.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Android Studio.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді Android application та .apk файлу, завантажених до онлайн репозиторію.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- керівництво користувача;
- програма та методика тестування;
- текст програми.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема структурна компонент;
- схема структурна класів програмного забезпечення;
- креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою роботи	21.10	
2.	Розробка технічного завдання	13.11	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	14.11	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	17.11	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	21.11	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	27.11	Тести, результати тестування
7.	Розробка матеріалів текстової частини роботи	30.11	Пояснювальна записка
8.	Розробка матеріалів графічної частини роботи	28.12	Графічний матеріал проекту
9.	Оформлення технічної документації роботи	30.12	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

ДОДАТОК Б

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024 р.

Мобільний застосунок для контролю особистого часу

Текст програми

КПІ.ІП-1324.045490.05.13

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Віталій НЕЩЕРЕТ

Київ – 2024

Файл LoginFragment.kt

```
package com.makelick.anytime.view.login

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.common.api.ApiException
import
com.google.firebase.auth.FirebaseAuthInvalidCredentialsException
import com.google.firebase.auth.FirebaseAuthInvalidUserException
import com.google.firebase.auth.FirebaseAuthUserCollisionException
import com.google.firebase.auth.FirebaseAuthWeakPasswordException
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentLoginBinding
import com.makelick.anytime.view.BaseFragment
import com.makelick.anytime.view.MainActivity
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class LoginFragment :

BaseFragment<FragmentLoginBinding>(FragmentLoginBinding::inflate)
{

    private val viewModel: LoginViewModel by viewModels()

    private val googleSignInLauncher:
ActivityResultLauncher<Intent> =

registerForActivityResult(ActivityResultContracts.StartActivityFor
Result()) { result ->
            try {
                val task =
GoogleSignIn.getSignedInAccountFromIntent(result.data)
                val account =
task.getResult(ApiException::class.java)
                viewModel.signInWithGoogle(account)
            } catch (e: ApiException) {
                Toast.makeText(
                    requireContext(),
                    getString(R.string.google_sign_in_failed),
                    Toast.LENGTH_SHORT
                ).show()
            }
```

```
        }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        observeViewModel()
        setupUI()
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            viewModel.isLoginMode.collect { changeMode(it) }
        }

        lifecycleScope.launch {
            viewModel.result.collect { handleResult(it) }
        }

        lifecycleScope.launch {
            viewModel.isLoading.collect {
                with(binding) {
                    button.isEnabled = !it
                    googleSignInButton.isEnabled = !it
                    changeMode.isEnabled = !it
                    binding.loadingBar.visibility = if (it)
View.VISIBLE else View.GONE
                }
            }
        }
    }

    private fun changeMode(isSignIn: Boolean) {
        with(binding) {
            if (isSignIn) {
                passwordConfirmationLayout.visibility = View.GONE
                button.text = getString(R.string.sign_in)
                changeMode.text =
getString(R.string.create_new_account)
            } else {
                passwordConfirmationLayout.visibility =
View.VISIBLE
                button.text = getString(R.string.sign_up)
                changeMode.text =
getString(R.string.sign_in_to_existing_account)
            }
        }
    }

    private fun handleResult(result: Result<Unit>) {
        if (result.isFailure) {
            with(binding) {
                when (result.exceptionOrNull()) {
                    is FirebaseAuthInvalidUserException -> {
```

```
                        emailLayout.error =
getString(R.string.invalid_email)
                    }

                    is FirebaseAuthUserCollisionException -> {
                        emailLayout.error =
getString(R.string.email_already_in_use)
                    }

                    is FirebaseAuthWeakPasswordException -> {
                        passwordLayout.error =
getString(R.string.weak_password)
                    }

                    is FirebaseAuthInvalidCredentialsException ->
{
                        emailLayout.error =
getString(R.string.invalid_email)
                    }

                    else -> {
                        passwordLayout.error =
getString(R.string.invalid_email_or_password)
                        emailLayout.error =
getString(R.string.invalid_email_or_password)
                    }
                }
                root.clearFocus()
            }
        } else {
            navigateToTasksFragment()
        }
    }

    private fun navigateToTasksFragment() {

findNavController().navigate(LoginFragmentDirections.actionLoginFr
agmentToTasksFragment())
        (activity as
MainActivity).changeBottomNavSelectedId(R.id.tasks)
    }

    private fun setupUI() {
        binding.apply {

            email.setOnFocusChangeListener { _, hasFocus ->
                if (hasFocus) emailLayout.error = null
            }

            password.setOnFocusChangeListener { _, hasFocus ->
                if (hasFocus) passwordLayout.error = null
            }
```

```
            passwordConfirmation.setOnFocusChangeListener { _,
hasFocus ->
                if (hasFocus) passwordConfirmationLayout.error =
null
            }

            changeMode.setOnClickListener {
                viewModel.changeMode()

                clearErrors()
                clearInputs()
                root.clearFocus()
            }

            button.setOnClickListener {
                clearErrors()
                root.clearFocus()
                attemptLogin()
            }

            googleSignInButton.setOnClickListener {

googleSignInLauncher.launch(viewModel.googleSignInIntent)
                root.clearFocus()
            }
        }
    }

    private fun clearErrors() {
        with(binding) {
            emailLayout.error = null
            passwordLayout.error = null
            passwordConfirmationLayout.error = null
        }
    }

    private fun clearInputs() {
        with(binding) {
            email.setText("")
            password.setText("")
            passwordConfirmation.setText("")
        }
    }

    private fun attemptLogin() {
        val emailStr = binding.email.text.toString()
        val passwordStr = binding.password.text.toString()

        if (!viewModel.isValidEmail(emailStr)) {
            binding.emailLayout.error =
getString(R.string.invalid_email_format)
            return
        }
```

```
        if (viewModel.isLoginMode.value) {
            viewModel.login(emailStr, passwordStr)
            return
        }

        if (passwordStr !=
binding.passwordConfirmation.text.toString()) {
            binding.passwordConfirmationLayout.error =
getString(R.string.passwords_do_not_match)
            return
        }

        viewModel.signUp(emailStr, passwordStr)
    }
}
```

Файл LoginViewModel.kt

```
package com.makelick.anytime.view.login

import android.content.Intent
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.makelick.anytime.model.AccountRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class LoginViewModel @Inject constructor(
    private val accountRepository: AccountRepository
) : ViewModel() {

    val isLoading = MutableStateFlow(false)
    val isLoginMode = MutableStateFlow(true)
    val result = MutableSharedFlow<Result<Unit>>()

    val googleSignInIntent: Intent
        get() = accountRepository.getGoogleSignInIntent()

    fun changeMode() {
        isLoginMode.value = !isLoginMode.value
    }

    fun login(email: String, password: String) {
        viewModelScope.launch {
            isLoading.value = true
            result.emit(accountRepository.signIn(email, password))
            isLoading.value = false
        }
    }
```

```
    fun signUp(email: String, password: String) {
        viewModelScope.launch {
            isLoading.value = true
            result.emit(accountRepository.signUp(email, password))
            isLoading.value = false
        }
    }

    fun signInWithGoogle(account: GoogleSignInAccount) {
        viewModelScope.launch {
            isLoading.value = true

result.emit(accountRepository.signInWithGoogle(account))
            isLoading.value = false

        }
    }

    fun isValidEmail(email: String): Boolean {
        val emailRegex = "^[A-Za-z](.*)(@)(.+)(\\.)(.+)$"
        return email.matches(emailRegex.toRegex())
    }
}
```

Файл AccountRepository.kt

```
package com.makelick.anytime.model

import android.content.Context
import android.content.Intent
import android.net.Uri
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.auth.api.signin.GoogleSignInClient
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.firebase.auth.GoogleAuthProvider
import com.google.firebase.auth.ktx.auth
import com.google.firebase.auth.ktx.userProfileChangeRequest
import com.google.firebase.ktx.Firebase
import com.makelick.anytime.R
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class AccountRepository @Inject constructor(
    @ApplicationContext private val context: Context
) {
    private val auth = Firebase.auth

    private var googleSignInClient: GoogleSignInClient =
        GoogleSignIn.getClient(
            context,
```

```
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)

.requestIdToken(context.getString(R.string.google_web_client_id))
                .requestEmail()
                .build()
        )

    fun getUser() = auth.currentUser

    fun updateProfile(username: String, photoUrl: Uri?) {
        auth.currentUser?.updateProfile(userProfileChangeRequest {
            displayName = username
            photoUri = photoUrl
        })
    }

    suspend fun signIn(email: String, password: String):
Result<Unit> {
        return try {
            auth.signInWithEmailAndPassword(email,
password).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun signUp(email: String, password: String):
Result<Unit> {
        return try {
            auth.createUserWithEmailAndPassword(email,
password).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    fun getGoogleSignInIntent(): Intent {
        return googleSignInClient.signInIntent
    }

    suspend fun signInWithGoogle(account: GoogleSignInAccount):
Result<Unit> {
        return try {
            val credential =
GoogleAuthProvider.getCredential(account.idToken, null)
            auth.signInWithCredential(credential).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
```

```
    }

    fun signOut() {
        auth.signOut()
        googleSignInClient.signOut()
    }
}
```

Файл ProfileFragment.kt

```
package com.makelick.anytime.view.profile

import android.net.Uri
import android.os.Bundle
import android.view.View
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import coil.load
import coil.transform.CircleCropTransformation
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentProfileBinding
import com.makelick.anytime.view.BaseFragment
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class ProfileFragment :
BaseFragment<FragmentProfileBinding>(FragmentProfileBinding::infla
te) {

    private val viewModel: ProfileViewModel by viewModels()

    private val pickImageLauncher:
ActivityResultLauncher<PickVisualMediaRequest> =

registerForActivityResult(ActivityResultContracts.PickVisualMedia(
)) {
            it?.let { uploadImage(it) }
        }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
    }

    private fun setupUI() {
```

```
        with(binding) {

            viewModel.user?.let { user ->
                profileImage.load(user.photoUrl) {
                    transformations(CircleCropTransformation())
                    fallback(R.drawable.ic_profile)
                    error(R.drawable.ic_profile)
                }
                username.setText(viewModel.user?.displayName)
            }

            edit.setOnClickListener {
                root.clearFocus()
                changeMode()
            }

            username.onFocusChangeListener =
View.OnFocusChangeListener { _, _ ->
                usernameLayout.error = null
            }

            profileImage.setOnClickListener {
                if (viewModel.isEditMode.value) {
                    pickImageLauncher.launch(

PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.Ima
geOnly)
                    )
                }
            }

            buttonManageCategories.setOnClickListener {
                navigateToCategories()
            }

            buttonLogOut.setOnClickListener {
                MaterialAlertDialogBuilder(requireContext()).apply
{
                    setTitle(getString(R.string.sign_out))

setMessage(getString(R.string.sign_out_message))
                    setPositiveButton(getString(R.string.yes)) {
dialog, _ ->
                        viewModel.signOut()
                        navigateToLogin()
                        dialog.dismiss()
                    }
                    setNegativeButton(getString(R.string.no)) {
dialog, _ -> dialog.dismiss() }
                    show()
                }
            }
        }
```

```
    }

    private fun navigateToCategories() {

findNavController().navigate(ProfileFragmentDirections.actionProfi
leFragmentToCategoriesFragment())
    }

    private fun navigateToLogin() {

findNavController().navigate(ProfileFragmentDirections.actionProfi
leFragmentToLoginFragment())
    }

    private fun changeMode() {
        if (viewModel.isEditMode.value) {
            if (binding.username.text.toString().isBlank()) {
                binding.usernameLayout.error =
getString(R.string.error_empty)
            } else {

viewModel.applyProfileChanges(binding.username.text.toString())
            }
        } else {
            viewModel.isEditMode.value = true
        }
    }

    private fun uploadImage(uri: Uri?) {
        lifecycleScope.launch {
            binding.imageLoadingBar.visibility = View.VISIBLE
            viewModel.loadNewImage(uri)
            binding.profileImage.load(viewModel.loadedImageUri) {
                transformations(CircleCropTransformation())
                fallback(R.drawable.ic_profile)
                error(R.drawable.ic_profile)
            }
            binding.imageLoadingBar.visibility = View.GONE
        }
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            viewModel.isEditMode.collect { editMode ->
                if (editMode) {
                    binding.edit.text = getString(R.string.save)
                    binding.usernameLayout.isEnabled = true
                    binding.imageText.visibility = View.VISIBLE
                } else {
                    binding.edit.text = getString(R.string.edit)
                    binding.usernameLayout.isEnabled = false
                    binding.imageText.visibility = View.GONE
                }
```

```
            }
        }

        lifecycleScope.launch {
            viewModel.completedTasksCount.collect {
                binding.completedTasks.text = it.toString()
            }
        }

        lifecycleScope.launch {
            viewModel.uncompletedTasksCount.collect {
                binding.uncompletedTasks.text = it.toString()
            }
        }

    }
}
```

Файл ProfileViewModel.kt

```
package com.makelick.anytime.view.profile

import android.net.Uri
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.makelick.anytime.model.AccountRepository
import com.makelick.anytime.model.FirestoreRepository
import com.makelick.anytime.model.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class ProfileViewModel @Inject constructor(
    private val accountRepository: AccountRepository,
    private val storageRepository: StorageRepository,
    private val firestoreRepository: FirestoreRepository
) : ViewModel() {

    val user = accountRepository.getUser()
    val isEditMode = MutableStateFlow(false)
    val completedTasksCount = MutableStateFlow(0)
    val uncompletedTasksCount = MutableStateFlow(0)

    var loadedImageUri: Uri? = null

    init {
        viewModelScope.launch {
            firestoreRepository.allTasks.collect {
                completedTasksCount.emit(
                    firestoreRepository.allTasks.value.count {
it.isCompleted }
                )
```

```
                uncompletedTasksCount.emit(
                    firestoreRepository.allTasks.value.count {
!it.isCompleted }
                )
            }
        }
    }

    fun signOut() {
        accountRepository.signOut()
    }

    fun applyProfileChanges(username: String) {
        accountRepository.updateProfile(username, loadedImageUri
?: user?.photoUrl)
        isEditMode.value = false
    }

    suspend fun loadNewImage(file: Uri?) {
        if (user != null && file != null) {
            val result = storageRepository.uploadImage(user.uid,
file)
            loadedImageUri = result.getOrNull().takeIf {
result.isSuccess }
        }
    }
}
```

Файл TasksFragment.kt

```
package com.makelick.anytime.view.tasks

import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentTasksBinding
import com.makelick.anytime.model.entity.Task
import com.makelick.anytime.view.BaseFragment
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class TasksFragment :
BaseFragment<FragmentTasksBinding>(FragmentTasksBinding::inflate)
{

    private val viewModel: TasksViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState:
```

```
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
    }

    private fun setupUI() {
        with(binding) {

            spinnerPriority.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(parent:
AdapterView<*>?, view: View?, position: Int, id: Long) {
                    viewModel.selectedPriority.value =
spinnerPriority.selectedItem.toString().convertPriorityToInt()
                    viewModel.loadTasks()
                }

                override fun onNothingSelected(p0:
AdapterView<*>?) {
                    viewModel.selectedPriority.value = -1
                    viewModel.loadTasks()
                }
            }

            spinnerCategory.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(parent:
AdapterView<*>?, view: View?, position: Int, id: Long) {
                    viewModel.selectedCategory.value =
spinnerCategory.selectedItem.toString()
                    viewModel.loadTasks()
                }

                override fun onNothingSelected(p0:
AdapterView<*>?) {
                    viewModel.selectedCategory.value = "All
categories"
                    viewModel.loadTasks()
                }
            }

            tasksRecyclerView.apply {
                adapter =
TasksAdapter(viewModel::changeTaskStatus, ::navigateToTaskInfo)
                layoutManager =
LinearLayoutManager(requireContext())
            }

            addTaskButton.setOnClickListener {
                navigateToCreateTask()
            }
```

```
        }
    }

    private fun String.convertPriorityToInt(): Int {
        return when (this) {
            getString(R.string.high_priority) -> 3
            getString(R.string.medium_priority) -> 2
            getString(R.string.low_priority) -> 1
            getString(R.string.no_priority) -> 0
            else -> -1
        }
    }

    private fun navigateToTaskInfo(task: Task) {
        val action =
TasksFragmentDirections.actionTasksFragmentToTaskInfoFragment(task
)
        findNavController().navigate(action)
    }

    private fun navigateToCreateTask() {
        val action =
TasksFragmentDirections.actionTasksFragmentToEditTaskFragment(true
, null)
        findNavController().navigate(action)
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            viewModel.isLoading.collect {
                binding.tasksLoadingBar.visibility = if (it)
View.VISIBLE else View.GONE
            }
        }

        lifecycleScope.launch {
            viewModel.tasks.collect {
                (binding.tasksRecyclerView.adapter as
TasksAdapter).submitList(it)
                binding.emptyTasksText.visibility = if
(it.isEmpty()) View.VISIBLE else View.GONE
            }
        }

        lifecycleScope.launch {
            viewModel.categories.collect {
                binding.spinnerCategory.adapter = ArrayAdapter(
                    requireContext(),
                    android.R.layout.simple_spinner_dropdown_item,
                    it
                )
            }
        }
```

```
    }
}
```

Файл TasksAdapter.kt

```
package com.makelick.anytime.view.tasks

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.core.content.ContextCompat.getColor
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView.ViewHolder
import com.makelick.anytime.R
import com.makelick.anytime.databinding.ItemTasklistBinding
import com.makelick.anytime.model.entity.Task

class TasksAdapter(
    private val onCheckboxClick: (Task) -> Unit,
    private val onTaskClick: (Task) -> Unit
) : ListAdapter<Task,
TasksAdapter.TasksViewHolder>(TaskDiffCallback()) {
    class TaskDiffCallback : DiffUtil.ItemCallback<Task>() {
        override fun areItemsTheSame(oldItem: Task, newItem: Task)
=
            oldItem.id == newItem.id

        override fun areContentsTheSame(oldItem: Task, newItem:
Task) =
            oldItem == newItem
    }

    inner class TasksViewHolder(private val binding:
ItemTasklistBinding) :
        ViewHolder(binding.root) {
        fun bind(task: Task) {
            with(binding) {

                taskTitle.text = task.title


taskPriority.setBackgroundColor(getPriorityColor(task.priority))

                taskCheckBox.isChecked = task.isCompleted == true
                taskCheckBox.setOnClickListener {
                    onCheckboxClick(task)
                }
                root.setOnClickListener {
                    onTaskClick(task)
                }
            }
        }

        private fun getPriorityColor(priority: Int?) = when
(priority) {
```

```
            1 -> getColor(binding.root.context,
R.color.low_priority)
            2 -> getColor(binding.root.context,
R.color.medium_priority)
            3 -> getColor(binding.root.context,
R.color.high_priority)
            else -> getColor(binding.root.context,
R.color.no_priority)
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int) =
        TasksViewHolder(
            ItemTasklistBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )

    override fun onBindViewHolder(holder: TasksViewHolder,
position: Int) {
        holder.bind(getItem(position))
    }
}
```

Файл TasksViewModel.kt

```
package com.makelick.anytime.view.tasks

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.makelick.anytime.model.FirestoreRepository
import com.makelick.anytime.model.entity.Task
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class TasksViewModel @Inject constructor(
    private val firestoreRepository: FirestoreRepository
) : ViewModel() {

    val isLoading = MutableStateFlow(true)
    val selectedPriority = MutableStateFlow(-1)
    val selectedCategory = MutableStateFlow("All categories")

    val tasks = MutableStateFlow<List<Task>>(emptyList())
    val categories = MutableStateFlow<List<String>>(emptyList())

    init {
        viewModelScope.launch {
            firestoreRepository.allTasks.collect {
```

```
                loadTasks()
            }
        }

        viewModelScope.launch {
            firestoreRepository.categories.collect {
                categories.value = loadCategories()
            }
        }
    }

    fun loadTasks() {
        viewModelScope.launch {

tasks.emit(filterTasks(firestoreRepository.allTasks.value))
            isLoading.value = false
        }
    }

    private fun filterTasks(tasks: List<Task>): List<Task> {
        return tasks.filter { task ->
            (selectedPriority.value == -1 || task.priority ==
selectedPriority.value) &&
                    (selectedCategory.value == "All categories" ||
task.category == selectedCategory.value)

        }.sortedBy { it.isCompleted }
    }

    private fun loadCategories(): List<String> {
        val result = mutableListOf("All categories")
        result.addAll(firestoreRepository.categories.value)
        return result
    }

    fun changeTaskStatus(task: Task) {
        viewModelScope.launch {
            task.isCompleted = !task.isCompleted
            firestoreRepository.updateTask(task)
            tasks.emit(tasks.value.sortedBy { it.isCompleted })
        }
    }
}
```

Файл FirestoreRepository.kt

```
package com.makelick.anytime.model

import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import com.makelick.anytime.model.entity.Task
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
import javax.inject.Singleton
```

```
@Singleton
class FirestoreRepository @Inject constructor(
    userId: String
) {

    private val userDocRef =
Firebase.firestore.document("users/$userId")
    private val tasksCollectionRef =
userDocRef.collection("tasks")

    val allTasks = MutableStateFlow<List<Task>>(emptyList())
    val categories = MutableStateFlow<List<String>>(emptyList())

    init {
        tasksCollectionRef.addSnapshotListener { value, error ->
            if (error != null) return@addSnapshotListener
            allTasks.value = value?.toObjects(Task::class.java) ?:
emptyList()
        }

        userDocRef.addSnapshotListener { value, error ->
            if (error != null) return@addSnapshotListener

            if (value?.exists() == true) {
                categories.value =
                    (value.get("categories") as? List<*>)?.map {
it.toString() } ?: emptyList()
            } else {
                userDocRef.set(mapOf("categories" to
listOf("Personal", "Work")))
            }
        }
    }

    suspend fun addTask(task: Task): Result<Unit> {
        return try {
            task.id = tasksCollectionRef.document().id
            updateTask(task)
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun updateTask(task: Task): Result<Unit> {
        return try {

tasksCollectionRef.document(task.id.toString()).set(task).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
```

```
    }

    suspend fun updateCategories(newCategories: List<String>):
Result<Unit> {
        return try {
            userDocRef.update("categories", newCategories).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun deleteTask(taskId: String) {
        tasksCollectionRef.document(taskId).delete().await()
    }
}
```

Файл FocusFragment.kt

```
package com.makelick.anytime.view.focus

import android.Manifest
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.view.View
import androidx.core.content.PermissionChecker
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentFocusBinding
import com.makelick.anytime.model.TimerRepository.Companion.SECOND
import com.makelick.anytime.model.entity.PomodoroMode
import com.makelick.anytime.view.BaseFragment
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import java.util.Locale

@AndroidEntryPoint
class FocusFragment :
BaseFragment<FragmentFocusBinding>(FragmentFocusBinding::inflate)
{

    private val viewModel: FocusViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
        askNotificationPermission()
    }

    private fun askNotificationPermission() {
```

```
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU)
{
            if (PermissionChecker.checkSelfPermission(
                    requireContext(),
                    Manifest.permission.POST_NOTIFICATIONS
                ) != PermissionChecker.PERMISSION_GRANTED
            ) {
                @Suppress("DEPRECATION")

requestPermissions(arrayOf(Manifest.permission.POST_NOTIFICATIONS)
, 1)
            }
        }
    }

    private fun setupUI() {

        binding.playButton.setOnClickListener {
            if (viewModel.isTimerRunning.value) {
                context?.stopService(Intent(context,
TimerService::class.java))
                viewModel.pauseTimer()

binding.iconPlay.setImageResource(R.drawable.ic_play)
            } else {
                context?.startService(Intent(context,
TimerService::class.java))

binding.iconPlay.setImageResource(R.drawable.ic_pause)
            }
        }

        binding.restartButton.setOnClickListener {
            stopTimer()
            binding.iconPlay.setImageResource(R.drawable.ic_play)
        }

        binding.nextButton.setOnClickListener {
            viewModel.nextMode()
            stopTimer()
            context?.startService(Intent(context,
TimerService::class.java))
            binding.iconPlay.setImageResource(R.drawable.ic_pause)
        }
    }

    private fun stopTimer() {
        context?.stopService(Intent(context,
TimerService::class.java))
        viewModel.stopTimer()
    }

    private fun observeViewModel() {
```

```
        lifecycleScope.launch {
            viewModel.isTimerRunning.collect {
                if (it) {

binding.iconPlay.setImageResource(R.drawable.ic_pause)
                } else {

binding.iconPlay.setImageResource(R.drawable.ic_play)
                }
            }
        }

        lifecycleScope.launch {
            viewModel.currentTime.collect {
                binding.time.text = getStringTime(it)
            }
        }

        lifecycleScope.launch {
            viewModel.timerMode.collect {
                with(binding) {
                    title.text = it.title
                    hint.text = getHint(it)


timeCard.setCardBackgroundColor(getTimerColor(it))

playButton.setCardBackgroundColor(getTimerColor(it))

                    if (it == PomodoroMode.POMODORO) {
                        countOfBreaks.text = getString(
                            R.string.focus_count_of_breaks,
                            viewModel.timerBreaksCount.value
                        )
                        countOfBreaks.visibility = View.VISIBLE
                    } else {
                        countOfBreaks.visibility = View.GONE
                    }
                }
            }
        }
    }

    private fun getTimerColor(mode: PomodoroMode): Int {
        return when (mode) {
            PomodoroMode.POMODORO ->
resources.getColor(R.color.primary, null)
            PomodoroMode.SHORT_BREAK ->
resources.getColor(R.color.secondary, null)
            PomodoroMode.LONG_BREAK ->
resources.getColor(R.color.accent, null)
        }
```

```
    }

    private fun getHint(mode: PomodoroMode): String {
        return when (mode) {
            PomodoroMode.POMODORO ->
getString(R.string.focus_hint_pomodoro)
            PomodoroMode.SHORT_BREAK ->
getString(R.string.focus_hint_short_break)
            PomodoroMode.LONG_BREAK ->
getString(R.string.focus_hint_long_break)
        }
    }

    private fun getStringTime(time: Long): String {
        val minutes = (time / SECOND) / 60
        val seconds = (time / SECOND) % 60
        return String.format(Locale.getDefault(), "%02d:%02d",
minutes, seconds)
    }
}
```

Файл FocusViewModel.kt

```
package com.makelick.anytime.view.focus

import androidx.lifecycle.ViewModel
import com.makelick.anytime.model.TimerRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

@HiltViewModel
class FocusViewModel @Inject constructor(
    private val timerRepository: TimerRepository
) : ViewModel() {

    val timerMode = timerRepository.timerMode
    val timerBreaksCount = timerRepository.timerBreaksCount
    val isTimerRunning  = timerRepository.isTimerRunning
    val currentTime = timerRepository.currentTime

    fun pauseTimer() {
        timerRepository.pauseTimer()
    }

    fun stopTimer() {
        timerRepository.stopTimer()
    }

    fun nextMode() {
        timerRepository.nextMode()
    }

}
```

Файл TimerService.kt

```
package com.makelick.anytime.view.focus

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.Service
import android.content.Intent
import androidx.core.app.NotificationCompat
import androidx.navigation.NavDeepLinkBuilder
import com.makelick.anytime.R
import com.makelick.anytime.model.TimerRepository
import com.makelick.anytime.model.TimerRepository.Companion.SECOND
import com.makelick.anytime.model.entity.PomodoroMode
import com.makelick.anytime.view.MainActivity
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import java.util.Locale
import javax.inject.Inject

@AndroidEntryPoint
class TimerService : Service() {

    @Inject
    lateinit var timerRepository: TimerRepository

    override fun onStartCommand(intent: Intent?, flags: Int,
startId: Int): Int {


timerRepository.startTimer(timerRepository.timerMode.value.timeInM
illis)
        createNotificationChannel()
        CoroutineScope(Dispatchers.Main).launch {
            timerRepository.currentTime.collect {
                if (it > 0) {
                    val notification = createNotification(

getString(R.string.timer_notification_content, getStringTime(it))
                    )
                    startForeground(1, notification)
                } else {
                    val notification = createNotification(

getString(R.string.notification_timer_finished),
                        true
                    )

getSystemService(NotificationManager::class.java).notify(2,
notification)
                    timerRepository.nextMode()
```

```
                    timerRepository.stopTimer()
                    delay(SECOND)

timerRepository.startTimer(timerRepository.timerMode.value.timeInM
illis)
                }
            }
        }

        return super.onStartCommand(intent, flags, startId)
    }

    override fun onBind(p0: Intent?) = null

    private fun getStringTime(time: Long): String {
        val minutes = (time / SECOND) / 60
        val seconds = (time / SECOND) % 60
        return String.format(Locale.getDefault(),"%02d:%02d",
minutes, seconds)
    }

    private fun createNotification(
        content: String,
        isFinal: Boolean = false
    ): Notification {
        val pendingIntent = NavDeepLinkBuilder(this)
            .setComponentName(MainActivity::class.java)
            .setGraph(R.navigation.graph)
            .setDestination(R.id.focusFragment)
            .createPendingIntent()


        return NotificationCompat.Builder(this,
NOTIFICATIONS_CHANNEL_NAME)

.setContentTitle(timerRepository.timerMode.value.title)
            .setContentText(content)
            .setSmallIcon(R.drawable.ic_focus)

.setForegroundServiceBehavior(NotificationCompat.FOREGROUND_SERVIC
E_IMMEDIATE)
            .setContentIntent(pendingIntent)
            .setSilent(!isFinal)
            .setOngoing(!isFinal)
            .build()
    }

    private fun createNotificationChannel() {
        val channel = NotificationChannel(
            NOTIFICATIONS_CHANNEL_NAME,
            PomodoroMode.POMODORO.title,
            NotificationManager.IMPORTANCE_HIGH
        ).apply {
```

```
            description =
getString(R.string.timer_notification_channel_description)
        }


getSystemService(NotificationManager::class.java).createNotificati
onChannel(channel)
    }

    companion object {
        const val NOTIFICATIONS_CHANNEL_NAME = "TIMER_CHANNEL"
    }
}
```

Файл TimerRepository.kt

```
package com.makelick.anytime.model

import android.os.CountDownTimer
import com.makelick.anytime.model.entity.PomodoroMode
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class TimerRepository @Inject constructor(
    private val dataStoreRepository: DataStoreRepository
) {

    private var timer: CountDownTimer? = null

    val timerMode = MutableStateFlow(PomodoroMode.POMODORO)
    val timerBreaksCount = MutableStateFlow(0)
    val isTimerRunning = MutableStateFlow(timer != null)
    val currentTime = MutableStateFlow<Long>(0)


    init {
        CoroutineScope(Dispatchers.IO).launch {
            timerMode.emit(
                getModeByTitle(

dataStoreRepository.getFromDataStore(DataStoreRepository.KEY_TIMER
_MODE)
                    .first() ?: PomodoroMode.POMODORO.title)
            )
            currentTime.value = timerMode.value.timeInMillis
        }
        CoroutineScope(Dispatchers.IO).launch {
            timerBreaksCount.emit(
```

```
dataStoreRepository.getFromDataStore(DataStoreRepository.KEY_TIMER
_BREAKS_COUNT)
                    .first() ?: 0
            )
        }
    }

    fun startTimer(timeInMillis: Long) {
        isTimerRunning.value = true
        timer = object : CountDownTimer(timeInMillis, SECOND) {
            override fun onTick(millisUntilFinished: Long) {
                currentTime.value = millisUntilFinished
            }

            override fun onFinish() {
                currentTime.value = 0
            }
        }.start()
    }

    fun pauseTimer() {
        isTimerRunning.value = false
        timer?.cancel()
        timer = null
    }

    fun stopTimer() {
        isTimerRunning.value = false
        timer?.cancel()
        timer = null
        currentTime.value = timerMode.value.timeInMillis
    }

    fun nextMode() {
        CoroutineScope(Dispatchers.IO).launch {
            if (timerMode.value == PomodoroMode.POMODORO) {
                if (timerBreaksCount.value == COUNT_OF_BREAKS) {
                    dataStoreRepository.saveToDataStore(
                        DataStoreRepository.KEY_TIMER_MODE,
                        PomodoroMode.LONG_BREAK.title
                    )
                    timerMode.value = PomodoroMode.LONG_BREAK

                    dataStoreRepository.saveToDataStore(

DataStoreRepository.KEY_TIMER_BREAKS_COUNT,
                        0
                    )
                    timerBreaksCount.value = 0
                } else {
                    dataStoreRepository.saveToDataStore(
                        DataStoreRepository.KEY_TIMER_MODE,
                        PomodoroMode.SHORT_BREAK.title
```

```
                    )
                    timerMode.value = PomodoroMode.SHORT_BREAK

                    dataStoreRepository.saveToDataStore(

DataStoreRepository.KEY_TIMER_BREAKS_COUNT,
                        timerBreaksCount.value + 1
                    )
                    timerBreaksCount.value =
timerBreaksCount.value + 1
                }
            } else {
                dataStoreRepository.saveToDataStore(
                    DataStoreRepository.KEY_TIMER_MODE,
                    PomodoroMode.POMODORO.title
                )
                timerMode.value = PomodoroMode.POMODORO
            }
            currentTime.value = timerMode.value.timeInMillis
        }
    }

    private fun getModeByTitle(title: String): PomodoroMode {
        return when (title) {
            PomodoroMode.POMODORO.title -> PomodoroMode.POMODORO
            PomodoroMode.SHORT_BREAK.title ->
PomodoroMode.SHORT_BREAK
            PomodoroMode.LONG_BREAK.title ->
PomodoroMode.LONG_BREAK
            else -> PomodoroMode.POMODORO
        }
    }

    companion object {
        private const val COUNT_OF_BREAKS = 4
        const val SECOND = 1_000L
        private const val MINUTE = 60 * SECOND
        const val POMODORO_TIME = 25 * MINUTE
        const val SHORT_BREAK_TIME = 5 * MINUTE
        const val LONG_BREAK_TIME = 15 * MINUTE
    }
}
```

Файл DataStoreRepository.kt

```
package com.makelick.anytime.model

import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.core.intPreferencesKey
import androidx.datastore.preferences.core.stringPreferencesKey
import androidx.datastore.preferences.preferencesDataStore
import dagger.hilt.android.qualifiers.ApplicationContext
```

```
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import javax.inject.Inject
import javax.inject.Singleton


@Singleton
class DataStoreRepository @Inject constructor(@ApplicationContext
private val context: Context) {

    private val Context.dataStore: DataStore<Preferences> by
preferencesDataStore(name = "AnyTimeDataStore")

    suspend fun <T> saveToDataStore(key: Preferences.Key<T>,
value: T) {
        context.dataStore.edit { preferences ->
            preferences[key] = value
        }
    }

    fun <T> getFromDataStore(key: Preferences.Key<T>): Flow<T?> {
        val data = context.dataStore.data.map { preferences ->
            preferences[key]
        }
        return data
    }

    companion object {
        val KEY_TIMER_MODE = stringPreferencesKey("mode")
        val KEY_TIMER_BREAKS_COUNT =
intPreferencesKey("breaks_count")
    }

}
```

ДОДАТОК В

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024 р.

Мобільний застосунок для контролю особистого часу

Програма та методика тестування

КПІ.ІП-1324.045490.04.51

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Віталій НЕЩЕРЕТ

Київ – 2024

ЗМІСТ

5 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є мобільний додаток AnyTime для контролю особистого часу, розроблений під ОС Android.

6 МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог.

7 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовується мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

8 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на мобільних пристроях з різною роздільною здатністю екрану;
- тестування на мобільних пристроях з різною версією операційної системи.

ДОДАТОК Г

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024 р.

Мобільний застосунок для контролю особистого часу

Керівництво користувача

КПІ.ІП-1324.045490.03.34

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Віталій НЕЩЕРЕТ

Київ – 2024

ЗМІСТ

– ПРИЗНАЧЕННЯ ПРОГРАМИ

AnyTime - це мобільний додаток для контролю особистого часу, який допомагає користувачам ефективно планувати та використовувати свій час. Застосунок надає можливість створювати та відстежувати завдання, використовувати Pomodoro таймер для оптимізації робочих інтервалів, а також взаємодіяти з ним на різних пристроях завдяки синхронізації даних через Firebase. AnyTime поєднує базовий функціонал з розширеними можливостями фільтрації, редагування та аналізу продуктивності, роблячи його універсальним інструментом для керування часом у різних сферах життя.

Кінцевий формат збірки застосунку — APK файл на репозиторії.

9 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

9.1 Системні вимоги для коректної роботи

Мінімальна конфігурація технічних засобів:
Android 8;
об‘єм ОЗП: 1 Гб;
процесор з тактовою частотою 1.5 ГГц;
підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:
Android 13;
об‘єм ОЗП: 4 Гб;
процесор з тактовою частотою 2.8 ГГц;
підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

9.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний apk-файл. Для цього спершу необхідно завантажити його на мобільний пристрій, а потім запустити для автоматичного встановлення на пристрій.

9.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

10 ВИКОНАННЯ ПРОГРАМИ

При запуску програмного застосунку користувачу буде відображено поля для вводу пошти та пароля та кнопки для входу в акаунт, створення нового акаунта та входу за допомогою Google (рисунок 3.1).

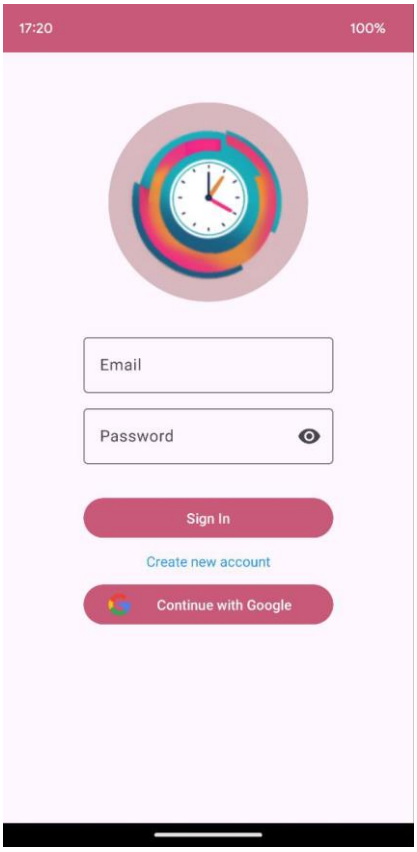

Рисунок 3.1 – Екран авторизації

Користувач має змогу скористатись будь-яким типом авторизації, при введені некоректних даних програма покаже помилку та пояснить які дані введені некоректно. Після авторизації користувач потрапляє на головний екран зі списком завдань (рисунок 3.2).

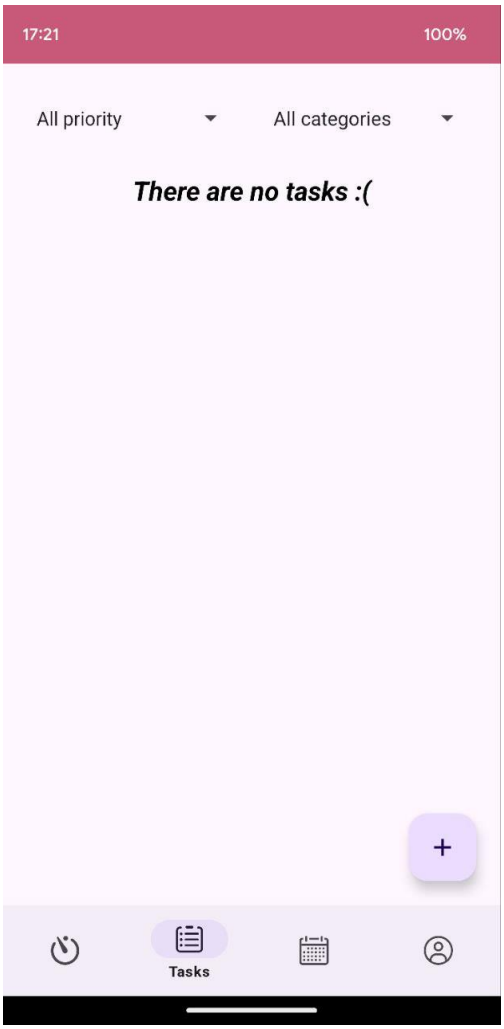

Рисунок 3.2 – Головний екран

Якщо у користувача нема завдань у списку, виведеться відповідне повідомлення. При натисканні на кнопку + користувач перейде до екрану створення завдання (рисунок 3.3).

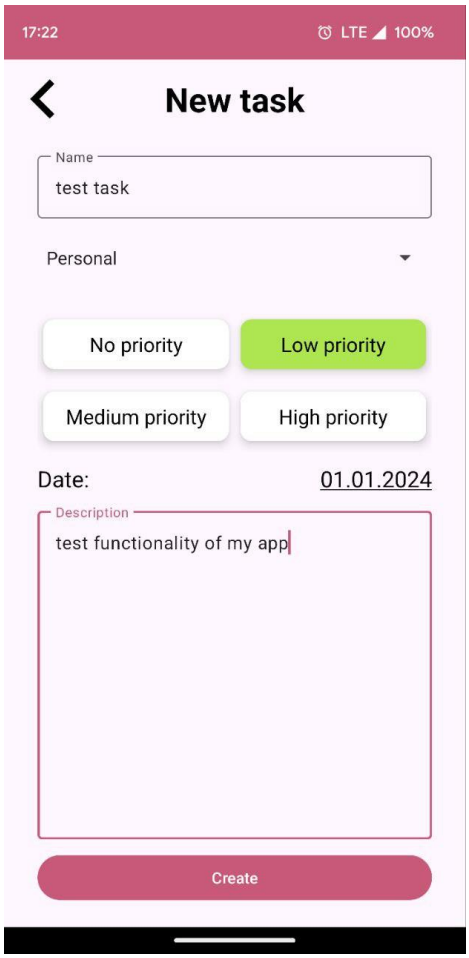

Рисунок 3.3 – Екран створення завдання

Користувач може ввести будь які значення окрім пустого заголовку та створити завдання. Коли в списку завдань є елементи, користувач може їх фільтрувати чи змінювати їх статус (рисунок 3.4).

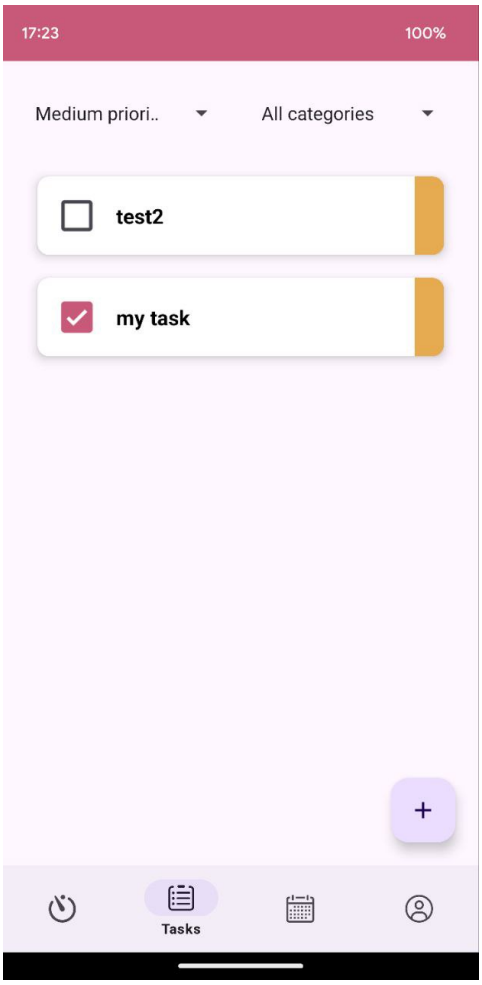

Рисунок 3.4 – Функціональність головного екрана

Також користувач може перейти до інших екранів, користуючись нижньою панеллю навігації. Наприклад, у екрані з календарем, користувач може побачити всі невиконані завдання на дату, яку він обрав (рисунок 3.5).

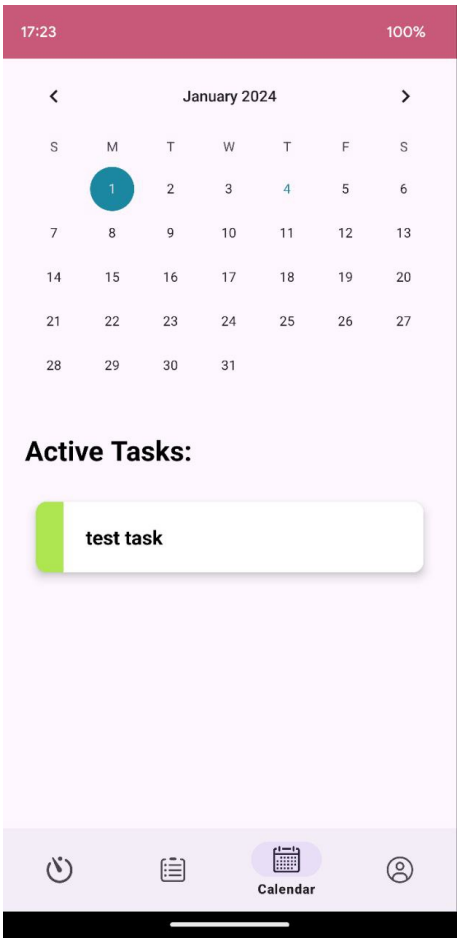

Рисунок 3.5 – Календар

Також, користувач може перейти до екрану фокусування (рисунок 3.6), щоб запустити, зупинити або перезапустити таймер. Також, на цьому екрані є кнопка, яка дозволяє перейти до наступного стану таймера, якщо користувач завчасно закінчив роботу, або не хоче відпочивати.

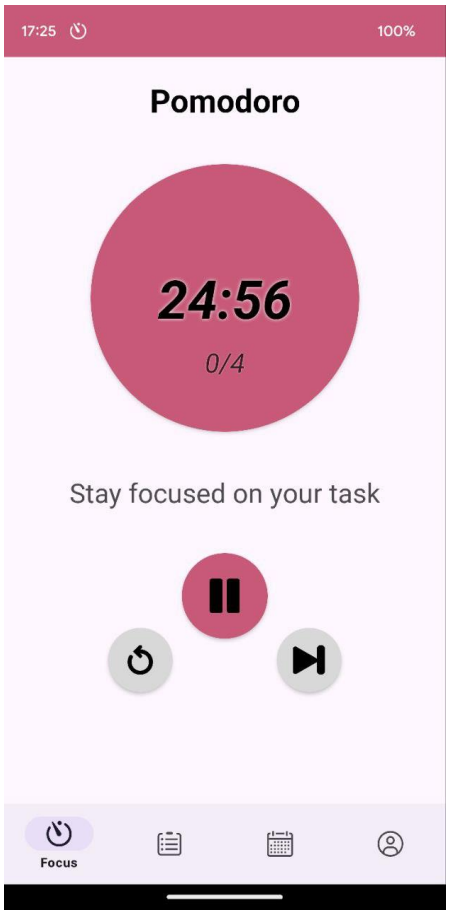

Рисунок 3.6 – Таймер

Ще одним з головних екранів додатку є профіль (рисунок 3.7). У якому користувач може побачити статистику виконаних завдань, кастомізувати нікнейм та фото, вийти з акаунту або перейти до екрану категорій.

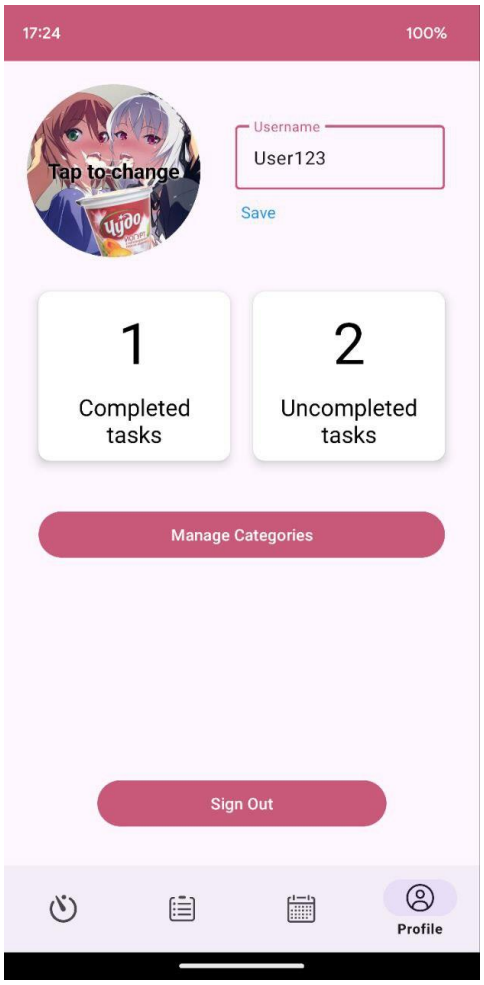

Рисунок 3.7 – Профіль

Екран для управління категоріями (рисунок 3.8), до якого можна перейти з профілю, дозволяє переглядати, додавати або видяляти категорії, які потім будуть використовуватись при створенні або фільтруванні завдань

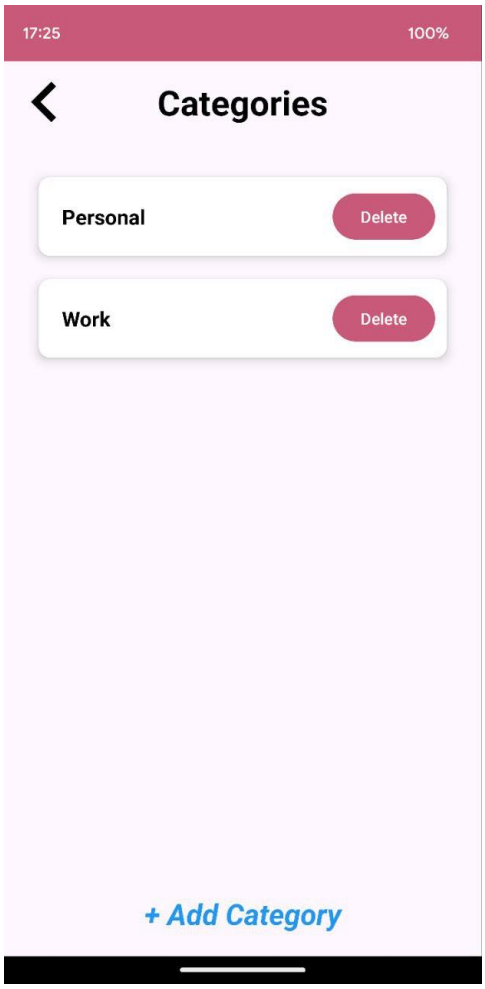

Рисунок 3.8 – Керування категоріями

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на курсову роботу	2	
2	А4	КПІ.ІП-1324.045490.01.81	Пояснювальна записка	51	
3	А4	КПІ.ІП-1324.045490.02.91	Технічне завдання	19	
4	А4	КПІ.ІП-1324.045490.03.34	Керівництво користувача	12	
5	А4	КПІ.ІП-1324.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1324.045490.05.13	Опис програми	30	
7					
8					
9					
10					

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-13.045490.00.91		
Розробн.		Нещерет В. О.			Відомість курсової роботи	Літ.	Лист 1 / Листів 1
Керівн.		Ахаладзе І. Е.					
Консульт						КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІП-13	
Н/контр.							
В.о.зав.каф.							