Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для планування та моніторингу виконання особистих задач і досягнень. Мобільний застосунок.
(тема)

Виконав:
Здобувач _4__ року навчання
групи ПЗПІ-21-1

Єлизавета ПОПОВА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник проф. кафедри ПІ Зоя ДУДАР
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Поповій Єлизаветі Романівні
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для планування та моніторингу виконання особистих задач і досягнень. Мобільний застосунок.
Затверджена наказом по університету від 19.05.2025р. №397Ст
2. Термін подання студентом роботи до екзаменаційної комісії 11.06.2025
3. Вихідні дані до роботи Розробити мобільний застосунок для програмної системи планування та моніторингу виконання особистих задач і досягнень, в якому буде реалізовано функціонал безпосередньо управління особистими задачами, перегляд статистик, отримання досягнень та зокрема функціонал соціальної взаємодії, з використанням мови програмування Kotlin та декларативного інструмента Jetpack Compose
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, огляд розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2025	*виконано*
2	Створення специфікації ПЗ	11.04.2025	*виконано*
3	Проектування ПЗ	17.04.2025	*виконано*
4	Розробка ПЗ	28.04.2025	*виконано*
5	Тестування ПЗ	15.05.2025	*виконано*
6	Оформлення пояснювальної записки	20.05.2025	*виконано*
7	Підготовка презентації та доповіді	23.05.2025	*виконано*
8	Попередній захист	05.06.2025	*виконано*
9	Нормоконтроль, рецензування	06.06.2025	*виконано*
10	Здача роботи у електронний архів	07.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	08.06.2025	*виконано*

Дата видачі завдання « 8 » « квітня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. кафедри ПІ Зоя ДУДАР
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 80 стор., 36 рис., 8 джерел.

ТРЕКЕР ЗВИЧОК, МОТИВАЦІЯ, ЦІЛІ, САМОДИСЦИПЛІНА, СТАТИСТИКА, ПІДТРИМКА, ANDROID ЗАСТОСУНОК, MVVM, KOTLIN, JETPACK COMPOSABLE.

Об’єктом розробки є мобільна програмна система типу «цифровий помічник», призначена для планування, моніторингу та мотивації користувача під час досягнення особистих цілей шляхом використання гейміфікованих механік, системи досягнень і візуальної аналітики продуктивності.

Метою розробки є створення інтуїтивно зрозумілого, адаптивного та масштабованого мобільного застосунку, який поєднує функції особистого планувальника з інструментами відстеження ефективності, гейміфікації та соціальної підтримки.

Метод вирішення передбачає застосування сучасних підходів до розробки клієнтських додатків на платформі Android із використанням архітектурного патерну MVVM (Model-View-ViewModel) у поєднанні з Kotlin та Jetpack Compose для декларативного UI

У результаті розробки створено мобільний застосунок, який включає в себе реалізацію функцій: авторизація користувачів, управління персональними цілями, перегляд статистики, підтримка гейміфікаційних елементів (бали, рівні, нагороди), а також реалізовано інтерфейс для соціальної взаємодії.

ABSTRACT

HABIT TRACKER, MOTIVATION, GOALS, SELF-DISCIPLINE, STATISTICS, SUPPORT, ANDROID APPLICATION, MVVM, KOTLIN, JETPACK COMPOSE

The object of development is a mobile software system of the “digital assistant” type, designed for planning, monitoring, and motivating the user in achieving personal goals through the use of gamification mechanics, an achievement system, and visual productivity analytics.

The purpose of the development is to create an intuitive, adaptive, and scalable mobile application that combines the functions of a personal planner with tools for performance tracking, gamification, and social support.

The solution approach involves the use of modern methods for client application development on the Android platform, utilizing the MVVM (Model-View-ViewModel) architectural pattern in combination with Kotlin and Jetpack Compose for declarative UI.

As a result of the development, there was created mobile application, which includes the implementation of features of a digital assistant: user authentication, personal goal management, statistics review, support for gamification elements (points, levels, rewards), as well as an interface for social interaction.

ЗМІСТ

ВСТУП

Проблема особистої ефективності, раціонального використання часу та досягнення індивідуальних цілей привертає все більшу увагу як у науковому середовищі, так і серед розробників програмних засобів. Попри наявність численних цифрових інструментів для тайм-менеджменту, більшість із них не враховує комплексної природи формування звичок, зниження мотивації, відсутності зворотного зв’язку та соціальної підтримки, що є критичними чинниками на шляху до досягнення довгострокових особистих результатів. Водночас дослідження у сфері поведінкової економіки, психології мотивації та гейміфікації демонструють значний потенціал інтеграції таких підходів у цифрові системи для підвищення рівня користувацького залучення та досягнення реальних змін у поведінці.

Особливу актуальність має створення систем, які поєднують можливості планування, самоконтролю, гейміфікації, соціальної взаємодії та візуалізації прогресу в одному інтерфейсі. Саме така комплексна інтеграція функціоналу дозволяє не лише зберігати дані про цілі, але й активно впливати на поведінку користувача шляхом регулярних нагадувань, елементів суперництва, підтримки друзів, а також системи досягнень.

Метою даної роботи є розробка мобільного застосунку як частини програмної системи для планування та моніторингу виконання особистих задач і досягнень. Застосунок повинен об'єднувати інструменти керування цілями, обліку прогресу, аналізу даних та гейміфікованої взаємодії, що в сукупності забезпечить користувачеві повноцінну цифрову підтримку в процесі особистісного розвитку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Сфера цифрового самоменеджменту є порівняно новим, проте стрімко зростаючим напрямом на перетині інформаційних технологій, поведінкової психології та персоналізованого коучингу. Її основною метою є підтримка користувача у формуванні сталих моделей поведінки, розвитку самодисципліни, досягненні цілей і регуляції психоемоційного стану. З огляду на зростання рівня стресу, фрагментацію уваги, постійну інфоприсутність та високі вимоги до особистої ефективності, цифрові інструменти особистого розвитку стають необхідною частиною повсякденного життя.

Трансформація ринку вказує на відхід від суто інструментальних моделей таких, як звичайні таск-менеджери чи календарі – на користь інтегрованих рішень, які поєднують у собі мотиваційні, аналітичні, рефлексивні та соціальні компоненти. У центрі таких рішень – користувач, як суб’єкт поведінкових змін, що потребує не лише контролю над задачами, а й підтримки, гнучкості, натхнення та зворотного зв’язку.

Поведінкові підходи, зокрема теорії саморегуляції, формування звичок, ефекту свіжого старту (Fresh Start Effect), втрати (Loss Aversion) або соціального зобов’язання (Accountability), активно використовуються в дизайні сучасних цифрових інтервенцій [1]. Відомо, що навіть прості механізми – як-от візуалізація прогресу, регулярне звітування чи гейміфікована оцінка досягнень – можуть істотно підвищувати мотивацію користувача до дій. Водночас ключовим викликом залишається довготривале залучення: користувачі часто втрачають інтерес, якщо не бачать позитивної динаміки або не отримують емоційної винагороди.

Тому особливої уваги набувають інструменти, що не просто фіксують факт виконання дій, а створюють емоційно насичене, персоналізоване середовище підтримки – з гнучкими цілями, соціальною взаємодією, механіками досягнень і рефлексивним аналізом. Саме така логіка закладена в концепцію розроблюваного додатку, який фокусується не лише на зовнішній продуктивності, а на внутрішній мотивації як джерелі стійкого саморозвитку.

Зміна акцентів у потребах користувачів – від продуктивності до емоційного добробуту – зумовлює нову хвилю запитів на додатки, які допомагають не лише "встигати", а й "розуміти, навіщо це робиться". Внутрішня мотивація, усвідомлення прогресу, підтримка з боку однодумців і можливість бачити власну динаміку розвитку – ці чинники стають визначальними в проєктуванні сучасних рішень у галузі self-management.

З технічної точки зору, актуальними стають рішення, які працюють на мобільних платформах, мають адаптивний UX, використовують елементи гейміфікації та соціальних мереж, і водночас зберігають приватність та автономію користувача. Саме мобільні додатки дозволяють створити контекстно-чутливу, завжди доступну систему підтримки, яка може надсилати нагадування, підказки або позитивне підкріплення у ключові моменти.

Окрему увагу варто приділити важливості такого типу додатків в Україні. З урахуванням травматичного досвіду війни, нестабільності та хронічної невизначеності, такі цифрові рішення набувають ще більшої ваги. Вони стають не лише помічниками у плануванні чи продуктивності, а й інструментами емоційного відновлення – допомагають повернути відчуття контролю, стабільності, сенсу в повсякденних діях. Саме тому розробка таких додатків має не лише ринкову, а й соціальну цінність.

Отже, цифровий самоменеджмент у сучасному вигляді – це не просто програмне забезпечення, а форма м’якої поведінкової інфраструктури, яка дозволяє людині не лише планувати майбутнє, але й емоційно його витримувати. Проєктований додаток логічно вписується в цю динаміку, пропонуючи поєднання індивідуальної мотиваційної роботи, візуалізації досягнень та соціальної взаємодії, що разом створюють передумови для глибокого і стійкого особистого розвитку.

1.2 Огляд існуючих систем

Серед великої кількості сучасних додатків для формування звичок та досягнення цілей, на ринку існує декілька помітних рішень, які по-різному підходять до проблеми самодисципліни, мотивації та структурування

повсякденного життя. Кожен із них вирізняється своїм підходом, функціональністю та стилем взаємодії з користувачем, тому варто розглянути основні альтернативи, щоб краще зрозуміти, яким є конкурентне поле та які сильні й слабкі сторони має кожен із гравців.

Одним із найпомітніших прикладів є Habitica [2], яка перетворює буденну рутину на рольову гру. Користувач не просто виконує завдання – він розвиває персонажа, отримує досвід, збирає золото і навіть бореться з монстрами разом із командою однодумців. Завдяки системі нагород та штрафів, виконання справ стає не лише обов'язком, а й грою з викликами. Проте такий підхід може виявитися занадто складним для новачків – велика кількість налаштувань, ігрових механік і взаємодій може перенавантажити того, хто шукає просте рішення. Також не всім користувачам підходить формат гейміфікації – для деяких це виглядає як зайва складність, яка відволікає від суті завдань. Незважаючи на це, відкритий код і потужна спільнота роблять Habitica гнучким і динамічним інструментом, здатним адаптуватися до потреб просунутих користувачів.

З іншого боку, Any.do [3] обирає протилежну стратегію – мінімалізм та інтуїтивність. Завдяки простому інтерфейсу, додаток дозволяє швидко створювати списки справ, встановлювати дедлайни, пріоритети та отримувати нагадування. Особливо зручною є інтеграція з Google Календарем і голосовими командами, що спрощує роботу й економить час. Крім того, можливість командної роботи робить Any.do привабливим для робочих груп та сімейного використання. Водночас, деякі користувачі можуть зіткнутися з тим, що доступні функції надмірні або складні в освоєнні, особливо у платній версії. А частина важливих інструментів, таких як розширене планування або аналітика, доступна лише за підпискою, що обмежує повноцінне використання без витрат.

Coach.me [4] позиціонує себе як додаток не лише для відстеження звичок, а й для особистісного зростання через коучінг. Тут важливу роль відіграє підтримка інших користувачів, а також можливість працювати з персональними тренерами. Такий підхід значно підвищує рівень відповідальності й ефективності, особливо для тих, хто потребує зовнішньої мотивації або порад від фахівців. Coach.me дає

змогу аналізувати свій прогрес за допомогою графіків та вести щоденник, що сприяє глибшому самоспостереженню. Проте не всі користувачі цінують соціальну взаємодію як складову особистого розвитку – для деяких вона може бути зайвою або навіть демотивуючою. Також повний спектр функцій відкривається лише за преміум-підпискою, що робить додаток менш доступним для широкої аудиторії.

Для тих, хто шукає найпростіше рішення без зайвих функцій, Loop Habit Tracker [5] пропонує лаконічний і безкоштовний підхід. Він дозволяє зосередитися лише на ключовому – повторенні звичок і спостереженні за їх прогресом. Завдяки мінімалістичному дизайну та графікам, користувач легко бачить результати і не втрачає час на другорядні налаштування. Однак, саме ця простота для частини користувачів стає недоліком: відсутність гейміфікації, командної роботи та кастомізації робить додаток менш привабливим для тих, хто хоче більше інструментів мотивації та взаємодії. Незважаючи на це, Loop зберігає перевагу у вигляді повної безкоштовності та відсутності реклами, що робить його особливо привабливим для користувачів, які цінують спокій та зосередженість.

Окрему нішу займає Respawn [6], який фокусується на концепції «стекування звичок» – створення послідовних рутин, що складаються з кількох етапів. Це підхід до самоменеджменту, що базується на психологічному ефекті послідовної дії, і особливо корисний для тих, хто хоче структурувати свій день. Завдяки повноекранному режиму фокусування, користувачі можуть уникати відволікань і максимально зосередитися на поточному завданні. Додаток пропонує гнучкі налаштування і систему балів, а також можливість створення соціальних викликів у платній версії. Однак, Respawn доступний виключно на Android, що значно обмежує його потенційну аудиторію. Крім того, різноманіття налаштувань може заплутати новачка, а відсутність командної роботи робить додаток орієнтованим виключно на індивідуальне використання.

Загалом, кожен із розглянутих додатків має власну філософію, орієнтовану на різні категорії користувачів. Habitica приваблює тих, хто цінує гру та спільноту; Any.do – людей, яким важлива чітка організація та зручна інтеграція; Coach.me – користувачів, що прагнуть до особистого розвитку під супроводом; Loop –

прихильників простоти без зайвого; а Respawn – тих, хто шукає методичний, послідовний підхід до рутини. Ці продукти формують розмаїте конкурентне поле, в якому кожен інструмент має свої переваги, проте жоден не є універсальним – вибір залежить від особистих потреб, мотиваційного стилю та очікувань конкретного користувача.

1.3 Виявлення та вирішення проблеми

Сучасні мобільні додатки для самоменеджменту, такі як Loop Habit Tracker, часто обмежуються стандартними шаблонами і не враховують індивідуальні потреби користувача. Відсутність персоналізованого підходу призводить до того, що користувачі не відчувають повного контролю над процесом планування й досягнення власних цілей. Через це інструмент здається зовнішнім, нав’язаним і швидко втрачає актуальність. Рішенням цієї проблеми є впровадження гнучкої системи персоналізації, яка дозволяє адаптувати інтерфейс, формувати власні категорії завдань і пріоритетів, а також узгоджувати функціонал із ритмом життя людини. Коли користувач може створити середовище, яке відповідає його цінностям, стилю мислення та звичкам, додаток перестає бути просто інструментом і перетворюється на особистісного цифрового помічника.

Ще однією поширеною проблемою є зниження довготривалої мотивації. У перші дні користування мобільні додатки викликають інтерес, однак з часом цей ентузіазм згасає. Системи нагадувань, притаманні додаткам на кшталт Any.do або Coach.me, не здатні підтримати людину в періоди емоційного спаду або психологічної втоми. Навіть у гейміфікованих рішеннях, як-от Habitica, мотиваційні механізми не завжди враховують зміну психоемоційного стану користувача, залишаючись статичними. Оптимальне вирішення цієї проблеми полягає у впровадженні адаптивної багаторівневої гейміфікації, що включає не лише зовнішні стимули, як-от бали та досягнення, але й внутрішні фактори – підтримку через надихаючі повідомлення, варіативність щоденних завдань і можливість змагатися з іншими користувачами або ділитися своїми результатами.

Така багатокомпонентна система дозволяє залишатися активним навіть тоді, коли внутрішніх ресурсів недостатньо.

Ще однією слабкою стороною сучасних сервісів є поверхове відстеження прогресу. Більшість із них обмежуються лінійними календарями або базовими графіками, що не дає змоги глибоко аналізувати власну динаміку. У результаті користувач не розуміє, як саме його щоденні дії впливають на досягнення довгострокових цілей. Вирішення цієї проблеми можливе шляхом впровадження аналітичного модуля, здатного візуалізувати продуктивність у різні періоди часу, оцінювати витрачений ресурс, відслідковувати зміни ефективності та формувати висновки, на основі яких можна коригувати особисті стратегії. Такий підхід дає змогу відчути реальну користь від роботи над собою, адже прогрес стає видимим і вимірюваним.

Окрім цього, багато застосунків нехтують соціальним компонентом, що є важливим джерелом підтримки й додаткової мотивації. Коли користувач позбавлений можливості взаємодіяти з іншими, його шлях до саморозвитку стає ізольованим і менш стабільним. Відомо, що наявність спільноти, групи підтримки або хоча б одного партнера з розвитку може суттєво підвищити ймовірність досягнення цілей. Тому доцільно реалізовувати такі функції, як взаємодія з друзями, об’єднання в тематичні групи, участь у спільних викликах або публікація результатів, які формують відчуття залученості до чогось більшого. Завдяки цьому цифрове середовище починає імітувати соціальні зв’язки, підтримуючи людину не лише функціонально, а й емоційно.

Водночас значною проблемою залишається фрагментарність функціоналу. Користувачеві доводиться користуватися кількома окремими додатками: один для планування, інший для трекінгу, ще інший для аналітики або спілкування. Це призводить до когнітивної втоми, необхідності дублювання інформації та втрати цілісної картини. Усунення цієї проблеми можливе лише за умов створення єдиного інтегрованого середовища, яке об’єднує усі ключові функції: планування, аналіз, трекінг, гейміфікацію й соціальну взаємодію. Такий підхід дозволяє уникнути

розсіювання уваги, економить час та зусилля користувача і забезпечує глибше залучення у власний процес розвитку.

Окремо варто зазначити проблему, пов’язану з мотиваційними кризами, що виникають через відсутність відчутних короткострокових результатів. При роботі над довгостроковими цілями прогрес часто не є очевидним у перші тижні або навіть місяці, що викликає розчарування й сумніви в обраній стратегії. Щоб уникнути цього, необхідно створювати систему постійного позитивного підкріплення. Це можуть бути щоденні нагороди, повідомлення про досягнуті проміжні цілі, надихаючі цитати та регулярні нагадування про вже пройдену частину шляху. Навіть дрібні досягнення мають отримувати відповідне відображення, адже саме вони створюють відчуття руху вперед і формують внутрішню впевненість у результаті.

Таким чином, вирішення ключових проблем сучасних мобільних додатків для самоменеджменту можливе лише через комплексний підхід: від глибокої персоналізації та багаторівневої мотиваційної підтримки до потужної аналітики, соціальної взаємодії та інтеграції всіх функцій у єдиному інтерфейсі. Саме поєднання цих рішень дозволяє створити ефективне цифрове середовище, яке не лише організовує процес досягнення цілей, а й підтримує користувача впродовж усього шляху особистого зростання.

1.4 Цільова аудиторія

Цільова аудиторія програмної системи включає повнолітніх користувачів віком від 18 років, незалежно від статі, професії чи рівня технічної підготовки. Ці користувачі зацікавлені в покращенні своєї особистої ефективності, плануванні та досягненні цілей, формуванні корисних звичок або розвитку нових навичок. Аудиторія складається з двох основних груп: новачки та досвідчені користувачі.

Новачки – це користувачі, які вперше використовують додатки для самоменеджменту. Вони можуть мати базову або обмежену технічну підготовку і шукають прості та інтуїтивно зрозумілі інструменти для організації свого часу, формування звичок та досягнення поставлених цілей.

Досвідчені користувачі – це особи, які вже мають сформовані підходи до самоорганізації та планування. Вони володіють певним досвідом у використанні подібних програм і шукають рішення з більш гнучким функціоналом, що дозволяє налаштовувати процеси віовідно до індивідуальних потреб і цілей.

Таким чином, система повинна бути універсальною та доступною як для новачків, так і для досвідчених користувачів, пропонуючи різноманітні функції для підтримки мотивації, відстеження прогресу та персоналізації роботи з цілями. - трошки детальніше

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальний опис

Розробка передбачає створення клієнтської частини у вигляді мобільного застосунку на платформі Android з використанням Kotlin та Jetpack Compose як основного UI-фреймворку. Зв’язок із серверною частиною здійснюватиметься через Retrofit. Серверна логіка буде реалізована на базі ASP.NET Core, що забезпечить надійну обробку даних, авторизацію та зберігання користувацької інформації.

Основна функціональність та її компоненти:

а) автентифікація та реєстрація:
   1) екран для створення нового облікового запису з введенням необхідних даних;
   2) екран входу з підтримкою збереження токена jwt для автоматичної авторизації;
   3) безпечне зберігання токена автентифікації;

б) персональний профіль:
   1) екран для створення нового облікового запису з введенням необхідних даних;
   2) екран входу з підтримкою збереження токена jwt для автоматичної авторизації;

в) управління задачами та цілями:
   1) інтерфейс для створення, редагування та видалення задач і підзадач;
   2) підтримка ієрархії задач та групування їх за категоріями або цілями;
   3) можливість встановлювати дедлайни та отримувати нагадування;

г) аналітика:
   1) графічне представлення статистики виконання задач у вигляді графіків;
   2) візуалізація цілей, прогресу та завершених завдань;

д) гейміфікація:

1) відображення рівня користувача, накопичених балів та віртуальних нагород;
2) доступ до загальної таблиці лідерів для порівняння досягнень з іншими користувачами;

е) соціальні функції:

1) можливість переглядати акаунти інших користувачів;
2) додавання користувачів у список друзів;
3) надсилання мотиваційних повідомлень або цитат друзям;

ж) аі-помічник:

1) екран чату для взаємодії з персональним аі-асистентом;
2) отримання згенерованих задач, порад або мотиваційних повідомлень від асистента;

з) сповіщення:

1) отримання локальних сповіщень про дедлайни, нові рівні, досягнення та повідомлення від друзів.

2.2 Загальні обмеження

Попри заплановану багатофункціональність і розширюваність програмної системи, її початкова версія має ряд технічних та архітектурних обмежень, які необхідно враховувати на етапі розробки та впровадження.

Додаток розробляється виключно для мобільних пристроїв з операційною системою Android. Підтримка інших платформ iOS, не передбачена у першому релізі, але може бути реалізована у наступних версіях.

Весь функціонал потребує активного підключення до Інтернету для синхронізації з сервером.

Передача персональних даних здійснюється виключно через захищені канали. Система автентифікації реалізується із використанням хешування паролів та токенів доступу. Зберігання чутливої інформації на стороні клієнта мінімізується.

Застосунок має бути оптимізований для стабільної роботи навіть на пристроях середнього рівня. Важливо забезпечити помірне споживання енергії, мінімальне використання оперативної пам’яті та швидкий запуск.

2.3 Припущення та залежності

Припущення щодо пристроїв: користувач має пристрій з Android не нижче версії 8.0 (Oreo), що підтримує всі необхідні компоненти Jetpack Compose та Retrofit.

Додаток передбачає наявність стабільного інтернет-з'єднання для обміну даними з сервером, оновлення статистики, соціальних функцій, використання функції розмови з ШІ.

Передбачається, що користувачі мають щонайменше базову мотивацію для саморозвитку та регулярного використання додатку. Цільова аудиторія – особи, що прагнуть організувати повсякденні задачі, розвивати корисні звички та стежити за особистим прогресом.

Серверна архітектура реалізується на основі ASP.NET Core та обслуговує REST API. Уся логіка автентифікації, зберігання даних, обробки статистики та взаємодії між користувачами виконується на серверній стороні.

Можливість масштабування: при проєктуванні враховується потенціал подальшого розширення функціоналу (наприклад, підтримка iOS, інтеграція машинного навчання, додавання офлайн-функціоналу тощо) без необхідності радикальної зміни базової архітектури.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML-проєктування

Розробка діаграми варіантів використання була логічним кроком після формування функціональних вимог до мобільного застосунку. Основна мета цього етапу – візуалізувати, яким чином зареєстрований користувач взаємодіє з системою, і які основні функціональні модулі реалізовані у вигляді окремих сценаріїв використання. Враховуючи попередньо визначені підсистеми – як-от управління задачами, облік досягнень, аналітика, соціальні функції, чат з ШІ – кожна з них була відображена у вигляді окремих use-case. Результуюча діаграма наведена у пункті В.1 додатка В.

Проєктування діаграми почалося з виокремлення одного ключового актора – зареєстрованого користувача. Це зумовлено тим, що функціональність додатку доступна лише після реєстрації, як зазначено у вимогах (див. розділ про автентифікацію та профіль). На етапі аналізу вимог також було передбачено, що користувач повинен мати змогу редагувати власний профіль, переглядати статистику, виконувати задачі, брати участь у викликах, а також взаємодіяти з іншими користувачами – саме ці функції стали базовими варіантами використання.

Підсистема роботи з задачами була реалізована як центральна гілка, що включає перегляд, створення, редагування, видалення та виконання задач. Варіанти використання побудовані з використанням розширень (extend), адже, наприклад, створення чи редагування задачі не обов'язкове під час кожного перегляду, але може бути ініційоване в межах основного процесу. Виконання задачі, у свою чергу, тісно пов’язане з підсистемою досягнень: під час виконання може бути нараховано досягнення, тому тут також використано extend.

Перегляд досягнень та перегляд статистики виокремлені як окремі модулі, що відображають мотиваційно-аналітичну складову застосунку, згідно з функціональними вимогами (див. підрозділ про аналітику та гейміфікацію). Оскільки статистика може деталізуватися за різними критеріями (категорії задач, досягнення тощо), до діаграми додано включення (include) на відповідні підвипадки, що дає змогу уникнути дублювання логіки в моделі.

Важливою особливістю додатку є соціальна взаємодія, яка реалізована через набір функцій, пов’язаних із друзями: пошук, надсилання та перегляд запитів, підтвердження або відхилення, а також перегляд друзів і взаємодія з ними. Ці сценарії включають як логічні include (підтвердження – обов’язкова частина обробки запиту), так і extend – наприклад, підтримка друга не є необхідною дією, але може бути викликана у відповідному контексті.

Інноваційна частина системи – розмова з ШІ-асистентом – відображена як окремий варіант використання. Вона дозволяє користувачеві надсилати запит і отримувати відповідь. Такий підхід структурує діалогову логіку, яку в майбутньому можна масштабувати, наприклад, підключивши обробку запитів на створення задач або рекомендації.

3.2 Проєктування архітектури

У рамках проєктування мобільного додатку одним із ключових завдань було обрати найефективніший архітектурний стиль. Після аналізу всіх вимог до програмного забезпечення було обрано патерн MVVM (Model-View-ViewModel) [7] для побудови клієнтської частини. Вибір цієї архітектури зумовлений необхідністю забезпечити високу масштабованість, чітке розділення відповідальностей, підтримку реактивного підходу до формування інтерфейсу, підвищення рівня тестованості бізнес-логіки, а також спрощення процесу підтримки та розвитку проєкту. У поєднанні з мовою програмування Kotlin, декларативним фреймворком Jetpack Compose та мережею бібліотек (зокрема Retrofit для роботи з API), архітектура MVVM забезпечує ідеальний баланс між гнучкістю, структурованістю коду та можливістю повторного використання компонентів.

Перш за все, застосування MVVM дає змогу чітко розмежувати відповідальність між трьома основними рівнями. Рівень Model відповідає за збереження та обробку даних, включаючи доступ до локальних баз даних або взаємодію з віддаленими серверними API. ViewModel слугує проміжною ланкою, що інкапсулює бізнес-логіку додатку, здійснює обробку запитів, трансформацію даних та зберігає стан інтерфейсу, ізолюючи його від впливу зовнішніх подій.

Рівень View, реалізований за допомогою Jetpack Compose, виконує функцію відображення даних на екрані, реагуючи на зміну станів, що надходять із ViewModel. Такий підхід дозволяє реалізовувати функціонал, наприклад, гейміфікації – зокрема, логіку нарахування балів, обліку досягнень та рівнів – без дублювання коду, що дає змогу відображати одні й ті самі дані у різних частинах інтерфейсу (наприклад, у профілі користувача та таблиці лідерів).

Окремою перевагою MVVM є висока тестованість бізнес-логіки. Оскільки ViewModel є незалежним від Android SDK, її можна легко перевіряти за допомогою юніт-тестів, не запускаючи інтерфейс. Це особливо важливо для модулів, пов’язаних з аналітикою, валідацією або прийняттям рішень, наприклад, при побудові алгоритмів категоризації цілей, розрахунку прогресу користувача, агрегації статистики щодо витраченого часу, середньої ефективності або правильності обчислення нагород. Така модульність не лише підвищує надійність системи, а й спрощує пошук і усунення помилок.

Також варто відзначити природну інтеграцію MVVM із реактивними потоками даних, такими як LiveData, StateFlow чи MutableState, які є невіддільною частиною Jetpack Compose. Завдяки цьому оновлення інтерфейсу відбувається автоматично, без необхідності прямого втручання з боку розробника. Наприклад, у випадку надходження сповіщень про нові виклики або наближення дедлайнів, ViewModel змінює свій стан, і відповідні елементи UI автоматично оновлюються, відображаючи актуальну інформацію в режимі реального часу, що значно покращує користувацький досвід.

Ще однією сильною стороною обраної архітектури є її масштабованість і модульність. Враховуючи наявність у додатку численних функціональних блоків – таких як автентифікація, соціальна взаємодія, гейміфікація, аналітика, планування та моніторинг цілей – можливість розробки кожного з них як окремого модуля із власними компонентами Model, ViewModel та View забезпечує не лише логічну ізоляцію, а й дає змогу організовувати паралельну розробку кількома командами або учасниками проєкту. Це також спрощує підтримку, рефакторинг та повторне

використання коду, наприклад, при реалізації веб-версії системи або створенні нових платформних адаптацій.

Нарешті, архітектура MVVM забезпечує низький рівень зв’язності між компонентами. Оскільки View не має прямого доступу до внутрішньої логіки ViewModel, а остання не залежить від деталей UI, можна гнучко змінювати дизайн інтерфейсу або вносити корективи у бізнес-логіку (наприклад, змінювати правила гейміфікації, валідації чи фільтрації завдань) без ризику порушення цілісності інших частин системи. Така гнучкість істотно знижує загальну складність підтримки проєкту та сприяє створенню стабільного, масштабованого і довготривалого програмного продукту.

Отже, вибір архітектурного патерну MVVM у контексті даного проєкту є обґрунтованим з позиції структурної організації, відповідності сучасним технологічним стандартам Android-розробки, забезпечення якісної взаємодії між бізнес-логікою та UI, підтримки реактивного підходу до управління даними, а також досягнення високого рівня модульності, тестованості та масштабованості системи загалом.

3.3 UI/UX-проєктування інтерфейсу

На етапі створення UI/UX дизайну було розроблено повноцінну систему інтерфейсів, що охоплює ключові функціональні модулі додатку, з урахуванням зручності використання, логічної структури, інтуїтивної навігації та естетики візуального стилю, вигляд дизайну наведено в пунктах Г.1 та Г.2 додатка Г.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Обґрунтування вибору платформи

Для реалізації мобільного застосунку було обрано платформу Android, оскільки вона найкраще відповідає як функціональним вимогам проєкту, так і очікуванням цільової аудиторії. Застосунок належить до категорії персональних трекерів продуктивності, що потребує стабільного, зручного, гнучкого середовища з широкими можливостями кастомізації, повідомлень, фонової синхронізації та роботи з локальними й хмарними даними. Усе це Android забезпечує на найвищому рівні.

Насамперед, Android домінує на світовому ринку мобільних пристроїв, займаючи понад 70% ринку. Це означає, що застосунок зможе охопити найширшу аудиторію користувачів без потреби в окремому бюджеті на розробку iOS-версії. Особливо актуальним це є для країн на кшталт України, де велика частка користувачів смартфонів використовують саме Android. Тобто, з практичного боку, саме ця платформа є найкращим вибором, якщо мета – максимальна доступність застосунку для звичайних користувачів, які хочуть відстежувати особисту ефективність, цілі, звички або щоденні задачі.

Окрім охоплення аудиторії, Android вирізняється гнучкою системою доступу до внутрішніх сервісів пристрою. У контексті розробки трекера задач це надзвичайно важливо, оскільки застосунок має працювати з локальними нагадуваннями, сповіщеннями, фонованими сервісами, інтеграцією з календарем, а також зберігати дані у фоновому режимі. Android надає розробнику повноцінний доступ до таких інструментів, як AlarmManager, WorkManager, Room, NotificationManager, а також дозволяє гнучко керувати дозволами, що відкриває широкі можливості для реалізації персоналізованих сценаріїв взаємодії з користувачем.

Для реалізації застосунку використано мову програмування Kotlin – сучасну, безпечну, лаконічну й офіційно рекомендовану Google. Завдяки підтримці null safety, Kotlin допомагає уникнути типових помилок при роботі з даними, що покращує стабільність застосунку. Це особливо важливо для трекера досягнень,

який має щоденно взаємодіяти з користувачем, не допускаючи збоїв або втрати інформації.

Для створення інтерфейсу було обрано Jetpack Compose – декларативний UI-фреймворк нового покоління. Jetpack Compose дозволяє реалізувати гнучкий та адаптивний інтерфейс, що швидко реагує на зміну стану даних – наприклад, додавання нової задачі, зміна статусу, фільтрація за датами або категоріями. Оскільки застосунок працює з великою кількістю взаємозалежних елементів інтерфейсу (списки, статуси, графіки, нагадування), реактивна природа Compose дозволяє мінімізувати ручне оновлення UI, що пришвидшує розробку та знижує ризик помилок.

Порівняно з кросплатформеними рішеннями на кшталт Flutter або React Native, обраний стек (Kotlin + Jetpack Compose) має важливі переваги:

- краща продуктивність – критично важлива для smooth UX в застосунках із великими списками та великою кількістю дрібних оновлень (на кшталт змін прогресу по задачах, додавання нових елементів, оновлення статусів у реальному часі);
- повна сумісність з Android-сервісами, тоді як у кросплатформеному середовищі часто потрібні «костилі» для доступу до системного функціоналу;
- швидкий запуск MVP, немає потреби тестувати та підтримувати поведінку застосунку одразу на кількох ОС. Це скорочує час і бюджет на розробку;
- можливість гнучкої кастомізації, зокрема тем, анімацій, жестів, кастомних елементів управління тощо – важливо для додатку, який має бути не лише функціональним, але й привабливим для щоденного використання.

Таким чином, платформа Android у поєднанні з Kotlin і Jetpack Compose повністю відповідає технічним вимогам застосунку для відстеження досягнень і задач, забезпечуючи зручність, продуктивність, стабільність і гнучкість. Саме ця екосистема дозволяє реалізувати інтуїтивно зрозумілий, візуально привабливий і технологічно надійний інструмент, що допоможе користувачам ефективно

планувати день, досягати цілей та розвивати власну дисципліну – будь-де й будь-коли.

4.2 Вибір бібліотек

Для реалізації функціональності мобільного застосунку було обрано перевірені, сучасні бібліотеки, які забезпечують стабільну, ефективну й масштабовану розробку.

Для зв’язку між інтерфейсом Jetpack Compose та ViewModel використано бібліотеку зв’язку ViewModel із Compose, яка дозволяє ефективно зберігати стан екранів та дотримуватись принципів архітектури MVVM.

Для взаємодії з API обрано бібліотеку Retrofit – одну з найпопулярніших у Android-розробці. Вона спрощує створення HTTP-запитів. У парі з нею використано Gson-конвертер, який дозволяє перетворювати дані з формату JSON у Kotlin-об’єкти автоматично.

Для зручного логування мережевих викликів застосовується мережевий логер на базі OkHttp.

Для завантаження зображення в інтерфейсі обрано Coil – бібліотеку для роботи з зображеннями, оптимізовану для Jetpack Compose. Вона забезпечує кешування та швидке відображення, що особливо важливо для покращення користувацького досвіду.

Для побудови сучасного дизайну з урахуванням Material Design використано Material-компоненти, а також розширений набір Material-іконок. Для навігації між екранами застосовується Compose-навігація, яка дозволяє зручно описувати маршрути в декларативному стилі.

Для зберігання простих даних, як-от налаштування користувача, використано DataStore – сучасну альтернативу SharedPreferences, що підтримує асинхронну роботу та є безпечнішою у використанні. Асинхронність в застосунку реалізується через короутини Kotlin, які дозволяють працювати з фоновими задачами (мережеві запити, збереження даних тощо) без блокування головного потоку.

Для обробки JWT-токенів використано бібліотеку для розбору JWT, яка дозволяє декодувати токени локально без звернення до сервера.

Для реалізації жестів «потягни, щоб оновити» використано Accompanist SwipeRefresh – легке рішення для оновлення контенту в стилі Android. Також для побудови адаптивних UI-елементів, зокрема тегів і гнучких макетів, застосовується Accompanist Flow Layout.

Для відображення статистики й прогресу реалізовано MPAndroidChart – бібліотеку для створення діаграм і графіків, яка підтримує різні типи графіків (лінійні, кругові тощо).

Для забезпечення сумісності з новими можливостями Java (наприклад, API java.time) на старіших версіях Android використовується бібліотека десугарингу JDK. Вона дозволяє використовувати нові функції мови навіть на пристроях зі старими SDK. Такий набір інструментів забезпечує надійність, продуктивність і зручність розробки, а також створює всі необхідні передумови для комфортного користування застосунком у повсякденному житті.

5 ОГЛЯД РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ

5.1 Загальна структура проєкту

Як зазначалось на етапі проєктування, розробка мобільного застосунку буде виконана відповідно до принципів MVVM-архітектури з чітким розділенням відповідальностей між різними шарами. На рисунку 5.1 зображено вигляд структури проєкту у середовищі розробки Android Studio.

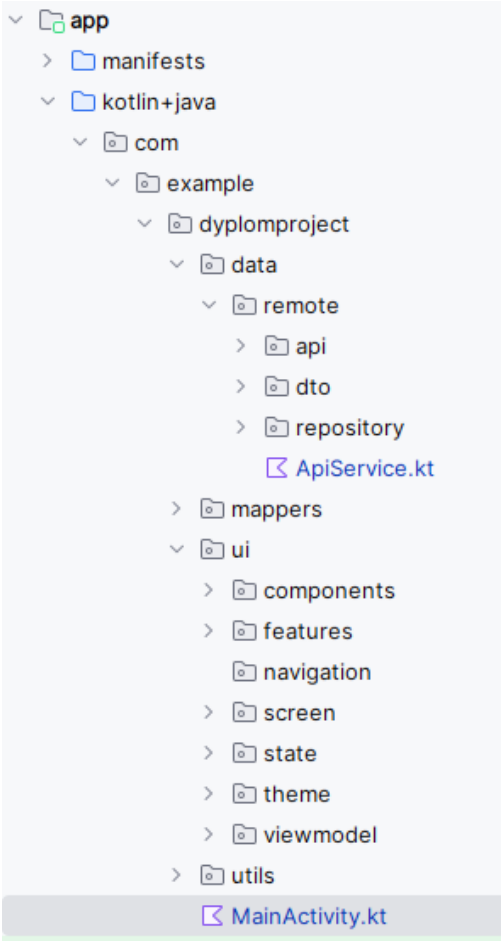

Рисунок 5.1 – Вигляд структури проєкту

Структура розробленого проєкту містить такі пакети:

– data – містить усе, що стосується роботи з даними: DTO-структури (data.remote.dto), які описують формат обміну з сервером, а також репозиторії (repository), які реалізують доступ до API;
– repository – виступає посередником між зовнішніми джерелами даних і бізнес-логікою. Кожен функціональний блок (аутентифікація, задачі, друзі) має окремий репозиторій, наприклад, AuthRepository,

UserRepository тощо;

- viewmodel – реалізує логіку станів і обробку подій користувача. ViewModel отримує дані з репозиторіїв, обробляє їх і передає в UI у вигляді UiState, що забезпечує реактивне оновлення інтерфейсу;
- ui/screens – містить Composable-функції, що формують інтерфейси окремих екранів (наприклад, екрани авторизації, задач, друзів);
- components – набір повторно використовуваних UI-елементів: кнопки, форми, діалоги, картки задач, тощо. Вони ізольовані від бізнес-логіки і створені для гнучкого повторного використання;
- theme – включає стилі, кольори, типографіку, що визначають загальний вигляд додатку у стилі Jetpack Compose

5.2 Запити до сервера

Запити до віддалених ресурсів організовано через репозиторії, що виступають єдиною точкою входу для зовнішніх даних у ViewModel. Для кожного основного домену функціональності реалізовано відповідну обгортку – AuthRepository, UserRepository, TaskRepository, NotificationRepository, тощо. Репозиторії інкапсулюють логіку звернень до API, виконують обробку результатів та ізолюють зовнішню взаємодію від бізнес-логіки та інтерфейсу.

Кожен репозиторій розроблений з урахуванням стабільності та відмовостійкості: запити репозиторіїв обгорнуті в блоки try/catch, а результати повертаються в уніфікованому форматі через Result<T>, що дозволяє ViewModel ефективно управляти помилками й адаптивно реагувати на зміну стану. У випадку збою мережевого або серверного – система не припиняє роботу, а інформує користувача відповідним повідомленням, зберігаючи інтерактивність і функціональність інтерфейсу.

У межах архітектури MVVM, репозиторії не зберігають стан, а виступають виключно як провайдери даних. Уся логіка викликів і трансформації результатів знаходиться у ViewModel, що підтримує чистоту архітектури та полегшує супровід коду. Такий підхід дозволяє легко масштабувати додаток, змінювати джерело даних

(наприклад, перейти від API до локального кешу) без порушення внутрішніх зв’язків між компонентами.

Комунікація з сервером відбувається в асинхронному режимі – API-функції оголошено як suspend, що дозволяє не блокувати головний потік і зберігати високу продуктивність додатку навіть у випадку великої кількості паралельних запитів. Це особливо важливо для сценаріїв з активною взаємодією - таких як чат, оновлення завдань, перевірка сповіщень тощо.

Для централізації та спрощення роботи з HTTP-запитами реалізовано окремий об’єкт RetrofitInstance (програмний код наведено у пункті Є.1 у додатку Є), який відповідає за створення та конфігурацію Retrofit-клієнта. Це єдиний точковий компонент, через який ініціюються всі звернення до віддаленого серверу, що забезпечує контрольованість і узгодженість мережевої взаємодії в межах усього додатку.

Базова адреса API задається централізовано, із можливістю легкої адаптації під різні середовища – емулятор, фізичний пристрій, або локальний сервер. Це дозволяє гнучко перемикатись між конфігураціями під час розробки без зміни структури запитів.

У складі Retrofit-конфігурації використовується OkHttpClient, доповнений HttpLoggingInterceptor, який дозволяє детально відслідковувати весь HTTP-трафік. Логування встановлено на рівні BODY, що дає змогу бачити повну інформацію про запити та відповіді, включно з тілами – це істотно полегшує дебагінг і виявлення помилок у мережевій взаємодії.

Особливу увагу приділено поліморфній серіалізації/десеріалізації складних моделей, зокрема задач і підзадач. Для цього застосовано механізм RuntimeTypeAdapterFactory з бібліотеки Gson, який дозволяє зберігати типову структуру об'єктів навіть у випадку різнорідних сутностей. Наприклад, TaskDto і TaskCreateDto можуть бути представлені у вигляді підтипів (standard, repeatable, scale, with_subtasks), і кожен тип коректно обробляється як при надсиланні, так і при отриманні даних із серверу.

Аналогічна схема реалізована й для підзадач (SubtaskDto, SubtaskCreateDto), що дозволяє гнучко працювати з різними варіантами внутрішньої структури задач, не порушуючи загальної архітектури об'єктів даних. Реєстрація всіх типів здійснюється в об’єкті Gson, який потім використовується як конвертер у Retrofit.

Готовий екземпляр API-сервісу (ApiService) створюється через by lazy, що гарантує його ініціалізацію лише за першого звернення. Це підвищує ефективність споживання ресурсів і забезпечує зручність при використанні в різних частинах програми – будь-який компонент, що потребує доступу до API, може безпосередньо звернутися до RetrofitInstance.api.

Об’єкт RetrofitInstance виконує роль високорівневого конфігуратора, який ізолює всі технічні деталі взаємодії з мережею, забезпечує підтримку складних ієрархій даних, логування запитів, централізоване управління базовою адресою і повну відповідність архітектурному підходу clean MVVM.

5.3 Навігація в застосунку

Навігація у додатку реалізована за допомогою Navigation Compose. Основна логіка навігації зосереджена в єдиній активності – MainActivity, що відповідає сучасним підходам до побудови Android-застосунків із використанням Jetpack Compose. Вибір вищезгаданої бібліотеки дозволив ця дозволило побудувати зрозумілу та масштабовану архітектуру маршрутів, уникнути фрагментів і побудувати весь UI на основі декларативного підходу.

Використання NavHost стало базовою точкою маршрутизації між екранами, програмний код функції в якій використовується NavHost наведено у пункті Є.2 у додатку Є. Навігацію було структуровано так, аби розділити основні вкладки: статистику, задачі, друзів і чат-помічник – в окремий вкладений навігаційний граф. Такий підхід робить код логічнішим і легким у підтримці. Кожна з цих вкладок має окрему візуальну складову з урахуванням адаптивності інтерфейсу. Для навігації між ними реалізовано нижнє меню (Bottom Navigation), що забезпечує швидкий доступ до ключових функцій застосунку.

Другорядні екрани, такі як профіль, запити в друзі, досягнення, сповіщення та налаштування, винесені в окрему вкладену гілку навігації з маршрутом secondary. У такий спосіб було чітко відокремлено основний функціонал від додаткового, не перевантажуючи основний навігаційний простір. Відкриття таких екранів здійснюється через іконки у верхній панелі (TopAppBar) або через внутрішню логіку взаємодії між елементами інтерфейсу.

Окрему увагу було приділено зручності користувача під час перемикання між екранами. Наприклад, при переході з одного основного екрану на інший, було забезпечено очищення стека попередніх маршрутів, аби уникнути небажаного «накопичення» в навігаційному дереві. Також реалізовано підтримку збереження стану екранів, що є важливим для трекера задач – користувач не втрачає контекст після перемикання.

Для гнучкості й масштабованості логіку авторизації та визначення маршруту за замовчуванням (головна або екран входу) було винесено безпосередньо в NavHost, залежно від стану автентифікації, який отримується з відповідного ViewModel. Цей підхід дозволяє правильно маршрутизувати як нових, так і авторизованих користувачів.

Таким чином, навігаційна архітектура застосунку є ієрархічною та модульною, побудованою з урахуванням можливості масштабування. У майбутньому вона легко може бути розширена шляхом додавання нових маршрутів до відповідного навігаційного графа без необхідності переписування основної логіки. Також було забезпечено повну ізоляцію бізнес-логіки ViewModel-ів від UI-рівня, що спрощує підтримку та тестування навігаційних сценаріїв.

5.4 Автентифікація та збереження сесії

Автентифікація реалізована через окремі екрани для входу та реєстрації, що забезпечують введення необхідних даних користувача (вигляд екранів наведено у пунктах Д.1, Д.2 у додатку Д). Зазначені екрани використовують спільний репозиторій AuthRepository для надсилання запитів на сервер, програмний код AuthRepository наведено у пункті Є.3 у додатку Є. Кожен із цих екранів працює з

відповідним станом, у якому зберігаються введені значення, можливі помилки валідації, прапорець завантаження та повідомлення про помилки. Користувач має змогу вводити електронну пошту, пароль, а під час реєстрації також дату народження, стать, нікнейм та персональне повідомлення. Усі поля автоматично перевіряються на валідність перед відправленням запиту до серверної частини.

Після підтвердження даних виконується запит до віддаленого API з відповідними параметрами. У разі входу запит містить пошту та пароль, у разі реєстрації – повний набір даних користувача. Відповідь сервера містить токен доступу, який зберігається у локальному сховищі пристрою для подальшого використання. Це дозволяє реалізувати збереження сесії між запусками додатку.

Збереження токена реалізовано за допомогою окремого менеджера сховища, який використовує механізм DataStore. Він відповідає як за збереження, так і за очищення токена під час виходу з системи. При кожному запуску застосунку зчитується токен зі сховища та виконується його декодування. Якщо токен дійсний, автоматично визначається ідентифікатор користувача, що дозволяє пропустити етап автентифікації та перейти безпосередньо до головного інтерфейсу програми.

Токен зберігається у вигляді текстового значення, яке є частиною JWT. Для отримання ідентифікатора користувача з токена виконується його розбір: із середньої частини (payload) видобувається інформація, яка містить унікальний ідентифікатор користувача. Це дає змогу надалі здійснювати запити до приватних частин API, пов’язаних із персоніфікованими даними.

У випадку, якщо користувач виходить із системи, токен видаляється зі сховища, а внутрішній стан автентифікації оновлюється, що призводить до повернення на екран входу. Таким чином, стан сесії синхронізовано з локальним сховищем і не потребує додаткових дій з боку користувача.

Оновлення інтерфейсу при зміні стану автентифікації відбувається автоматично завдяки реактивній моделі – як тільки зберігається або очищується токен, змінюється відповідний стан автентифікації, що активує відповідні переходи між екранами.

Загальна архітектура побудована так, щоб централізовано зберігати та обробляти інформацію про стан користувача, мінімізуючи повторення логіки та спрощуючи підтримку. Усі дії, пов’язані з перевіркою користувача, входом у систему, виходом та збереженням сесії, виконуються узгоджено на основі поточного стану токена та пов’язаного з ним ідентифікатора.

5.5 Огляд розробки екрана «Друзі»

Екран «Друзі» реалізує ключовий функціонал соціальної взаємодії в додатку (вигляд екран наведено на рисунку Д.3 у додатку Д). Він забезпечує можливість перегляду поточних друзів, ознайомлення з усіма користувачами системи, фільтрацію за категоріями, пошук за іменем, надсилання запитів у друзі, а також підтримку вже існуючих контактів. Програмний код FriendsViewModel – ViewModel яка реалізує бізнес логіку поточного екрана наведено у пункті Є.4 у додатку Є.

Користувацька взаємодія адаптується відповідно до обраної вкладки, де у фокусі може бути або перелік поточних друзів, або список усіх користувачів. Залежно від контексту, динамічно змінюється набір доступних дій: можливість підтримати друга, надіслати запит на дружбу, або застосувати фільтри для пошуку за категоріями. Перемикання між вкладками супроводжується оновленням вмісту, скиданням пошукових параметрів і завантаженням релевантних даних. Дані кешуються, що дозволяє уникати зайвих запитів до сервера при повторному перегляді.

Пошук користувачів реалізовано через текстовий запит із можливістю комбінації з категоріальною фільтрацією. У разі зміни пошукових умов або категорії, відбувається оновлення вмісту списку відповідно до нових параметрів. Категорії представлені у вигляді окремого елемента інтерфейсу, що дозволяє гнучко обмежити результати за тематикою чи роллю користувача. Тестування функції пошуку друзів з використання поля вводу та фільтрацією наведено на рисунку Д.4 у додатку Д.

Функціонал додавання у друзі передбачає можливість надіслати запит, який змінює статус користувача у списку та сигналізує про активну взаємодію. У випадку

поточних друзів реалізовано підтримку – окрему дію, яка відзначає особливу форму соціальної активності. Усі дії з користувачами супроводжуються візуальними індикаторами, що інформують про поточний стан: чи є користувач другом, чи вже надіслано запит, чи отримано підтримку.

Списки користувачів формуються динамічно, реагуючи на зміну параметрів стану. Користувач має змогу оновити дані вручну, використовуючи жест оновлення – при цьому система самостійно визначає, які дані потребують перезавантаження на основі активного контексту.

Інтерфейс користувача включає спеціалізовані елементи відображення для кожного типу взаємодії – як для поточних друзів, так і для загального списку користувачів. Вони відрізняються набором доступних дій та візуальним оформленням залежно від статусу зв’язку.

Окрім надсилання запитів у друзі, користувач може переглянути свій список запитів у друзі та відреагувати на них на додатковому екрані «Запити у друзі» (вигляд екрана наведено у пункті у Д.5 додатку Д).

5.6 Огляд розробки функціоналу екрана «Задачі»

Екран «Задачі» реалізує функціональність організації та управління особистими завданнями (вигляд екрана наведено у пункті у Д.3 додатку Д).. Основний поділ завдань побудований навколо структури «папок», кожна з яких містить власний набір задач. Для цього використовується компонент FolderTabs, що дозволяє створювати, редагувати, обирати та видаляти папки. Горизонтальний список табів побудований на основі LazyRow, з реакцією на короткі та довгі натискання.

Після вибору активної папки, у секції FolderContent відображаються задачі, згруповані за статусами: «В процесі» та «Виконано». Для завершених задач враховуються всі підстатуси завершення (Completed, CompletedInTime, тощо). Порожня папка виводить інформаційне повідомлення з підказкою щодо створення нових задач.

Інтерфейс екрану «Задачі» побудовано на реактивному рендерінгу: TasksUiState містить поточну вибрану папку, перелік усіх папок з прив’язаними задачами, статус завантаження, а також список категорій для задач (завантажується окремо при ініціалізації.

Кожна секція задач представлена через компонент TaskSection, який дозволяє згорнути/розгорнути список задачі. Задачі рендеряться за допомогою TaskDisplay (програмний код наведено у пункті Є.5 у додатку Є), який динамічно адаптується до типу задачі – стандартної, зі шкалою, повторюваною або з підзадачами . Для кожної задачі підтримується режим редагування, відображення додаткової інформації, а також в залежності від типу задачі можлива взаємодія з підзадачами, днями повторення або шкалою. Якщо задача має тип «зі шкалою», відображається поле для введення значення, кнопки інкременту та декременту, а також компонент шкали, який демонструє прогрес виконання задачі у вигляді відношення поточного значення до цілі. Якщо підзадача має тип «повторювальна», тоді виводиться інтерфейс для відмітки днів виконання тренувань. Компонент відмічання тренувань дозволяє вибирати дні тижня та реєструвати відмітки. Якщо задача має тип «з підзадачами» то відповідно в цій можна управляти ще й вкладеними підзадачами. Важливо зауважити, що тільки задача може містити в собі підзадачі, а власне сама підзадача не може мати далі ієрархію з вкладеними в неї підзадами. У додатку Д, на рисунку Д.7 наведено вигляд задач типу «повторювальна» та «зі шкалою».

Компонент SubtaskDisplay в свою чергу відповідає за відображення окремої підзадачі в інтерфейсі користувача та реалізує логіку взаємодії з нею, враховуючи тип підзадачі. Як і батьківський компонент задача, компонент підзадача динамічно адаптується залежно від типу: стандартна, зі шкалою або повторювальна

Форма створення нової задачі доступна через TaskCreationForm, яка відкривається натисканням кнопки з написом «Створити задачу» (вигляд форми наведено на рисунку Д8 та Д9). Логіка форми включає базову валідацію полів залежно від типу задачі, наприклад, для повторюваних задач обов’язкове поле з днями тижня. Користувач може надати таку інформацію для створення задачі: ім’я

(обов’язкове), опис, теги, категорія (обов’язкова), дата дедлайну, дата та час нагадування, тип (обов’язковий).

Такі запити до API, як завантаження папок, задач, категорій, оновлення, створення, видалення – централізовано обробляються в TasksViewModel. Для кожної дії передбачено відповідні методи (updateTask(), deleteTask(), updateTaskStatus(), тощо), що змінюють стан екрану через MutableStateFlow.

Редагування папок реалізовано через FolderEditBottomSheet, який дозволяє перейменувати або видалити обрану папку. Стан нижнього листа (showBottomSheet) управляється локально в TaskScreen.

Завдяки розподілу на такі окремі невеликі компоненти, які було описано вище, вдалось зберігти чистоту, зрозумілість та послідовність коду.

5.7 Огляд розробки функціоналу екрана «Статистики»

Екран «Статистики» є ключовим інструментом аналітики в додатку, що дозволяє користувачу оцінити свій прогрес у досягненні цілей, проаналізувати активність у різних категоріях, а також порівняти власні результати з даними друзів і глобальної спільноти. Архітектура екрана побудована на централізованому управлінні станом, що забезпечує чітку синхронізацію інтерфейсу з даними та реагування на взаємодії користувача.

Користувацька взаємодія організована через вкладки, що представляють три основні контексти: особисту статистику, статистику друзів і глобальні показники. Залежно від обраної вкладки, екран динамічно оновлює візуалізацію даних – діаграму категорій, лінійну діаграму продуктивності, блоки статистики та таблицю лідерів. Перемикання між вкладками супроводжується завантаженням відповідного набору даних, скиданням попереднього контексту та реактивним оновленням інтерфейсу.

Інтерфейс складається з 4 візуальних блоків екрана: діаграма розподілу категорій, картки зі зведеними показниками, лінійна діаграма продуктивності таблиця лідерів.

Діаграма розподілу категорій (Piechart) відображає розподіл виконаних задач у кожній категорії (див. рис. Д.10 додатка Д). Даний блок дозволяє користувачу зрозуміти, які сфери є найактивнішими, а які – потенційними зонами росту. Діаграма змінюється залежно від обраної вкладки. Програмний код, який відповідає відображенню графіка на екрані наведено у пункті Є.6 додатка Є;

Картки зі зведеними показниками демонструє кількість користувачів, днів у додатку, загальна кількість задач та відсоток виконаних (див. рис. Д.11 додатка Д). Ці статистики надають узагальнену інформацію та дозволяють швидко оцінити основні метрики;

Лінійна діаграма продуктивності відображає зміну кількості виконаних задач по місяцях (див. рис. Д.12 додатка Д). Діаграма дає змогу відстежити динаміку прогресу та виявити періоди зростання або спаду. Програмний код, який відповідає за відображення графіка на екрані наведено у пункті Є.7 додатка Є.

Таблиця лідерів демонструє найактивніших користувачів платформи (див. рис. Д.13 додатка Д). Вона містить фото (за наявності), ім’я, нікнейм та кількість балів, формуючи мотиваційний елемент через соціальне порівняння.

Кожна секція супроводжується коротким описом, що розкриває її ціль і мотивує користувача до подальшого розвитку. Вся взаємодія реалізована через реактивну модель – при зміні вкладки, оновленні або повторному вході користувача, компоненти екрану автоматично перебудовуються відповідно до нового стану.

Механізм Swipe-to-Refresh дозволяє користувачу вручну оновити дані, при цьому застосунок самостійно визначає, який саме набір інформації потребує оновлення. Завдяки кешуванню, повторний доступ до даних не спричиняє зайвих запитів до сервера, що позитивно впливає на продуктивність і знижує навантаження.

Інтерфейс оформлений у відповідності до загальної стилістики додатку: використані брендові кольори, іконки та типографіка, а також спеціалізовані компоненти, що покращують сприйняття даних. Візуальна ієрархія та просторове

компонування дозволяють легко орієнтуватися в інформації, незалежно від обсягу представлених статистичних даних.

У підсумку, екран «Статистика» поєднує функціональність, візуальну привабливість та аналітичну глибину. Завдяки гнучкому підходу до представлення інформації та інтерактивності, він стимулює користувача до саморефлексії, соціальної активності та досягнення нових цілей.

5.8 Огляд розробки функціоналу екрана «Чат з помічником»

Екран «Чат з помічником» реалізує функцію персонального діалогу між користувачем і вбудованим інтелектуальним агентом (див. рис. Д.14 додатка Д). Він забезпечує зручне середовище для текстової взаємодії, перегляду історії спілкування та створення задач безпосередньо з повідомлень. Вся логіка побудована з урахуванням принципів модульності та ефективної синхронізації даних, що дозволяє гнучко реагувати на дії користувача.

При завантаженні екрана автоматично отримується історія попереднього листування. Повідомлення виводяться у вигляді діалогу, де кожна сторона має свій візуальний стиль: репліки користувача вирівнюються праворуч, а відповіді помічника – ліворуч. Усі повідомлення форматуються з урахуванням ролі автора, часу створення та специфіки запиту (звичайне чи пов’язане із задачами).

Інтерфейс введення включає текстове поле, кнопку відправлення та додатковий перемикач, що дозволяє користувачеві позначити запит як потенційну задачу. У разі активованого перемикача система по-іншому інтерпретує запит і формує відповідь із урахуванням потреб створення завдання.

Обробка повідомлень виконується у фоновому режимі: одразу після надсилання повідомлення додається до локального списку, паралельно відправляючись на сервер для генерації відповіді. По завершенню обробки, відповідь помічника додається до діалогу. Уся комунікація побудована на основі асинхронного виконання, що дозволяє підтримувати високу швидкість відгуку та плавність інтерфейсу.

Особливістю екрана є підтримка опції створення задач: при ввімкненому відповідному параметрі користувач може ініціювати формування завдання напряму із тексту повідомлення. Це відкриває шлях до глибшої інтеграції помічника з іншими модулями додатку, зокрема з системою управління завданнями.

Візуальна складова відрізняється простотою та функціональністю. Компоненти повідомлень мають адаптивне оформлення, що змінюється залежно від ролі відправника. Повідомлення згруповані за хронологією, останні репліки завжди у фокусі – завдяки автоматичному прокручуванню списку. Інтерфейс також передбачає обробку винятків – з можливістю розширення для відображення повідомлень про помилки в мережі або збої при надсиланні.

Екран «Чат з помічником» об'єднує функціональність діалогу, інтелектуальної підтримки та потенційної генерації задач. Його реалізація забезпечує адаптивну взаємодію, зручне оновлення даних і чіткий розподіл обов’язків між логічними компонентами системи.

5.9 Огляд розробки функціоналу екрана «Повідомлення»

Екран «Повідомлення» забезпечує користувача своєчасною інформацією про ключові події у додатку (див. рис. Д.15 додатка Д). Основні функції екрана – перегляд нових і прочитаних повідомлень, оновлення вмісту за запитом користувача, а також позначка всіх повідомлень як прочитаних. Інтерфейс адаптується відповідно до контексту та поточного статусу повідомлень.

У фокусі екрана знаходиться поділ повідомлень на «Нові» та «Прочитані», що візуально структурує взаємодії користувача. Нові повідомлення відображаються у спеціальному блоці з акцентованим фоном, який дозволяє швидко розпізнати та обробити їх. Одним натисканням користувач може відзначити всі нові повідомлення як прочитані.

Завантаження повідомлень виконується автоматично при відкритті екрана, а також може бути ініційоване вручну за допомогою жесту «потягни, щоб оновити». Після оновлення відображаються актуальні дані відповідно до статусу кожного повідомлення.

Якщо повідомлень немає, інтерфейс інформує про це користувача через центроване повідомлення, що підтримує чистоту візуального оформлення. Якщо ж повідомлення наявні, вони відображаються у вигляді зручного списку з прокручуванням, згрупованого за статусом.

Кожен елемент повідомлення включає іконку, короткий опис тексту та стилізоване оформлення відповідно до загальної візуальної концепції застосунку. Нові повідомлення мають окреме стилістичне виділення для привернення уваги.

Користувач має змогу швидко оновити список повідомлень, переглянути їхній вміст або очистити нові повідомлення. Уся логіка взаємодії реалізована через ViewModel, яка працює з репозиторієм сповіщень, керує завантаженням, оновленням і змінює статус повідомлень.

Таким чином, екран «Повідомлення» пропонує зручну, логічно структуровану та гнучку у використанні взаємодію.

5.10 Огляд розробки функціоналу екрана «Профілю користувача»

Екран «Профіль користувача» реалізує персоналізований перегляд інформації про користувача, з акцентом на його прогрес, опис, соціальні зв’язки та ігрові досягнення (див. рис. Д.16 та Д.17 додатка Д). Основна мета екрана – надати повний контекст про активність і статус користувача в системі, а також забезпечити зручну навігацію до інших функцій профілю.

Після завантаження профілю відображається нікнейм користувача, його аватар, повне ім’я, рівень та XP-прогрес із візуальним індикатором заповнення. Особливу увагу приділено стилістичному оформленню: кольорові індикатори, типографіка, вирівнювання і контури компонентів витримані в єдиній естетиці. Опис користувача представлено у вигляді текстового блоку з підтримкою вирівнювання за шириною для покращення читабельності.

Інформація про рівень включає числове позначення, шкалу прогресу з градієнтним заповненням, а також розрахунок досвіду (XP), що відображає загальний обсяг до наступного рівня. Це сприяє гейміфікації взаємодії та мотивації до подальшого розвитку.

Блок досягнень представлений у вигляді окремого сегменту інтерфейсу, з візуально виділеним заголовком та картками досягнень. Досягнення мають різну стилізацію залежно від статусу (звичайні або рідкісні) – рідкісні маркуються градієнтним фоном.

У разі відсутності досягнень система інформує користувача відповідним повідомленням. Підхід до компонування елементів у сітці дозволяє адаптувати інтерфейс як до повного списку, так і до обмеженої кількості досягнень без порушення структури.

Окремий блок відведено списку друзів користувача, що дозволяє переглянути пов’язаних користувачів прямо в межах профілю. Друзі представлені у вигляді компактних карток з можливістю взаємодії – це дозволяє швидко ідентифікувати соціальні зв’язки без переходу до окремого екрану.

Завершує екран функція виходу з облікового запису, представлена у вигляді текстової кнопки зі стилізованим оформленням, розміщеної в нижній частині екрана. Розміщення забезпечує легкий доступ, не перевантажуючи основну область профілю.

Завдяки структурованому поділу на логічні блоки – аватар, базова інформація, рівень, досягнення, друзі – користувач отримує чітку ієрархію контенту та зручну взаємодію з профілем.

5.11 Огляд розробки функціоналу екрана «Досягнення»

Екран «Досягнення» забезпечує користувача деталізованим переглядом усіх його досягнень у системі, впорядкованих за категоріями або типами (див. рис. Д.18 додатка Д). Основна мета: візуалізувати прогрес, відзначити вже здобуті досягнення та стимулювати подальшу активність через інтерфейсну гейміфікацію.

Інтерфейс реалізовано як прокручуваний список карток досягнень, кожна з яких містить назву, опис, статус виконання, а також візуальні елементи - іконку та рамку зі стилізацією залежно від типу досягнення (наприклад, звичайне або рідкісне). Для рідкісних досягнень використовується градієнтне оформлення або спеціальна обводка, що підкреслює їх винятковість.

На кожній картці присутня кнопка «Отримати», кнопка є активною до поки користувач нею не скористається та не отримає бали за досягнення. Після взаємодії її стан змінюється на «Отримано», що знижує навантаження на користувача і не провокує повторних дій.

Список досягнень адаптивно оновлюється, якщо у користувача з’являється нове досягнення або змінюється статус поточного. При цьому зберігається цілісність візуальної структури, незалежно від кількості елементів.

Усі елементи екрана структуровані за допомогою сітки, що забезпечує стабільність розташування карток навіть при зміні розміру екрана чи кількості досягнень. Інтерфейс узгоджено з іншими екранами застосунку – іконографіка, типографіка та принципи відображення відповідають загальній стилістиці.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проаналізовано сучасний стан і проблематику цифрового самоменеджменту, зокрема труднощі, з якими стикаються користувачі при організації особистої ефективності, плануванні завдань та досягненні цілей. Проведений огляд наявних мобільних рішень дозволив виявити їхні недоліки та відсутність комплексного підходу до управління власним часом, ресурсами та мотивацією.

На основі отриманих результатів досліджень та аналізу було спроєктовано та розроблено мобільний додаток, що враховує виявлені потреби користувачів та реалізує нові функціональні можливості, спрямовані на покращення процесу самоменеджменту. Додаток поєднує інструменти планування, моніторингу прогресу, мотиваційної підтримки та персоналізації взаємодії, що робить його перспективним засобом для підвищення особистої продуктивності. Створене програмне рішення може стати основою для подальшого тестування, вдосконалення та впровадження у практичну діяльність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jangra, Hitkul, Rajiv Ratn Shah, and Ponnurangam Kumaraguru. 2024. “Put Your Money Where Your Mouth Is: Dataset and Analysis of Real World Habit Building Attempts”. Proceedings of the International AAAI Conference on Web and Social Media 18 (1):1967-78. DOI: https://doi.org/10.1609/icwsm.v18i1.31440.

2. Habitica – Habitica Team [Електронний ресурс]. URL: https://habitica.com/static/home (дата звернення: 08.05.2025).

3. Any.do – Any.do To-do list & Calendar [Електронний ресурс]. URL: https://www.any.do/ (дата звернення: 08.05.2025).

4. Coach.me – Coach Kendra, Inc. [Електронний ресурс]. URL: https://coach.me/habit-tracker (дата звернення: 08.05.2025).

5. Loop - Habit Tracker – Álinson S Xavier [Електронний ресурс]. URL: https://loop-habit-tracker.vercel.app/ (дата звернення: 08.05.2025).

6. Respawn – Respawn Team [Електронний ресурс]. URL: https://respawn.pro/ (дата звернення: 08.05.2025).

7. MVVM Architecture in Android using Kotlin: A Practical Guide - Jaykishan Sewak [Електронний ресурс]. URL: https://medium.com/@jecky999/mvvm-architecture-in-android-using-kotlin-a-practical-guide-73f8de1d9c58 (дата звернення: 0.05.2025).

8. Посилання на власний GitHub репозиторій [Електронний ресурс]. URL: https://github.com/NurePopovaYelyzaveta/2025_B_PI_PZPI-21-1_Popova_Y_R