МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

КОРИЦЬКИЙ ЕДУАРД РУСЛАНОВИЧ

РОЗРОБКА ТА ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ «ELEGIX» ДЛЯ ОРГАНІЗАЦІЇ ПЛАНУ ЖИТТЯ

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
МАМЧИЧ ТЕТЯНА ІВАНІВНА,
доцент кафедри комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп’ютерних наук
та кібербезпеки від _____ 2025 р.
Завідувач кафедри (_____) Гришанович Т. О.

ЛУЦЬК - 2025

ЗМІСТ

ВСТУП

У сучасному світі, що стрімко змінюється під впливом цифрових технологій, дедалі актуальнішою стає потреба в особистій ефективності, чіткому плануванні та свідомому управлінні власним життям. Інформаційне перевантаження, відсутність балансу між особистим та професійним, а також швидкий ритм життя призводять до втрати фокусу, прокрастинації та хронічної нестачі мотивації. У відповідь на ці виклики зростає інтерес до цифрових інструментів саморозвитку, серед яких особливе місце займають мобільні застосунки.

На сьогодні ринок мобільних застосунків пропонує велику кількість рішень, пов’язаних із плануванням задач, постановкою цілей, відстеженням звичок та ментальним добробутом. Проте більшість із них або занадто загальні, або не враховують цілісного підходу до життя користувача, зокрема його цінностей, цілей у різних сферах, особистих пріоритетів і джерел мотивації.

У зв’язку з цим особливої актуальності набуває створення інноваційного мобільного застосунку “Elegix”, який допомагає користувачам проектувати своє життя, базуючись на п’яти ключових сферах (здоров’я, стосунки, кар’єра, саморозвиток, духовність) та концепції ікігай. Цей застосунок не лише слугує органайзером чи трекером, а є цілісною системою для досягнення гармонії, цілеспрямованості та внутрішньої дисципліни.

Метою даної роботи є розробка та проєктування мобільного застосунку “Elegix” для організації плану життя користувача з урахуванням принципів UX/UI-дизайну, психології цілей і сучасних технологій розробки програмного забезпечення.

Для досягнення поставленої мети необхідно вирішити наступні завдання**:**

- проаналізувати існуючі рішення в сфері саморозвитку та планування життя;
- сформувати функціональні та нефункціональні вимоги до застосунку;

- розробити архітектуру та дизайн інтерфейсу відповідно до принципів зручності та мотиваційної ефективності;
- реалізувати базовий функціонал mvp-застосунку за допомогою сучасних мобільних технологій;
- провести тестування та оцінити користувацький досвід використання продукту.

Об’єкт дослідження – цифрові інструменти для планування життя та саморозвитку.

Предмет дослідження – процес розробки та проєктування мобільного застосунку на базі підходу UX/UI-дизайну, сучасних технологій розробки та принципів особистісної ефективності.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ЗАСАДИ ТА ТЕХНОЛОГІЧНІ ОСНОВИ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ ОРГАНІЗАЦІЇ ЖИТТЯ

1.1 Поняття мобільного застосунку та його характерні особливості

У сучасному світі, де інформаційне перевантаження та високий темп життя стають нормою, особисте управління часом, звичками й цілями набуває критичного значення. У цьому контексті мобільні застосунки для організації життя стають невід'ємними цифровими помічниками, що дозволяють людям структурувати повсякденну діяльність, розвивати дисципліну, підтримувати баланс між сферами життя та досягати поставлених цілей.

Мобільний застосунок [1] – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних застосунків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн-магазинів мобільних застосунків, таких як App Store [2], Google Play [3] та інших, безкоштовно або за плату.

Сучасні мобільні застосунки мають низку характеристик, що зумовлюють їхню ефективність, масштабованість та привабливість для розробників

Стабільна робота: застосунок має працювати без збоїв, фризів чи втрати даних.

Висока швидкодія: мінімальні затримки при взаємодії з інтерфейсом, швидке завантаження контенту.

Повнота функціоналу: усі заявлені функції мають бути доступні, логічно згруповані та протестовані.

Підтримка основних ОС: Android [4] та iOS [5], з урахуванням особливостей кожної платформи.

Уніфікований досвід: інтерфейс може адаптуватися, але має зберігати основну логіку та структуру.

Офлайн-режим: важливі функції мають бути доступні без інтернету (якщо це логічно для типу застосунку).

Кешування даних: збереження попередньо завантаженої інформації для швидкого доступу.

API-зв’язки [6]: синхронізація з хмарними сховищами, сторонніми сервісами (Google, Apple, Facebook, тощо).

Push-сповіщення [7]: своєчасне інформування користувача з можливістю персоналізації.

Гнучкість у розвитку: можливість додавання нових функцій без суттєвих змін в архітектурі.

Зворотний зв’язок: механізми збору відгуків і помилок від користувачів для подальшого покращення.

Успішний мобільний застосунок має не лише вирішувати конкретну задачу користувача, а й бути зручним, швидким, безпечним, адаптивним до платформ та відкритим до подальшого масштабування. Саме відповідність цим загальним вимогам забезпечує довіру та лояльність користувачів.

1.2. Класифікація мобільних застосунків

Мобільні застосунки стали невід'ємною частиною цифрового повсякдення, охоплюючи найрізноманітніші сфери життя – від комунікацій до здоров’я, від бізнесу до саморозвитку. Для ефективного проєктування, розробки та просування мобільних продуктів важливо розуміти їхню класифікацію за основними критеріями: призначення, архітектура, технології та характер взаємодії.

За типом основного завдання мобільні застосунки умовно поділяються на такі категорії:

- Освітні застосунки – застосунки для навчання мов, підготовки до іспитів, курсів тощо (Duolingo, Coursera);

- застосунки для здоров’я та саморозвитку – трекери звичок, медитацій, харчування, фітнесу, ментального добробуту (Headspace, Habitica, MyFitnessPal);
- соціальні мережі та комунікаційні застосунки – месенджери, соцмережі, відеозустрічі (Instagram, Telegram, Zoom);
- фінансові застосунки – банківські системи, інвестиційні платформи, криптогаманці (Revolut, Monobank);
- розважальні застосунки – ігри, музика, відео, стрімінгові сервіси (Spotify, Netflix);
- комерційні застосунки – маркетплейси, мобільний шопінг, бронювання (Amazon, Glovo).

Мобільні застосунки можуть мати різну архітектурну побудову.

Нативні застосунки (Рис 1.1) [8] – розроблені окремо для iOS або Android з використанням Swift/Obj-C чи Kotlin/Java. Дають максимальну продуктивність і доступ до функцій пристрою.

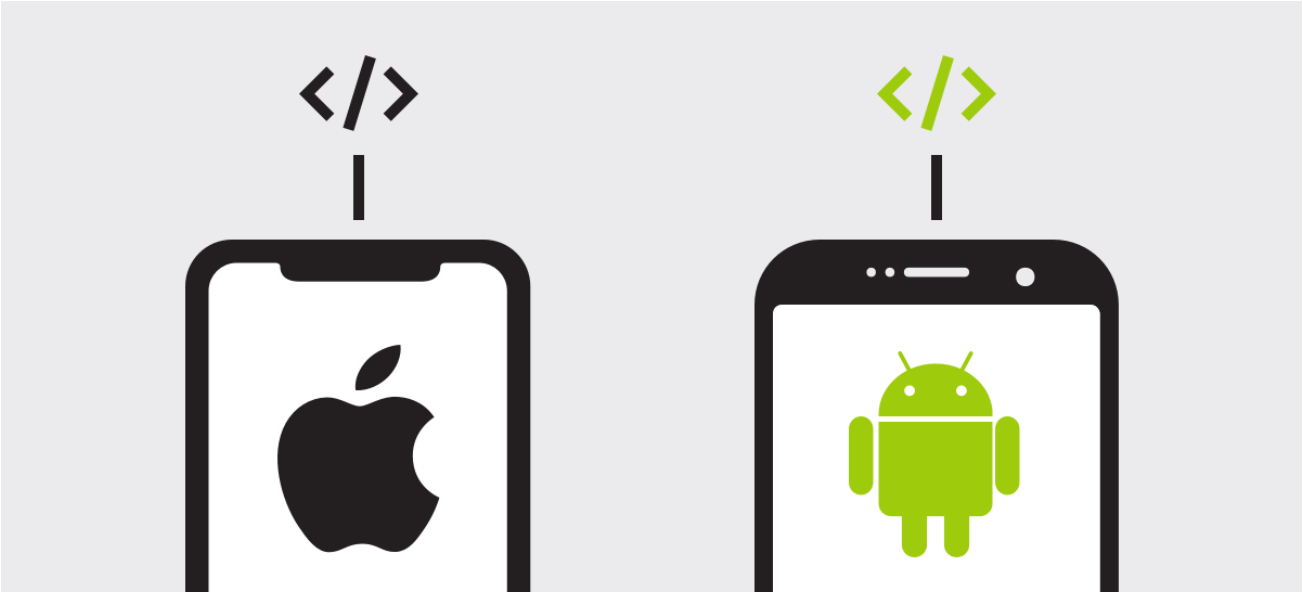

Рисунок 1.1 – Нативні мобільні застосунки

Кросплатформені застосунки (Рис 1.2) – створюються на одній базі коду (React Native [9], Flutter), що дозволяє скоротити час і бюджет розробки.

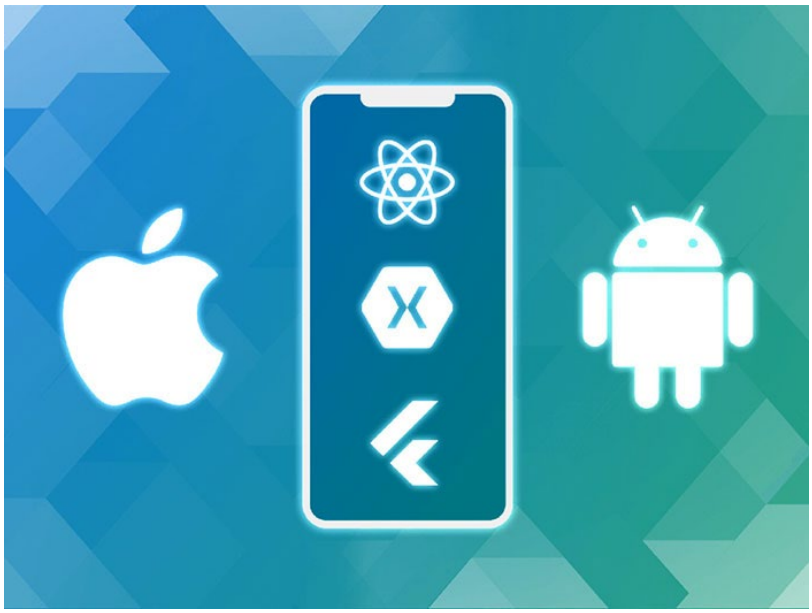

Рисунок 1.2 – Кросплатформені мобільні застосунки

Гібридні застосунки – поєднують вебтехнології та доступ до API пристрою через оболонку (Ionic, Cordova). Менш продуктивні, але зручні в розробці.

Отже, Класифікація мобільних застосунків дає змогу краще розуміти логіку їхнього проєктування, враховувати потреби аудиторії, особливості платформи, вибирати правильну архітектуру та технології. Це критично важливо як на етапі планування MVP [10], так і під час масштабування продукту.

1.3 Архітектура та компоненти мобільного застосунку для організації життя

Сучасний мобільний застосунок для особистісного зростання та організації життя є потужним інструментом цифрової трансформації повсякдення, який дозволяє користувачеві структурувати свої цілі, звички, сферу відповідальності та досягнення. Ефективна робота такого застосунку базується на продуманій архітектурі, що поєднує декілька ключових функціональних компонентів, які взаємодіють між собою та утворюють єдину екосистему продуктивності.

Мобільний застосунок реалізується за клієнт-серверною архітектурною моделлю, де інтерфейс користувача працює на мобільному пристрої, а бізнес-логіка, обробка запитів і збереження даних – на серверній стороні. Такий підхід дозволяє централізовано обробляти інформацію, підтримувати стабільність роботи системи, реалізовувати резервне копіювання та масштабувати функціонал.

Типова архітектура мобільного застосунку для саморозвитку включає три ключові шари.

Клієнтський рівень (інтерфейс користувача) – інтерфейс користувача (UI) [11] є ключовим елементом взаємодії з системою, що забезпечує зручність, естетику та мотивацію.

Основні функції клієнтського інтерфейсу включають:

- візуалізацію особистих цілей, прогресу, звичок і щоденних завдань;
- навігацію між п’ятьма життєвими сферами (здоров’я, стосунки, кар’єра, розвиток, духовність);
- інтеграцію з системами нагадувань, аналітики та персональних досягнень;
- адаптивний дизайн для смартфонів із підтримкою темної та світлої тем.

Серверна логіка (Backend) – на цьому рівні реалізовано обробку запитів користувача, персоналізовану логіку планування, збереження динаміки виконання цілей, генерацію рекомендацій на основі відповіді на стартове опитування та поведінки в застосунку.

Фреймворки, що можуть бути використані для реалізації:

- Node.js / C# / Django – для обробки;
- Firebase Functions – для швидкої реалізації динамічних сценаріїв.

Система збереження даних (Database Layer) - цей компонент відповідає за надійне зберігання даних користувача: його сфер життя, цілей, звичок, результатів, аналітики й історії активності.

Основні характеристики:

- структуроване збереження даних (наприклад, за допомогою Firestore, PostgreSQL);

- реалізація механізмів шифрування даних і бекапу;
- оптимізація запитів для швидкого доступу навіть на повільних пристроях.

Модуль цілей і звичок. Користувачі можуть створювати цілі, ділити їх на підзадачі, додавати звички, які автоматично враховуються в прогресі. Важливо, що один шаблон звички може бути прив'язаний до кількох цілей одночасно.

Аналітика та звітність – система візуалізує прогрес у кожній сфері життя, показує збережені шаблони, streak-и, рівень балансу між сферами, а також автоматично видає рекомендації для поліпшення.

Система автентифікації та безпеки – передбачено авторизацію через email/соцмережі, збереження даних із шифруванням, а також резервне копіювання у хмарі.

Інтеграція зі сторонніми сервісами - API-інтеграції дозволяють підключати календарі, нагадування, Google Fit/Apple Health, а в майбутньому – GPT-інтерфейси для персональних порад і коучингу. сприяючи формуванню єдиного цифрового освітнього простору.

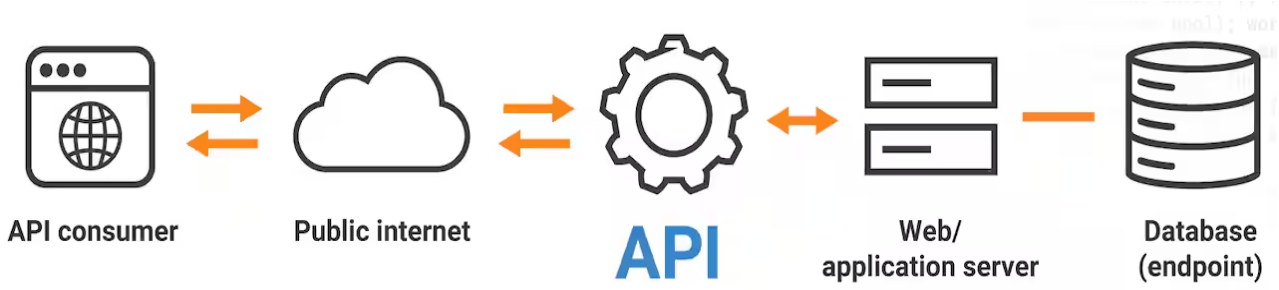

Рисунок 1.3 – Робота API

1.4. Платформи для організації життя, їх можливості та обмеження

У цифрову епоху, на тлі зростаючої складності життя та постійних викликів, люди дедалі частіше звертаються до технологій для покращення самоорганізації. Платформи для організації життя – це не просто інструменти, а

екосистеми, які охоплюють різні сфери: цілі, звички, час, навчання, роботу та особистий розвиток.

1.4.1. Персоналізація завдяки технологіям штучного інтелекту

Сучасні платформи, зокрема мобільні застосунки й вебсервіси, дедалі частіше інтегрують штучний інтелект, що дозволяє глибше аналізувати поведінку користувача. На основі його цілей, стилю мислення, темпу та активності – ШІ формує персональні плани, надсилає рекомендації, попереджає про прокрастинацію, а також адаптує контент для кращого досягнення результатів.

У майбутньому очікується поява повністю індивідуалізованих помічників, які підлаштовуватимуть усю цифрову рутину під потреби конкретної особистості.

1.4.2. Життя як безперервний розвиток (Lifelong Self-Management)

Концепція "життя як проєкту" поширюється серед амбітних людей. Платформи на кшталт Notion, Todoist, Trello, Sunsama, Habitica, Motion, а також спеціалізовані трекери цілей підтримують ідею безперервного самовдосконалення. Вони дають змогу структурувати інформацію, контролювати прогрес, підвищувати усвідомленість і залишатися гнучкими в будь-яких життєвих обставинах..

Сучасні мобільні застосунки для організації життя стрімко розвиваються, впроваджуючи інноваційні технології, які підвищують ефективність життя. Ключовими тенденціями є інтеграція мобільних технологій та розвиток інтерфейсів з фокусом на користувача.

Ключові можливості сучасних платформ:

- Цифрове планування: календарі, списки справ, щоденники.
- Трекінг звичок: системи з візуалізацією прогресу, повторюваними діями, мотиваційними сигналами.

- Цілепокладання: розбиття великих цілей на етапи, дедлайни, контроль результатів.
- Аналітика: трекінг продуктивності, часових витрат, емоційного стану.
- Гейміфікація: бали, досягнення, рейтинги, що підвищують мотивацію (Habitica, Forest).
- AI-нагадування і підтримка: чат-боти, розумні рекомендації, автоматичні звіти.
- Комунікація та підтримка: інтеграція з менторськими чатами, підтримка спільнот або коучів.

1.5. Технологічна основа розробки мобільного застосунку

Розробка сучасного мобільного застосунку потребує використання ефективного та узгодженого технологічного стеку, який забезпечує високу продуктивність, масштабованість, безпеку, а також зручність у підтримці та подальшому розвитку продукту. Технології, що використовуються у процесі створення мобільних рішень, зазвичай охоплюють кілька ключових компонентів: фреймворки для клієнтської частини, серверні рішення, бази даних, API та інструменти керування версіями.

Клієнтська частина мобільного застосунку реалізується з використанням фреймворку React Native, який дозволяє створювати повноцінні нативні застосунки для платформ Android та iOS із єдиної кодової бази на мові програмування JavaScript або TypeScript. Це значно знижує витрати на розробку, дозволяє швидше виводити продукт на ринок та підтримувати однаковий функціонал на обох мобільних платформах.

1.5.1. React Native: переваги та архітектура

React Native побудований на основі філософії компонетно-орієнтованого підходу. Це означає, що інтерфейс користувача створюється як ієрархія

незалежних UI-компонентів, кожен з яких відповідає за конкретну частину логіки або візуального представлення.

Ключовими перевагами React Native є:

- Кросплатформеність – один код працює на Android та iOS;
- Висока продуктивність завдяки рендерінгу нативних компонентів (не веб-огортка, як у WebView);
- Гнучкість у дизайні інтерфейсів та багата екосистема бібліотек;
- Підтримка гарячого перезавантаження (Hot Reload), що дозволяє миттєво бачити зміни без перевстановлення застосунку;
- Активна спільнота та підтримка Facebook, що гарантує стабільність фреймворку та регулярні оновлення.

React Native використовує віртуальний DOM (через бібліотеку React), що дозволяє ефективно оновлювати інтерфейс при зміні стану компонентів. Це особливо важливо для мобільних застосунків, де важлива висока швидкодія і плавність.

1.5.2. Управління станом: Zustand, Redux, Context API

Для керування внутрішнім станом застосунку, особливо при роботі з великою кількістю взаємозалежних даних (наприклад, завдання, звички, статуси користувача, авторизація), застосовуються сучасні інструменти для управління станом:

- Zustand [12] – легка, реактивна бібліотека з мінімальним кодом, чудово підходить для мобільних застосунків, що потребують простоти та швидкості.
- Redux [13] – потужніший підхід з предикативним потоком даних, добре масштабується, зручно дебажити за допомогою Redux DevTools.
- React Context API – нативний механізм передачі даних між компонентами без необхідності прокидування props.

У реальному проєкті можливе комбіноване використання Zustand для локального стану (UI, форма), а Redux – для глобального стану (автентифікація, прогрес користувача).

1.5.3. Expo SDK як інструмент для швидкого запуску

Для прискорення початкової розробки та забезпечення швидкої інтеграції з нативними можливостями пристрою використовується Expo SDK – набір бібліотек та інструментів, побудований поверх React Native [14]. Expo спрощує роботу з:

- датчиками пристрою (акселерометр, гіроскоп, GPS, компас);
- камерою та мультимедіа;
- Push-сповіщеннями (через Expo Notifications API);
- біометричною автентифікацією (FaceID, TouchID);
- файловою системою;
- Splash-екраном, іконками, оновленнями по повітрю (OTA).

Expo дозволяє тестувати застосунок без компіляції – достатньо сканувати QR-код у мобільному додатку Expo Go, що значно економить час на початкових етапах.

1.5.4. Структура та архітектура проєкту

Застосунок реалізується за принципами чистої архітектури (Clean Architecture) [15], де код розділений на логічні шари:

- UI – інтерфейсні компоненти, стилі, анімації;
- State management – Zustand/Redux/Context для керування даними;
- Services – логіка роботи з API, базою даних, авторизацією;
- Navigation – реалізована через react-navigation, з підтримкою стеків, вкладок і модалок;
- Utils/hooks – загальні утиліти, хелпери, кастомні хуки.

Це забезпечує масштабованість, зручність у розробці нових функцій та зрозумілість структури проєкту для інших учасників команди.

Серверна частина та API. У межах розробки мобільного застосунку було прийнято рішення використовувати Firebase як основну платформу для реалізації серверної частини. Firebase – це хмарна платформа від компанії Google, яка надає широкий спектр сервісів для створення, розгортання та масштабування мобільних і вебзастосунків. Її використання дозволяє значно скоротити час на розробку серверної інфраструктури, забезпечити гнучке масштабування та реалізувати функціонал бекенду без потреби у створенні окремого сервера.

До основних компонентів Firebase, що використовуються у застосунку, належать:

- Firebase Authentication – сервіс, який забезпечує автентифікацію користувачів за допомогою електронної пошти, пароля, OAuth-провайдерів (Google, Apple, Facebook тощо). Він спрощує процес авторизації, а також підтримує механізми підтвердження електронної пошти, відновлення пароля та багатофакторну автентифікацію.
- Cloud Firestore – документно-орієнтована база даних, яка дозволяє зберігати структуровані дані, синхронізувати їх у режимі реального часу та забезпечує гнучкі правила доступу на рівні окремих документів.
- Cloud Functions for Firebase – серверні функції, які дозволяють реалізовувати бізнес-логіку на сервері. Вони можуть викликатися за подіями у базі даних, автентифікації, HTTP-запитах тощо. Через цей сервіс реалізовано логіку REST API та взаємодію з клієнтською частиною.
- Firebase Cloud Messaging (FCM) – механізм відправлення push-сповіщень користувачам, що використовується для надсилання нагадувань про завдання, звички, досягнення цілей та інші сповіщення.
- Firebase Storage – хмарне сховище для збереження зображень профілю користувача, медіа-ресурсів та інших файлів.

Використання цих сервісів дозволяє реалізувати повноцінну архітектуру бекенду без розгортання фізичного або віртуального сервера, що позитивно впливає на час розробки, стабільність роботи та простоту обслуговування системи.

Реалізація API. Обмін даними між клієнтською частиною (застосунок, розроблений на React Native з використанням Expo) та сервером здійснюється через REST API, створений за допомогою Cloud Functions. Такий підхід забезпечує:

- Гнучку реалізацію логіки: логіка доступу, обробки даних і відповіді на запити винесена у серверні функції, що викликаються через HTTP.
- Безпеку: кожна функція перевіряє автентифікацію користувача через Firebase Authentication.
- Контроль доступу: реалізовано перевірки прав доступу на основі ролей користувача (наприклад, звичайний користувач, ментор, адміністратор).

У перспективі можливе впровадження GraphQL API [16] для оптимізації запитів та зменшення обсягу переданих даних.

Для керування розробкою застосунку використовується система контролю версій Git, яка є стандартом у сучасному програмному забезпеченні. Git забезпечує:

- Збереження історії змін: можливість відстежувати, коли й ким були внесені зміни до коду.
- Паралельну роботу команди: завдяки гілкуванню (branching) кожен розробник може працювати над окремими частинами проєкту незалежно.
- Можливість відновлення попереднього стану: у разі помилок або збоїв можна швидко повернутися до стабільної версії проєкту.

У рамках проєкту використовується репозиторій на платформі GitHub, а також впроваджено елементи CI/CD (Continuous Integration / Continuous Deployment) [17] для автоматизації тестування та збирання застосунку. Зокрема, за допомогою GitHub Actions та Expo Application Services (EAS) реалізовано:

- Автоматичну перевірку коду після кожного коміту.
- Генерацію збірок для Android та iOS.
- Підготовку застосунку до публікації у відповідні магазини додатків.

1.6. Дослідження аналогів мобільних застосунків для організації життя

У сучасному цифровому середовищі представлено велику кількість мобільних застосунків, що спрямовані на допомогу користувачам в організації особистого життя, підвищенні продуктивності, досягненні цілей та розвитку корисних звичок. Ці рішення умовно можна класифікувати на кілька категорій: трекери звичок, планувальники справ, додатки для постановки цілей та саморозвитку. Попри широкий вибір, значна частина таких продуктів має вузький функціонал або занадто загальне позиціонування, що ускладнює створення цілісної системи особистісного розвитку. Деякі з них фокусуються лише на задачах або звичках, інші – на мотиваційних інструментах або щоденниках, рідко поєднуючи всі ці аспекти в одному рішенні.

Аналіз існуючих застосунків дозволяє виявити найкращі UX-практики, ключові функціональні модулі, очікування користувачів, а також недоліки, що залишають можливість для створення конкурентної переваги. Для аналізу було обрано п’ять популярних застосунків: Habitica, Fabulous, Notion, Finch та Todist, які по-різному підходять до теми особистої ефективності й організації життя (Таблиця 1).

Таблиця 1

Порівняльна характеристика платформ для навчання

Платформа	Основні характеристики	Переваги	Обмеження
Habitica	Гейміфікований трекер звичок і завдань	Мотивація через ігрову механіку, командні челенджі, візуальний стиль	Складний для новачків, низька гнучкість під потреби
Fabulous	Мотиваційний коуч-додаток з фокусом на здорові звички	Психологічна підтримка, гарний	Обмежений контроль над

		дизайн, науковий підхід	цілями, закритий функціонал
Notion	Універсальний органайзер з гнучкими базами даних	Висока кастомізація, шаблони, інтеграції	Складність для нових користувачів, відсутність гейміфікації
Finch	Додаток для емоційного добробуту у форматі віртуального помічника	Турботливий UX, мікрозадачі, саморефлексія	Обмежений набір функцій щодо планування і звичок
Todoist	Популярний таск-менеджер для особистих і робочих завдань	Простий інтерфейс, потужна система задач, теги та пріоритети	Немає фокусу на звичках чи саморозвитку

Таким чином, більшість розглянутих застосунків акцентують увагу на окремих аспектах особистої ефективності – як-от завдання, звички або ментальне здоров’я, проте на ринку відсутній цілісний застосунок, який би об’єднував ці компоненти в системну модель саморозвитку, орієнтовану на досягнення цілей у ключових сферах життя. Це відкриває нішу для створення рішення, що інтегрує функціонал трекінгу звичок, планування цілей, ментального супроводу та гейміфікації в одному інтерфейсі.

Habitica [18] – це додаток для трекінгу звичок, який перетворює повсякденні завдання на елементи рольової гри (RPG). Ідея полягає в тому, що користувач створює персонажа і виконує реальні дії, як-от чистити зуби, займатися спортом чи писати код, щоб заробляти віртуальні винагороди: досвід, золото, предмети. Якщо користувач не виконує поставлені звички чи задачі – він втрачає життя, тобто система також включає покарання.

Основна фішка Habitica – це гейміфікація. Інтерфейс оформлено у стилі 8-бітних ігор: користувач має піксельного аватара, який отримує спорядження, броню, зброю. Коли людина досягає прогресу у реальному житті, її персонаж прокачується. Це створює ілюзію гри, де прогрес прив'язаний до реальних дій.

В додатку є три основні типи дій: звички (які можуть бути позитивні або негативні), щоденні задачі (які потрібно виконувати регулярно) і одноразові задачі (разові to-do). Наприклад, звичка «пити воду» може додаватися як позитивна; за кожен раз, коли ти натискаєш, що зробив її – отримуєш XP. Щоденні задачі обнуляються щодня. Якщо не виконав – втрачаєш життя. Це створює постійну потребу повертатись до апки.

Також у Habitica можна вступати в гільдії, брати участь у групових челенджах або битвах з монстрами. Якщо твоя команда не виконує задачі – весь загін втрачає HP. Це стимулює соціальну відповідальність і підсилює мотивацію. Є віртуальні магазини, де можна купувати косметичні предмети чи апгрейди для аватара, витрачаючи золото, зароблене виконанням завдань.

Однак інтерфейс може здатися перевантаженим для новачка. Через акцент на гейміфікації перший досвід може бути складним. Є безліч кнопок, меню, статусів, налаштувань. Для повного розуміння механіки треба витратити час. Також Habitica не дуже підходить для детального планування великих проектів або цілей. Це більше про щоденні маленькі дії, а не про стратегічне мислення. Гнучкість функціоналу обмежена: наприклад, не можна робити вкладені підзадачі або будувати складні ланцюжки планування.

Цільова аудиторія – це переважно молоді люди, які люблять ігри або потребують зовнішньої мотивації. Особливо добре працює для тих, кому важко мотивувати себе стандартними списками справ. Якщо користувач любить проходити квести, збирати предмети чи отримувати досягнення – Habitica викликає емоційне залучення. Але для серйозного трекінгу цілей або роботи в бізнесі – вона часто здається дитячою іграшкою.

У підсумку, Habitica – це незвичайний інструмент, який чудово працює для людей, яким важко підтримувати самодисципліну традиційними методами. Вона

створює почуття досягнення, прогресу та відповідальності, але водночас потребує часу на адаптацію і не дає гнучких можливостей для великого планування.

Notion – це багатофункціональний інструмент для організації інформації, який поєднує в собі функції нотатника, менеджера завдань, бази знань, редактора документів і трекера проектів. Його головна особливість – гіпергнучкість, яка дозволяє користувачам самостійно будувати інтерфейс під свої потреби. На відміну від класичних застосунків типу Todoist чи Trello, Notion не нав'язує структуру – він дає порожній простір, в якому користувач сам створює системи організації: від простих списків завдань до повноцінних CRM або wiki-баз.

Користувацький досвід у Notion починається з так званих «сторінок», які можна наповнювати будь-чим: текстами, зображеннями, таблицями, базами даних, чеклістами, кнопками, таймлайнами, календарями та навіть вбудованими інтеграціями. Весь інтерфейс побудований за принципом блоків: кожен елемент – це блок, який можна вільно переміщувати, вкладати один в інший, клонувати або пов’язувати між собою.

Notion активно використовують як індивідуальні користувачі для особистого планування, так і команди – для ведення документації, управління проєктами, координації завдань або створення внутрішніх порталів.

Одна з найсильніших сторін Notion – бази даних, які можуть відображатись у різних форматах: таблиці, дошки, галереї, списки, календарі, таймлайни.

Також важливою особливістю є інтеграція з AI, яка дозволяє автоматизовувати рутинні дії: генерувати тексти, підсумки зустрічей, структурувати хаотичні нотатки, вигадувати заголовки, писати email-відповіді, планувати завдання або навіть придумувати ідеї. ШІ інтегровано безпосередньо в блоки, тому його можна викликати в будь-який момент.

Інтерфейс Notion є водночас простим і складним. З одного боку, він інтуїтивний, з мінімалістичним дизайном, який не відволікає. З іншого боку – новачки часто губляться в його безмежності, адже немає чітких правил, як саме слід організовувати інформацію. Для ефективної роботи потрібно інвестувати

час у побудову власної системи або скористатись шаблонами, які створює спільнота.

Notion не має класичної системи нагадувань, що характерно для трекерів звичок. Це більше про контекстну організацію життя, ніж про щоденні пуші. Тому для звичних форматів “сформуй звичку, отримай нагороду” – він не підходить. Але якщо людина хоче бачити всі свої проекти, цілі, завдання, контент і знання в єдиній системі – Notion ідеально виконує цю функцію.

У результаті, Notion – це не просто апка, а ціла філософія організації життя. Вона не диктує користувачу, як треба працювати, а надає інструменти, щоб побудувати свою власну систему – гнучку, візуальну і розширювану. Але вона вимагає системного мислення, самодисципліни та готовності витратити час на її налаштування.

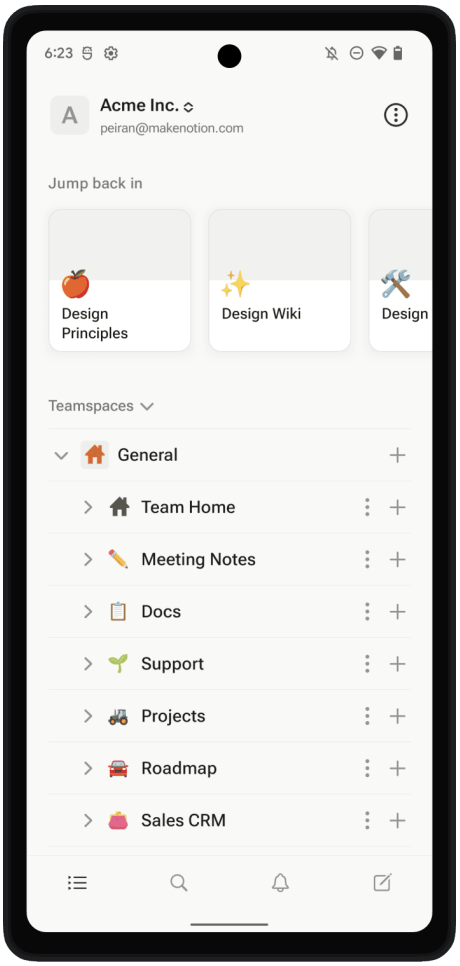

Рисунок 1.4 – Платформа Notion

Todoist – це багатоплатформний застосунок для управління завданнями (task manager), який було запущено компанією Doist у 2007 році. Протягом останніх півтора десятка років продукт утримує позицію одного з найпопулярніших інструментів у своєму класі, зокрема серед фахівців, які дотримуються методології Getting Things Done (GTD), фокусуються на дисциплінованому підході до планування та потребують щоденної структурованості в особистих або робочих задачах.

Todoist доступний на всіх основних операційних системах (iOS, Android, macOS, Windows, Web) та має розширення для браузерів, інтеграції з календарями й месенджерами, що дозволяє користувачу створити єдине цифрове середовище для управління часом і завданнями.

Todoist має мінімалістичний, інтуїтивно зрозумілий інтерфейс, який базується на принципі «інбоксу» та контекстного сортування завдань. Основна логіка роботи з додатком зводиться до створення задач (tasks), які можна:

- Групувати в проєкти (projects).
- Позначати мітками (labels).
- Фільтрувати через кастомні умови (filters).
- Пріоритезувати за допомогою системи P1–P4.

Суттєвим елементом є використання “natural language input”, що дозволяє створювати завдання швидко, прямо в текстовому полі, наприклад: Зателефонувати Олександру завтра о 16:00 @дзвінки #робота p1.

Todoist підтримує також підзавдання (subtasks), вкладену структуру задач, нагадування (для платних користувачів) і коментарі до завдань – що відкриває можливості для колаборації в команді.

Основна логіка взаємодії з додатком побудована навколо циклу “захоплення – організація – виконання”:

- Захоплення (Capture): користувач швидко фіксує думки або завдання без категоризації.
- Організація (Organize): сортує задачі за проєктами, пріоритетами, мітками або датами.

- Фокус (Focus): працює зі списком “Сьогодні” (Today) або за кастомним фільтром (наприклад, “P1 AND @висока_енергія”).

У Todoist немає надлишкового функціоналу або складної кривої навчання – саме тому продукт високо цінується тими, хто шукає швидку, стабільну та неінвазивну систему планування.

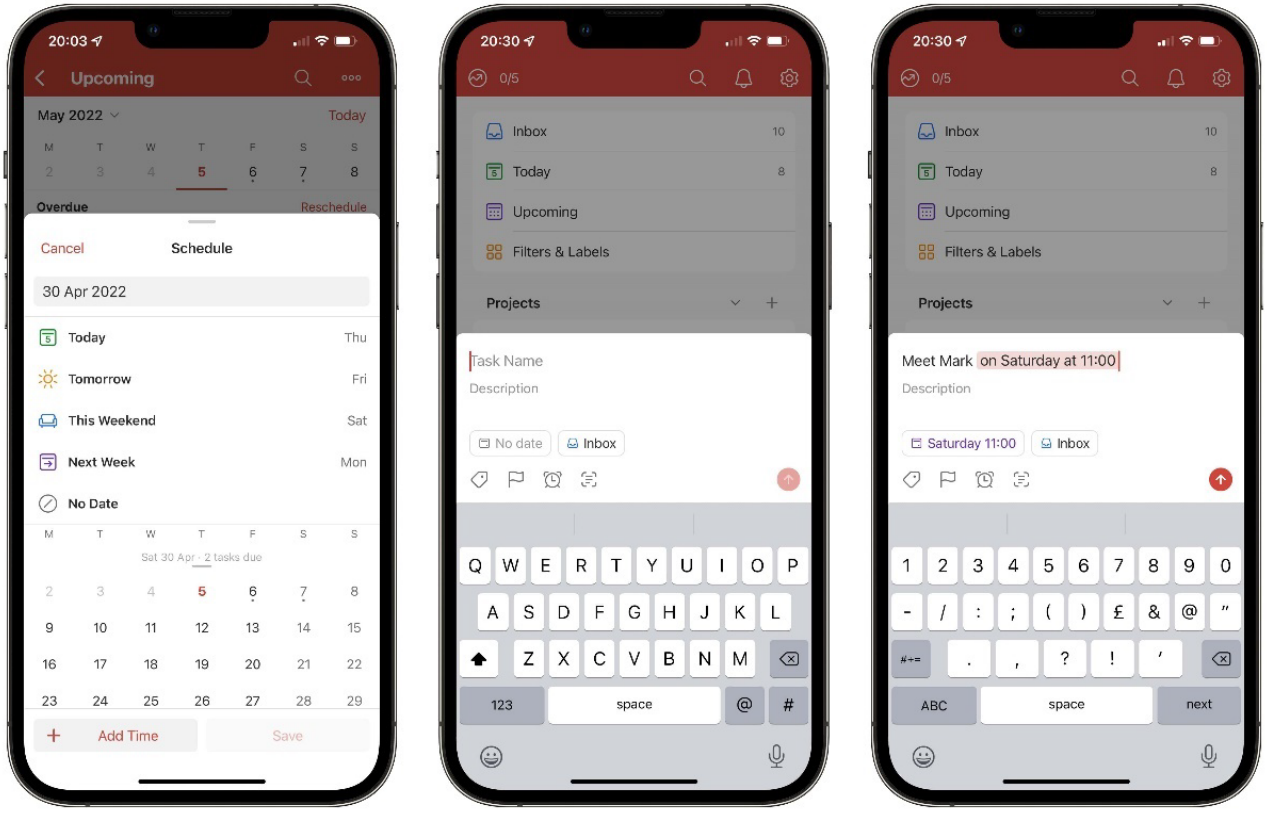

Рисунок 1.5 – Платформа TodoIst

РОЗДІЛ 2.
РОЗРОБКА ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ЖИТТЯ

2.1. Постановка задачі та визначення вимог до мобільного застосунку для організації життя

Метою даної роботи є розробка мобільного застосунку "Elegix Organizer", який допоможе користувачам підвищити продуктивність, покращити особисту організацію життя, ставити цілі та досягати їх у різних сферах. На основі аналізу предметної області, потреб цільової аудиторії та існуючих аналогів сформульовано наступні ключові вимоги до програмного продукту:

- застосунок повинен надавати можливість реєстрації нових користувачів через електронну пошту або соціальні мережі для створення персоналізованого профілю, що забезпечить індивідуальний підхід і збереження даних користувача;
- користувач повинен мати можливість редагувати особисту інформацію (ім'я, прізвище, налаштування профілю) та змінювати пароль, що гарантуватиме комфорт та безпеку використання застосунку;
- застосунок має підтримувати функціонал створення, редагування і структуризації особистих цілей та завдань у межах п’яти основних сфер життя (здоров’я, кар’єра, особистий розвиток, відносини, духовність), а також можливість вибору 3 сфер для фокусування;
- користувач повинен мати інструменти для побудови детального плану досягнення цілей з розбиттям на завдання та підзадачі, включаючи повторювані звички та трекінг їх виконання;
- передбачено інструмент для створення та ведення щоденних чи тижневих планів із функцією відкладення, архівації і повторного використання шаблонів завдань і цілей;

- застосунок має підтримувати систему нагадувань і сповіщень, що допоможуть користувачу залишатися сфокусованим на пріоритетах;
- для підтримки мотивації передбачено візуалізацію прогресу, зведені звіти та інші аналітичні інструменти;
- застосунок повинен мати зрозумілий та адаптивний інтерфейс, з урахуванням темної та світлої тем, що забезпечить комфортне використання в різних умовах;
- забезпечення безпеки даних користувачів має здійснюватись за допомогою SSL-шифрування та надійної системи автентифікації.

На основі цих вимог визначено такі принципи взаємодії користувача із застосунком:

- для повноцінного використання функціоналу кожен користувач повинен пройти процедуру реєстрації або увійти через існуючий акаунт;
- після входу користувач отримує доступ до персонального кабінету, де може створювати, редагувати та відстежувати свої цілі, завдання і звички;
- користувач має можливість обирати сфери життя для фокусування і коригувати їх відповідно до своїх пріоритетів;
- під час створення цілей застосунок допомагає структурувати їх у вигляді плану з детальним розподілом на завдання та підзадачі, з урахуванням повторюваних дій (звичок);
- всі зміни автоматично зберігаються, а інформація про виконання доступна у вигляді візуальних дашбордів та звітів;
- застосунок підтримує регулярні нагадування, що сприяють підтриманню дисципліни та продуктивності.

Дотримання вказаних вимог сприятиме створенню зручного, ефективного інструменту для самоменеджменту, що допоможе користувачам підвищити якість життя та досягати поставлених цілей у різних сферах.

2.2. Вибір моделі розробки програмного засобу

Для розробки мобільного застосунку "Elegix Organizer" було обрано ітераційну модель розробки. Ця модель передбачає поділ процесу створення застосунку на серію коротких циклів (ітерацій), кожен з яких включає етапи планування, аналізу, проєктування, реалізації та тестування.

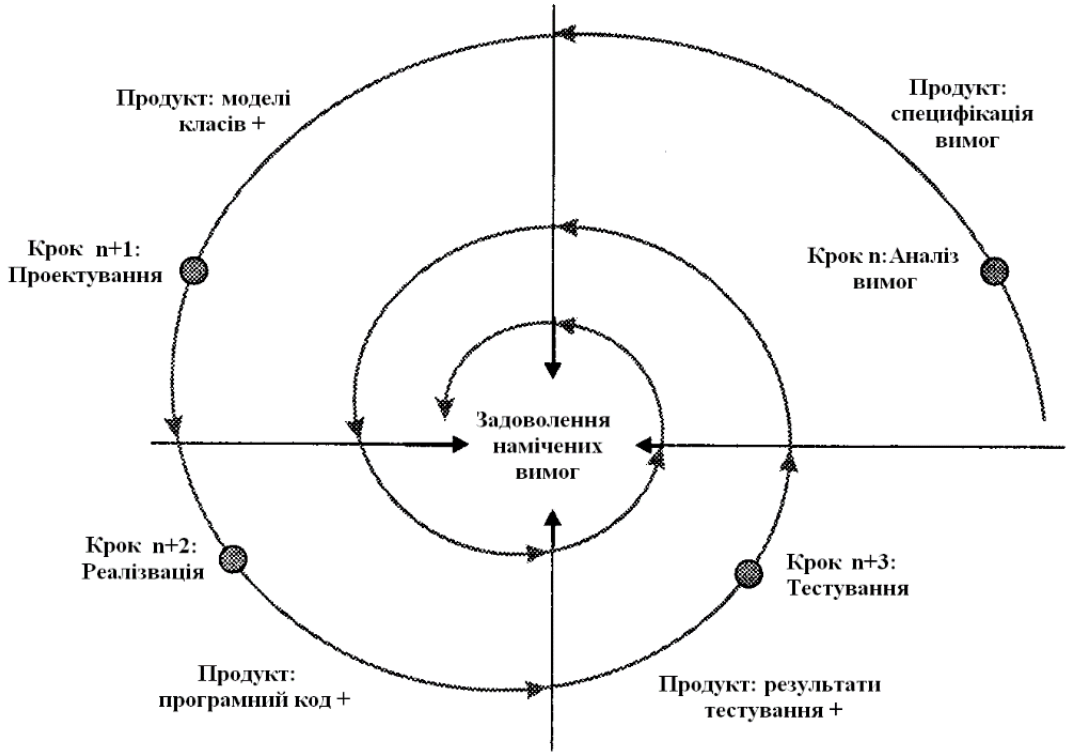

Рис 2.1 – Ітеративна модель

Головною перевагою такого підходу є можливість поступового нарощування функціоналу, а також гнучке реагування на зміни вимог або виявлені проблеми протягом розробки.

Ключова ідея ітераційної моделі полягає у заміні тривалого одноразового процесу розробки низкою менших циклів, що дає змогу не лише покращувати окремі модулі застосунку, а й оперативно інтегрувати нові функції та виправлення у вже готові компоненти.

Основні етапи, які повторюються в кожній ітерації, включають:

- аналіз вимог та планування наступного функціоналу;
- проєктування архітектури системи та вибір технологічного стеку;
- реалізація нових або вдосконалення існуючих функцій;
- тестування, перевірка якості та усунення помилок;
- оцінка результатів і збір зворотного зв’язку для корекції наступних кроків.

Вибір ітераційної моделі розробки для "Elegix Organizer" забезпечує ефективне управління ризиками, дозволяє швидко реагувати на потреби користувачів та гарантує високу якість кінцевого продукту. Кожен цикл завершується релізом робочої версії застосунку з оновленим функціоналом, що дає змогу своєчасно отримувати відгуки та вносити необхідні зміни в процесі подальшої розробки.

2.3. Загальний опис проекту

Для реалізації мобільного застосунку "Elegix Organizer" було обрано архітектурний підхід Clean Architecture у поєднанні з клієнт-серверною моделлю (Client-Server). Такий вибір зумовлений необхідністю забезпечити гнучкість, масштабованість системи та полегшити її підтримку й подальший розвиток.

Clean Architecture передбачає логічний поділ системи на концентричні шари, кожен із яких має власну зону відповідальності та чітко визначені залежності. У контексті нашого застосунку це означає:

- зовнішній шар включає конкретні технології та інструменти, такі як React Native для розробки мобільного інтерфейсу, бекенд на C#, а також систему зберігання даних (наприклад, Firebase чи інші бази даних).
- шар адаптерів інтерфейсів (Interface Adapters) відповідає за трансформацію даних між зовнішніми інтерфейсами користувача та внутрішньою логікою застосунку (тут розташовуються компоненти, які обробляють запити від користувача, взаємодію з локальним сховищем або сервером);

- шар варіантів використання (Use Cases / Application Business Rules) реалізує основну бізнес-логіку застосунку, керуючи процесами постановки цілей, планування дій, відстеження звичок та іншими функціями;
- внутрішній шар містить ключові бізнес-об'єкти (сутності), такі як цілі, завдання, звички, які є незалежними від конкретних технологій та реалізацій.

Для роботи з даними на стороні сервера або локального сховища застосовується патерн "Репозиторій" (Repository Pattern), що створює абстракцію між бізнес-логікою та механізмами зберігання даних. Це дозволяє ізолювати код від конкретної реалізації зберігання (наприклад, Firebase, SQLite чи API сервер) та спрощує тестування й заміну джерела даних.

Загалом, вибір Clean Architecture, клієнт-серверної моделі та патерну Repository забезпечує надійність, гнучкість і масштабованість мобільного застосунку "Elegix Organizer". Це дозволяє спростити розробку, підтримку і подальше розширення функціоналу для комфортної організації життя користувачів.

Загалом, вибір Clean Architecture, Client-Server підходу та Repository Pattern забезпечує якість, надійність і гнучкість мобільного застосунку для організації-життя. Це спрощує розробку, підтримку та подальше розширення системи.

2.4. Обґрунтування вибору інструментальних засобів розробки

Для реалізації мобільного застосунку "Elegix Organizer", що включає клієнтську частину на React Native та серверну логіку на C# (.NET) [19], було обрано інтегровані середовища розробки (IDE), які забезпечують високу продуктивність, зручність і якість програмування.

Для фронтенду використовується Visual Studio Code [20] – легке, але потужне середовище, яке має повноцінну підтримку JavaScript, TypeScript і React Native. Вбудовані функції автодоповнення, відлагодження, інтеграція з

системою контролю версій Git, а також численні розширення підвищують ефективність розробки інтерфейсу мобільного застосунку.

Для серверної частини обрано Visual Studio 2022 [21] (або новішу версію), яка є найкращим інструментом для розробки на мові C# із використанням фреймворку .NET. Visual Studio пропонує глибоку інтеграцію з .NET, зручні засоби відлагодження, профілювання продуктивності, а також інструменти для роботи з базами даних і керування залежностями через NuGet. Підтримка Azure DevOps та інших CI/CD інструментів дозволяє ефективно організувати процес безперервної інтеграції та доставки.

Обране поєднання інструментів Visual Studio Code та Visual Studio забезпечує чітке розмежування клієнтської та серверної розробки, дозволяє швидко знаходити та виправляти помилки, а також підтримує масштабованість і якість коду в процесі створення мобільного застосунку.

2.4.1. Вибір C# та .NET для розробки серверної частини

Вибір мови програмування C# та фреймворку .NET для реалізації серверної частини мобільного застосунку є цілком обґрунтованим і базується на низці технічних і практичних переваг. C# – це сучасна, потужна та типізована мова програмування, яка активно підтримується корпорацією Microsoft і має розвинену екосистему для створення масштабованих, надійних і високопродуктивних серверних рішень.

Наведемо основні переваги використання C# та .NET.

Безпека та стабільність: .NET забезпечує потужні вбудовані механізми захисту від поширених вразливостей, таких як SQL-ін'єкції, Cross-Site Scripting (XSS) [22] та Cross-Site Request Forgery (CSRF).

Висока продуктивність і масштабованість: Завдяки можливостям асинхронного програмування, масштабування на рівні сервера та підтримці розподілених систем, .NET дозволяє ефективно працювати з великими навантаженнями.

Гнучкість і розширюваність: Широка бібліотека готових пакетів, підтримка middleware, а також потужні інструменти для створення RESTful API роблять розробку зручною і швидкою.

Розвинена екосистема: Інструменти для роботи з базами даних, ORM Entity Framework, вбудована підтримка тестування, CI/CD та інтеграція з хмарними платформами (Azure [23], AWS) дозволяють ефективно створювати, розгортати та підтримувати серверні додатки.

Ключовим фактором вибору C#/.NET стала їх надійність, підтримка сучасних архітектурних підходів та широкий спектр можливостей для побудови безпечних і масштабованих серверних сервісів для мобільного застосунку.

2.4.2. Використання React Native для створення клієнтської частини

Для реалізації клієнтської частини мобільного застосунку Elegix було обрано React Native – потужний фреймворк з відкритим кодом, який дозволяє створювати кросплатформені мобільні застосунки з єдиною кодовою базою для iOS та Android. Це рішення забезпечує високу швидкість розробки, спрощене обслуговування і зниження витрат на підтримку окремих платформ.

Наведемо переваги використання React Native.

Кросплатформеність: дозволяє створювати застосунок одночасно для iOS та Android, що значно скорочує час розробки та ресурси команди.

Компонентний підхід: забезпечує модульність та повторне використання UI-елементів, що підвищує підтримуваність та масштабованість застосунку.

Висока продуктивність: завдяки зв’язку з нативними компонентами через міст (bridge), React Native забезпечує продуктивність, наближену до нативної.

Hot Reload та Fast Refresh: спрощують процес розробки, дозволяючи миттєво бачити зміни без перезапуску застосунку.

Розвинена екосистема: інтеграція з популярними бібліотеками (наприклад, React Navigation, Redux, AsyncStorage) спрощує створення складних функціональностей.

Використання React Native дозволяє реалізувати мобільний застосунок Elegix із сучасним інтерфейсом, високим рівнем продуктивності та чудовим користувацьким досвідом на всіх основних мобільних платформах. Завдяки цьому фреймворку команда зосереджується на розробці функціоналу, а не дублюванні коду для різних ОС, що критично важливо на етапі MVP.

2.4.3. Використання FireBase для зберігання та опрацювання даних

Для реалізації бекенду мобільного застосунку Elegix було обрано Firebase – платформу від Google, яка надає готові хмарні сервіси для зберігання, обробки та синхронізації даних у реальному часі. Це рішення дозволяє значно прискорити розробку MVP, зменшити інфраструктурні витрати та сконцентруватись на бізнес-логіці застосунку.

Ключові переваги використання Firebase: До основних переваг MySQL належать:

- Realtime Database / Firestore: забезпечує синхронізацію даних у реальному часі, що дозволяє оновлювати стан застосунку миттєво для всіх користувачів без необхідності ручного оновлення.
- Authentication: підтримує готові механізми аутентифікації (Google, Apple, Email/Password тощо), що прискорює реалізацію безпечного входу користувача.
- Cloud Functions: дозволяє реалізовувати серверну логіку без потреби у власному сервері, використовуючи подієво-орієнтовану архітектуру.
- Cloud Storage: зручно використовувати для зберігання медіафайлів, аватарів, документів користувача тощо.
- Безпечний доступ: через правила безпеки Firebase можна гнучко налаштовувати права доступу до різних частин бази даних і файлів.
- Масштабованість: платформа автоматично масштабується при збільшенні навантаження, що є важливим для стартапів, які планують швидке зростання.

Firebase ідеально підходить для мобільних застосунків із динамічними інтерфейсами, які потребують обміну даними в реальному часі. Завдяки своїй гнучкості та широкому набору інструментів Firebase став ефективним бекенд-рішенням для Elegix, що дозволило пришвидшити розробку, знизити технічну складність та гарантувати безперебійну роботу з даними.

2.5. Особливості програмної реалізації

Програмна реалізація мобільного застосунку Elegix Organizer – системи управління цілями, звичками та сферами життя – була здійснена з використанням кросплатформенного фреймворку React Native (Expo) та хмарної платформи Firebase. Такий стек забезпечує швидку розробку MVP, зручну інтеграцію з мобільною інфраструктурою, а також масштабованість та безпеку обробки персональних даних.

2.5.1. Серверна частина (Backend – C# та .NET)

Авторизація та управління користувачами

Уся логіка реєстрації та входу в систему реалізована за допомогою Firebase Authentication. Підтримується автентифікація за email/паролем, а також Google-акаунтом (через Expo Google Sign-In). Після успішного входу застосунок зберігає токен користувача у SecureStore для повторної авторизації.

Користувачі мають можливість редагувати свій профіль, включаючи ім’я, email та фото (яке завантажується на Firebase Storage).

Структура даних та збереження інформації

Вся інформація зберігається у Firebase Cloud Firestore, що дозволяє працювати з NoSQL-структурою даних у реальному часі. Основні колекції бази даних:

- users – інформація про користувача, його вибрані сфери життя, активні цілі та прогрес;

- goals – збережені цілі користувача, з полями: назва, опис, сфера, дедлайн, статус, прогрес, план дій;
- habits – звички користувача, які можуть бути прив’язані до кількох цілей одночасно;
- actions – окремі кроки в межах цілей;
- reflections – нотатки користувача, звіти про досягнення, саморефлексії;
- categories – попередньо визначені сфери життя (наприклад, "Здоров’я", "Кар’єра" тощо).

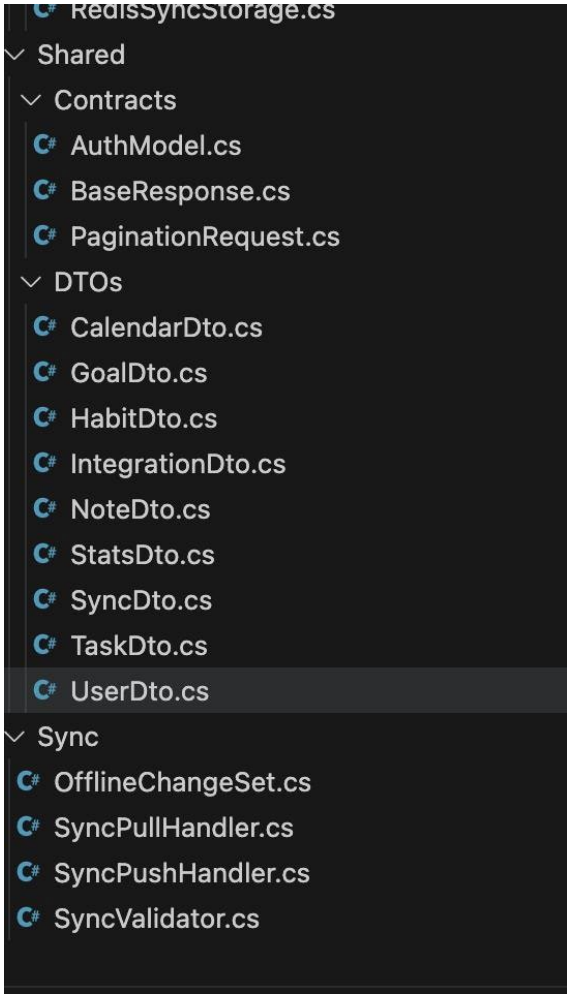

Рисунок 2.2 – Головна структура файлів

Обробка та логіка на стороні сервера. Для обробки складних запитів, збереження статистики або взаємозв’язків між цілями та звичками використовуються Firebase Cloud Functions. Наприклад:

- при завершенні певного кроку (action), автоматично оновлюється прогрес цілі;
- при щоденному завершенні звички – оновлюється streak та статистика в профілі користувача;
- при створенні цілі – відправляється вітальне повідомлення через Firebase Messaging (у майбутніх версіях).

Реактивний інтерфейс та робота з Firebase. Завдяки використанню React Native та Firebase SDK застосунок реагує на зміни в базі даних в реальному часі: цілі, звички, кроки оновлюються без потреби в ручному оновленні сторінки. Компоненти побудовані з урахуванням atomic design system, що дозволяє легко масштабувати інтерфейс та додавати нові блоки.

Управління зображеннями та файлами. Фото користувача та обкладинки до цілей завантажуються у Firebase Storage. Перед завантаженням зображення стискаються та оптимізуються на стороні клієнта за допомогою бібліотеки expo-image-manipulator.

Захист та безпека. Firebase надає правила безпеки доступу до бази даних та файлів. Встановлені правила дозволяють користувачеві доступ лише до його особистих документів, перевіряючи auth.uid на кожному запиті.

2.5.2. Клієнтська частина (Frontend – React Native)

Клієнтська частина застосунку розроблена з використанням бібліотеки React Native, що дозволяє створювати динамічний та інтерактивний користувацький інтерфейс. Для управління станом, маршрутизації та взаємодії з API використовуються відповідні інструменти та підходи.

Точкою входу є файл main.tsx, де ініціалізується кореневий React-компонент App. Компонент App налаштовує маршрутизацію за допомогою react-router-dom та обгортає застосунок у ThemeProvider (для застосування кастомної теми theme.tsx) та AuthProvider (для управління станом автентифікації).

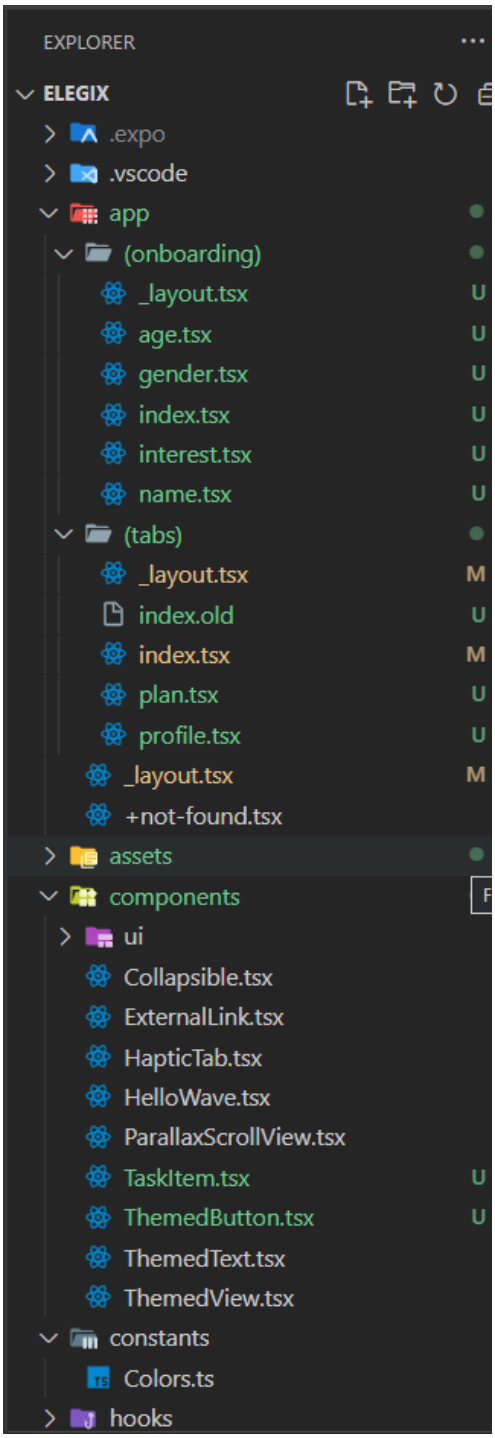

Рисунок 2.3 – Структура проєкту

Основні сторінки застосунку представлені файлами в директорії src/pages/(tabs): index.tsx (Головна сторінка) – (Рис 2.4), plan.tsx (Рис - 2.5), profile.tsx (Рис - 2.6) та інші.

Перевикористовувані UI-елементи, такі як навігаційна панель (Navbar.tsx), кнопки, форми, винесені в окремі компоненти в директорії src/components.

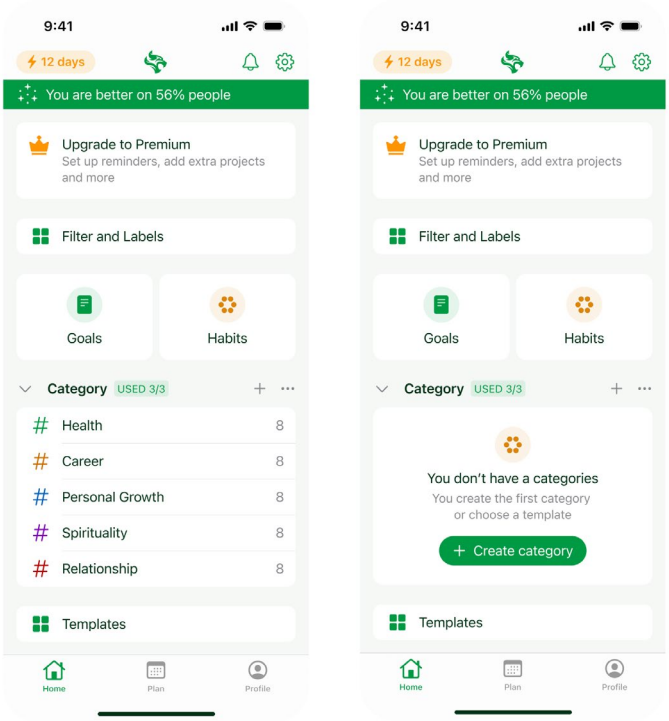

Рисунок 2.4 – Головна сторінка застосунку.

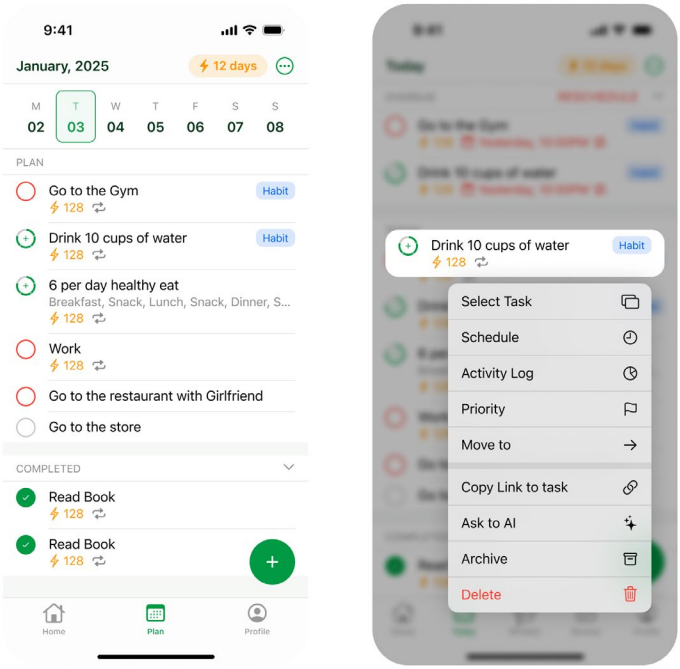

Рисунок 2.5 – Сторінка списку задач.

Модуль аутентифікації (Login) відповідає за безпечне підтвердження особи користувача, що дає доступ до персоналізованих функцій додатку Elegix. Модуль забезпечує підтримку трьох способів входу: через електронну пошту з паролем, через Google та через Apple ID (для iOS). Це підвищує зручність використання додатку та охоплює широкий спектр користувачів.

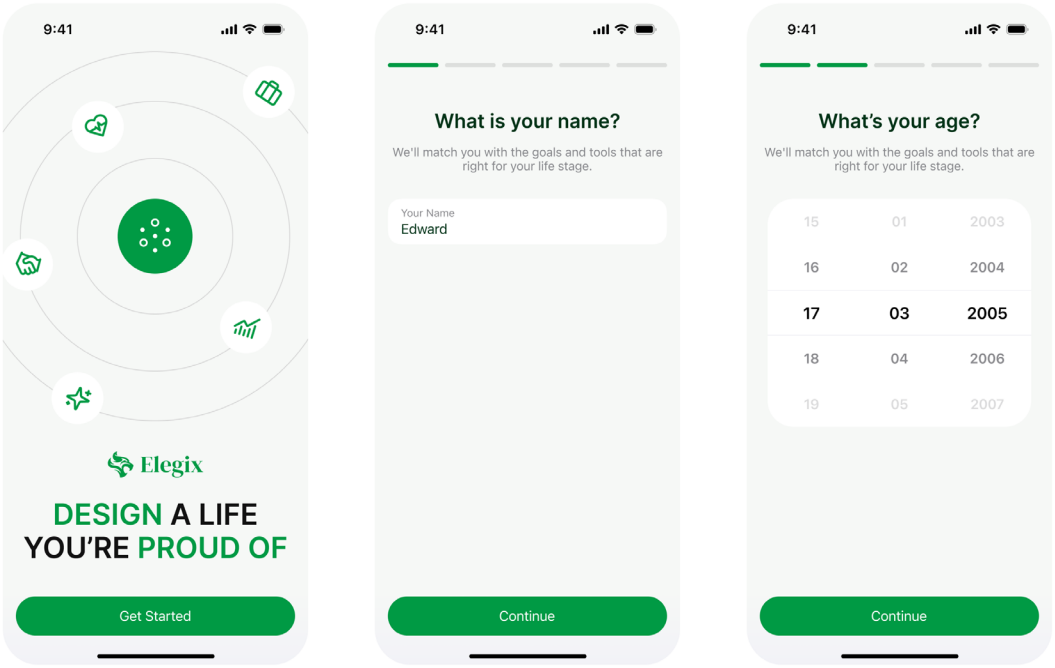

Рисунок 2.6 – Форма логіну та онбординг.

LoginScreen є React Native компонентом, що реалізує форму аутентифікації. Компонент створений із застосуванням бібліотеки Formik для управління станом форми і її валідацією, а також бібліотеки Yup для опису правил валідації.

Логіка роботи стандартної форми Email та Пароль.

Структура форми:

- Email (тип поля – email, обов'язкове для заповнення);
- пароль (тип поля – password, обов'язкове, мінімальна довжина 8 символів);
- кнопка "Увійти";
- посилання на сторінку реєстрації;

- повідомлення про помилки (некоректний email, неправильний пароль, мережеві помилки).

Валідація введених даних. Валідація здійснюється на стороні клієнта за допомогою Yup-схеми, яка містить:

- перевірку коректності email-адреси за шаблоном;
- обов'язковість заповнення пароля;
- мінімальна довжина пароля (8 символів);
- помилки валідації відображаються під відповідними полями у вигляді підказок.

Обробка події submit. При натисканні кнопки "Увійти" запускається обробник події:

- формуються дані форми (email, пароль);
- відправляється HTTP POST запит через axios на endpoint бекенду /api/login;
- очікується відповідь із токеном автентифікації (JWT) [24] та інформацією про користувача;

Обробка відповіді сервера.

Успішний вхід – токен зберігається у безпечному сховищі (SecureStorage / AsyncStorage). Контекст автентифікації оновлюється (AuthContext) – інформація про користувача зберігається у стані, що дозволяє контролювати доступ до захищених сторінок. Користувач перенаправляється на головну сторінку додатку (Dashboard / Home).

Помилка аутентифікації – відображається повідомлення про некоректний email чи пароль. Можливість повторити спробу.

Логіка соціальної аутентифікації (Google Sign-In, Apple Sign-In).

Google Sign-In та Apple Sign-In (для iOS)

- Використовується бібліотека react-native-google-signin та react-native-apple-authentication.
- При натисканні кнопки "Увійти через":
- Запускається процес аутентифікації через Google або Apple SDK.

- Користувач обирає акаунт, надає дозвіл на передачу даних.
- Отриманий токен Google або Apple передається на бекенд для верифікації та отримання JWT додатку.

Якщо верифікація успішна – оновлюється AuthContext, зберігається токен.

Модуль створення задачі (Рис 2.12) відповідає за інтуїтивний, швидкий та гнучкий процес формування нових задач користувачем у межах поставлених цілей. Цей модуль є ключовим у структурі системи планування та продуктивності, адже саме від якості постановки задач залежить ефективність їх виконання.

Компонент реалізовано на React Native з використанням Formik для управління станом форми, Yup для валідації, та Context API (або Redux) для інтеграції з глобальним станом додатку.

Основні поля форми створення задачі:

- Назва задачі (обов’язкове) – короткий і змістовний опис.
- Опис задачі (необов’язкове) – детальна інформація, інструкції, примітки.
- Сфера життя (обов’язкове) – вибір із 5 сфер (Здоров’я, Кар’єра, Відносини, Віра, Особистий розвиток).
- Ціль (Goal) (обов’язкове) – прив’язка задачі до однієї з цілей у вибраній сфері.
- Пріоритет (низький, середній, високий) – допомагає розставити акценти в плані.
- Термін виконання (deadline) – дата або час до якого задача має бути виконана.
- Повторюваність (опціонально) – можливість налаштувати періодичність задачі (щодня, щотижня тощо).
- Теги / Категорії – для кращої фільтрації та пошуку.

Також було додано функцію вкладених задач в цілях.

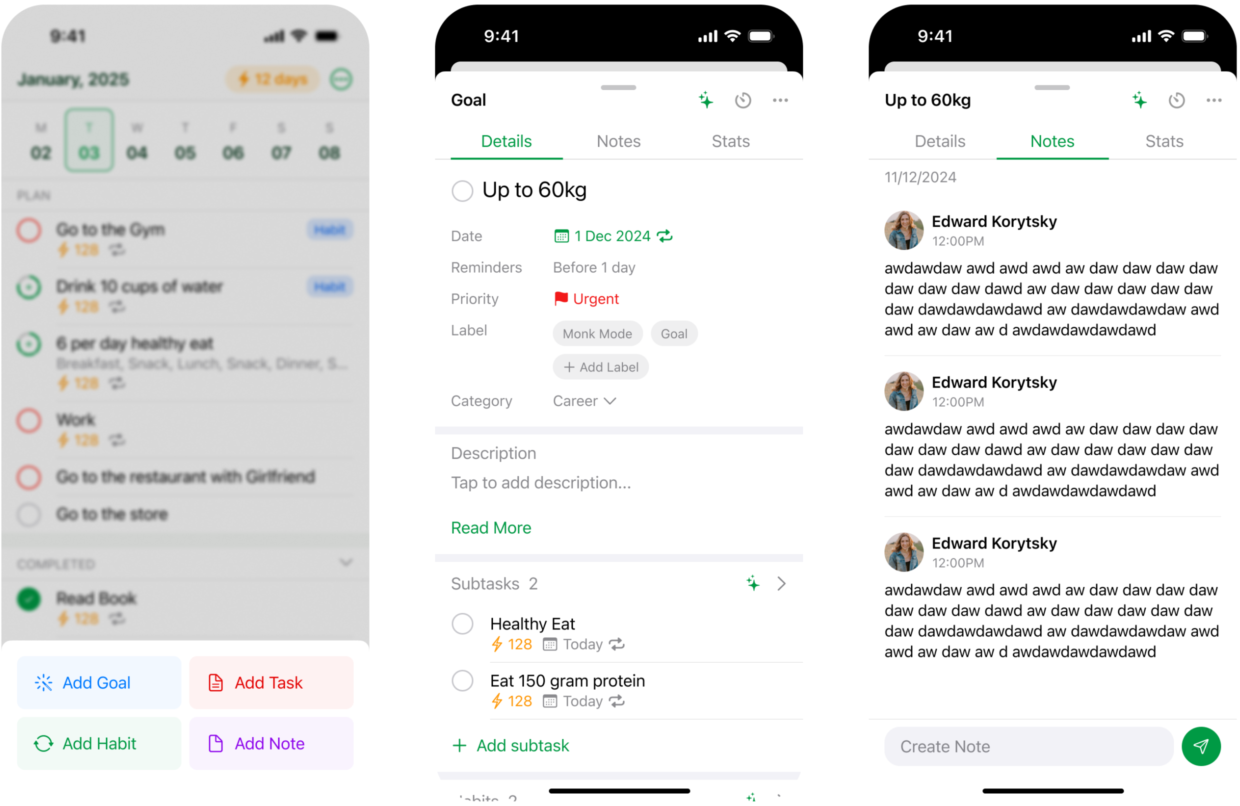

Рисунок 2.7 – Форма створення задачі

Модуль Профіль (Рис 2.8) забезпечує персоналізацію користувацького досвіду, дозволяючи редагувати особисту інформацію, керувати налаштуваннями додатку, безпекою, а також інтеграціями. Цей модуль покликаний зробити взаємодію користувача зі стартапом Elegix більш комфортною, безпечною та адаптованою під індивідуальні потреби.

- Редагування особистих даних (ім'я, прізвище, фото, email та інше)
- Зміна пароля та керування безпекою облікового запису
- Налаштування теми інтерфейсу (світла / темна)
- Налаштування підписок користувача
- Вибір робочих днів та вихідних (для індивідуальної продуктивності)
- Управління сповіщеннями (push, email)
- Налаштування інтеграцій (календар, таск-менеджери)

- Перегляд підключених девайсів
- Мова інтерфейсу
- Вихід із системи
- Синхронізація

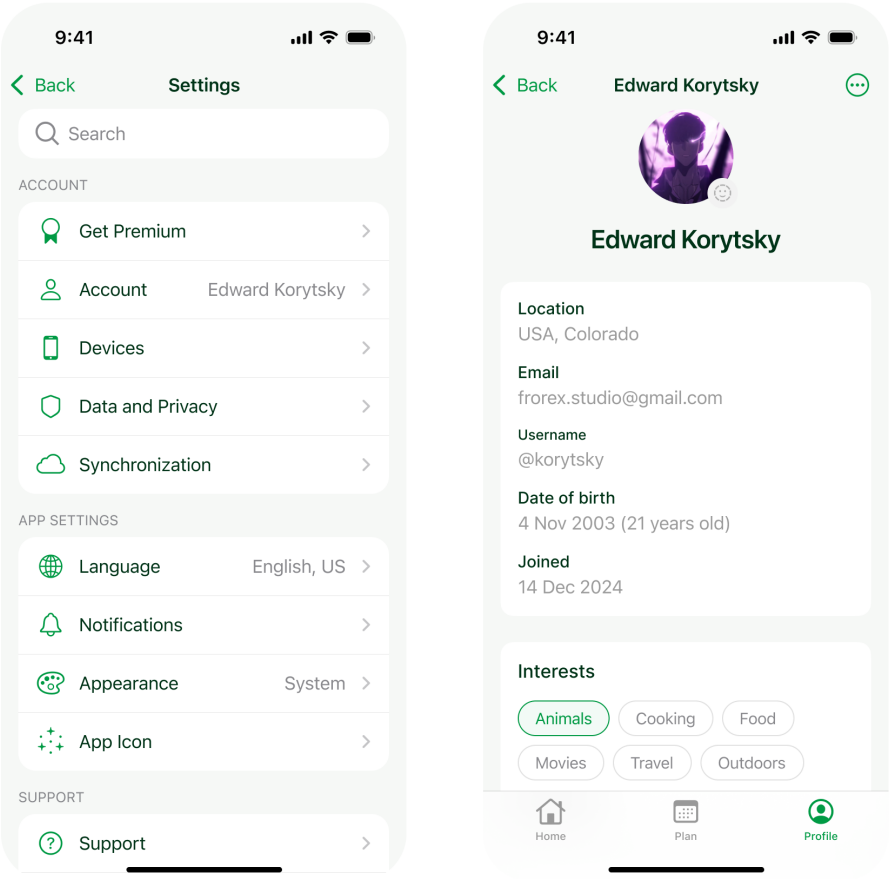

Рисунок 2.8 – Сторінка налаштувань акаунту

Модальні вікна (Рис 2.9) служить для ефективної комунікації з користувачем, надання важливої інформації, підтвердження дій і мотивації. Модальні вікна підвищують залученість користувача та допомагають підтримувати потрібний UX на кожному етапі взаємодії зі стартапом Elegix.

Основні типи модальних вікон у застосунку Elegix:

- Мотиваційні повідомлення. Наприклад, “Ура, ти набрав 5 стріків!” із можливістю поділитися досягненням у соцмережах.

- Підтвердження дій. Наприклад, питання: “Ви змінили мову на девайсі, хочете переключити мову в застосунку?”
- Запити на відгуки та оцінку. Наприклад, “Оцініть Elegix” із кнопками для швидкої оцінки або переходу до магазину застосунків.

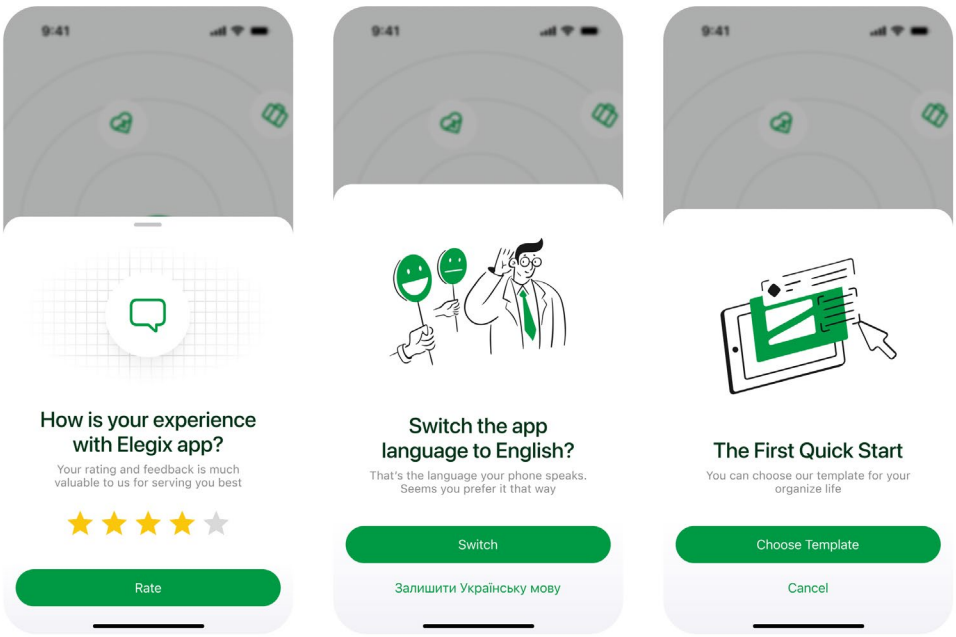

Рисунок 2.9 – Інтерфейси модальних вікон

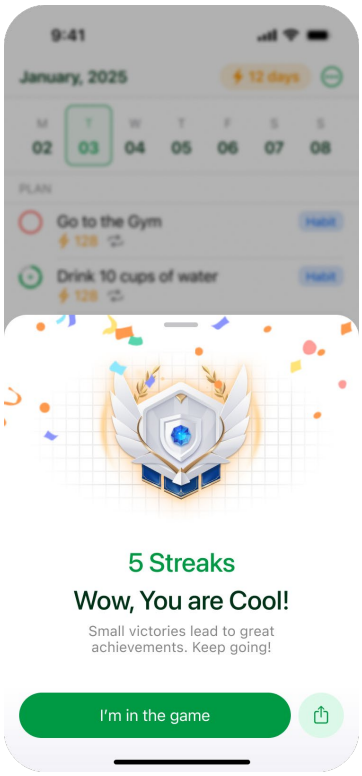

Рисунок 2.10 – Екран прогресу стріків.

Для побудови користувацького інтерфейсу застосунку Elegix використовується власна кастомна бібліотека компонентів, розроблена спеціально під унікальні потреби проєкту. У файлі theme.tsx визначено кастомну тему – кольорову палітру, типографіку та стилі, які відповідають філософії бренду Elegix і застосовуються до всього застосунку через ThemeProvider.

Компоненти цієї бібліотеки – Button, TextField, Container, Box, AppBar, Avatar, Paper, Checkbox, Radio, List, Grid, Chip – створені для забезпечення візуальної цілісності, преміального вигляду і високої юзабіліті. Наприклад, навігаційна панель Navbar.tsx використовує AppBar, Toolbar, Button та Avatar із кастомної бібліотеки, що відповідають стилю та архетипу Правителя, який символізує контроль і порядок.

Форми, як у TaskCreate.tsx чи AccountSettings.tsx, застосовують TextField, Button, Select із власної бібліотеки, що гарантує зручність вводу, зрозумілість і легкість у користуванні. Такий підхід дозволяє Elegix підтримувати преміальний рівень дизайну та підсилює впізнаваність бренду через унікальний, цілісний UI..

2.6. Організація тестування та налагодження програмного засобу

Для забезпечення стабільної роботи та високої якості продукту Elegix тестування та налагодження виконувалися комплексно на всіх етапах розробки.

Функціональне тестування здійснювалося постійно під час впровадження кожного нового функціоналу, зокрема:

Реєстрація користувачів та різні варіанти автентифікації (зокрема через Google, Apple та звичайний логін).

Створення, редагування, перегляд та управління задачами в рамках різних життєвих сфер.

Робота з особистим профілем користувача, налаштуваннями застосунку та відстеженням прогресу (стріки, досягнення).

Взаємодія з модальними вікнами, оповіщеннями та системою мотивації.

UI/UX тестування зосереджувалося на відповідності інтерфейсу кастомній бібліотеці компонентів Elegix, забезпечуючи високу зручність користування, візуальну цілісність і логіку взаємодії. Перевірялося коректне відображення елементів на різних пристроях і в різних умовах, щоб підтримувати преміальний вигляд та інтуїтивність.

Інтеграційне тестування охоплювало перевірку коректної взаємодії клієнтської частини (React Native) з бекендом (.NET), включно з роботою API, автентифікацією, оновленням даних, а також обробкою помилок і статусів.

Для прискорення перевірки API на етапі розробки активно використовувалися інструменти на кшталт Postman.

Налагодження коду проводилося із застосуванням стандартних засобів розробника, включаючи консоль браузера, інспектор DOM, моніторинг мережевих запитів, що дозволяло оперативно знаходити і усувати помилки.

Перед початком розробки та протягом усього життєвого циклу застосунку Elegix активно застосовувався підхід customer development. Було проведено серію інтерв'ю, опитувань та тестових запусків із цільовою аудиторією, щоб глибше зрозуміти їхні потреби, проблеми та очікування щодо продуктивності, саморозвитку та мотивації.

Аналіз ринку показав високий попит на інструменти, що допомагають уникнути відволікань і підтримувати фокус, а також на персоналізовані рішення для постановки та досягнення цілей у різних сферах життя. Цей аналіз визначив ключові функції, які були реалізовані в застосунку – онбординг з кількома способами логіну, управління задачами у життєвих сферах, гейміфікація у вигляді стріків, а також модальні оповіщення для підтримки мотивації.

Це дозволило створити продукт, що відповідає реальним запитам користувачів, підвищує їхню продуктивність та сприяє особистісному розвитку, закладаючи міцний фундамент для подальшого масштабування та розвитку.

2.7. Рекомендації щодо використання та впровадження програмного засобу

Розроблений застосунок Elegix створений для підвищення продуктивності, саморозвитку та досягнення цілей у різних сферах життя. Для максимально ефективного використання та успішного впровадження продукту варто врахувати кілька важливих аспектів.

Для користувачів, які прагнуть покращити свій режим, рекомендується пройти повну реєстрацію та налаштувати особистий профіль, обрати ключові сфери життя для розвитку і поставити цілі. Активне використання функції створення задач, планування дій і відстеження стріків допоможе підтримувати мотивацію та фокус. Варто регулярно користуватися модальними нагадуваннями і оповіщеннями, що допомагають не втрачати продуктивність і святкувати досягнення.

Для команди, яка буде підтримувати та розвивати застосунок, важливо звернути увагу на налаштування безперебійної роботи бекенду (.NET) і клієнтської частини (React Native). Забезпечення безпеки даних, регулярне оновлення системи, резервне копіювання користувацьких даних – це базові вимоги для стабільної роботи. Особливу увагу слід приділити швидкості синхронізації даних і коректній роботі API.

Перспективними напрямками розвитку Elegix можуть бути:

- розширення функціоналу для глибшої персоналізації цілей та звичок;
- впровадження системи нагород і гейміфікації для підвищення залученості;
- інтеграція з популярними календарями та додатками для автоматизації планування;
- запровадження розширеної аналітики для відстеження прогресу та виявлення зон для покращення;
- можливість ділитися досягненнями у соцмережах напряму із застосунку;
- покращення мобільної адаптації та оптимізація інтерфейсу під різні пристрої.

Дотримання цих рекомендацій забезпечить не лише комфортне і продуктивне використання Elegix для кінцевих користувачів, а й стабільне масштабування та розвиток платформи в майбутньому.

ВИСНОВКИ

У процесі виконання дипломної роботи було успішно досягнуто поставленої мети – розроблено застосунок Elegix, який сприяє підвищенню продуктивності, саморозвитку та досягненню особистих цілей у різних сферах життя. Розробка базувалась на сучасних технологіях: серверна частина реалізована на мові програмування C# із використанням платформи .NET, а клієнтська – на JavaScript-бібліотеці React Native, що забезпечує кросплатформену мобільну підтримку.

Робота починалася з глибинного аналізу потреб цільової аудиторії, проведення customer development і вивчення ринку інструментів продуктивності та саморозвитку. Це дозволило чітко сформулювати вимоги до функціоналу з урахуванням 5 ключових сфер життя та концепції ікігай. Основними завданнями стали розробка системи постановки та відстеження цілей, створення задач і планів дій, впровадження механізмів мотивації (стріки, нагороди), а також створення інтуїтивного інтерфейсу з власною кастомною бібліотекою компонентів.

Об’єктом дослідження був процес створення мобільного застосунку для організації продуктивного і збалансованого способу життя, предметом – ефективність інструментів самоменеджменту і мотивації. Результатом роботи став функціональний застосунок Elegix, що допомагає користувачам свідомо ставити цілі, планувати їх досягнення та підтримувати високий рівень продуктивності без вигорання.

Таким чином, всі поставлені завдання були виконані, а мета – досягнута. Elegix є практичним рішенням для людей, які прагнуть покращити якість свого життя через дисципліну, усвідомленість і системність. Застосунок має великий потенціал для подальшого розвитку, зокрема через розширення функціоналу, інтеграцію з іншими сервісами та вдосконалення користувацького досвіду..

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний застосунок. Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA
2. App Store. App Store. URL: https://www.apple.com/app-store/
3. Google Play. Google Play. URL: https://play.google.com/store/games?hl=en
4. What Is Android OS? History, Features, Versions, and Benefits. spiceworks.com. URL: https://www.spiceworks.com/tech/tech-general/articles/android-os/
5. Apple iOS : що це таке. gsmhub.com.ua URL: https://gsmhub.com.ua/glossary/apple-ios
6. What is an API (Application Programming Interface)? aws.amazon.com URL: https://aws.amazon.com/what-is/api/
7. Push Notification. useinsider.com. URL: https://useinsider.com/glossary/push-notification/
8. Типи мобільних додатків. training.qatestlab.com. URL: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/
9. React Native Official Documentation. Meta Platforms, Inc. URL: https://reactnative.dev/
10. Мінімально життєздатний продукт (Minimum viable product, MVP). Uxpub. URL: https://ux.pub/zhmikhov/minimalno-zhittiezdatnii-produkt-minimum-viable-product-mvp-3if3
11. User interface це не просто прикраса продукту. foxminded.ua. URL: https://foxminded.ua/user-interface-tse/
12. Практичне застосування Zustand, від теорії до практики. hackyourmom.com. URL: https://hackyourmom.com/osvita/praktychne-zastosuvannya-zustand-vid-teoriyi-do-praktyky/
13. Redux Official Documentation. redux.js.org. URL: https://redux.js.org/

14. Expo Official Documentation. Expo.dev. URL: https://expo.dev/
15. Clean Architecture in ASP .NET Core Web API. Medium.com. URL: https://medium.com/@mohanedzekry/clean-architecture-in-asp-net-core-web-api-d44e33893e1d
16. Що ж це таке GraphQL?. Medium.com. URL: https://cryptoaxel.medium.com/%D1%89%D0%BE-%D0%B6-%D1%86%D0%B5-%D1%82%D0%B0%D0%BA%D0%B5-graphql-5ea333add9e9
17. What is CI/CD?. RedHat.com. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd
18. Habitica. Habitica. URL: https://habitica.com/
19. Dot Net Official Documentation. dotnet.microsoft.com. URL: https://dotnet.microsoft.com/en-us/
20. Visual Studio Code. Code.visualstudio.com. URL: https://code.visualstudio.com/
21. Visual Studio. visualstudio.microsoft.com. URL: https://visualstudio.microsoft.com/
22. Cross-site scripting. portswigger.net. URL: https://portswigger.net/web-security/cross-site-scripting
23. Azure. Azure.microsoft.com. URL: https://azure.microsoft.com/en-us/
24. Розуміння JWT (JSON Web Token). Vpnunlimited.com. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/jwt?srsltid=AfmBOopgRjKAc5kHpQFLMuc0w-lSmp5rpRJ40BzI_PLMgv84ShYAJCLa

ДОДАТКИ

Додаток А

Технічне завдання

1. Вступ

Дане технічне завдання описує мобільний застосунок Elegix – платформу для особистого розвитку, що допомагає користувачам визначати свої життєві пріоритети, ставити цілі, будувати покрокові плани, формувати корисні звички та досягати більшого балансу в 5 ключових сферах життя. Застосунок орієнтований на людей, які прагнуть підвищити ефективність, знайти внутрішню гармонію та реалізувати свій потенціал згідно з концепцією ікігай.

2. Підстави для розробки

Розробка проводиться на основі індивідуального завдання, поставленого науковим керівником дипломної роботи для спеціальності "Комп’ютерні науки та кібербезпека" (або вашої спеціальності).

3. Призначення розробки

Elegix покликаний допомогти користувачам аналізувати себе в розрізі 5 сфер життя. Ставити усвідомлені цілі відповідно до обраних сфер. Розбивати цілі на покроковий план дій. Закріплювати результат через систему звичок. Відстежувати прогрес та переглядати досягнення. Отримувати рекомендації на основі ШІ.

Платформа має бути однаково зручною для індивідуального використання, а також потенційно масштабованою для мікроспільнот чи команд.

4. Вимоги до програми чи програмного продукту

4.1. Вимоги до функціональних характеристик

Система повинна надавати можливість реєстрації нових користувачів, з вказанням імені, email, пароля, статі та дати народження. Реєстрація супроводжується первинним опитуванням, у якому користувач обирає 3 із 5 сфер життя для розвитку (Здоров’я, Стосунки, Віра/Духовність, Кар’єра, Саморозвиток).

Система повинна забезпечувати безпечний вхід зареєстрованих користувачів за допомогою облікових даних (email та пароль, або OAuth через Google/Apple).

Користувачі повинні мати можливість редагувати дані свого профілю (ім'я, стать, дата народження, email, фото, зміна пароля, обрані сфери життя).

Користувачі повинні мати можливість створювати цілі, прив’язані до обраних сфер життя. Для кожної цілі вказується назва, опис, дедлайн, мотивація (чому ця ціль важлива), рівень пріоритету, а також статус (активна, відкладена, завершена).

До кожної цілі повинна бути можливість створення покрокового плану дій – списку завдань, з можливістю додавати дедлайни, коментарі, позначки прогресу.

Система повинна дозволяти додавати звички, як частину цілей або окремо. Звички можуть бути одноразовими або повторюваними (щодня, щотижня тощо). Кожна звичка має статус виконання, може мати трекер із кількістю виконань на день.

Система повинна забезпечувати відображення прогресу по звичках та цілях у вигляді графіків, відсотків виконання та streak’ів (серій виконання).

4.2. Вимоги до надійності

1. Стабільна робота застосунку без збоїв.
2. Захист даних користувача.
3. Обробка некоректного введення.
4. Захист від типових загроз: SQL-ін’єкцій, XSS, CSRF.

4.3. Умови експлуатації

1. Працює на мобільному пристрої.
2. Доступ через інтернет та офлайн.
3. Працює 24/7, крім технічного обслуговування.

4.4. Вимоги до складу і параметрів технічних засобів

Будь-який планшет або смартфон з встановленою сучасною операційною системою (Android, IOS) та доступом до мережі Інтернет.

4.5. Вимоги до інформаційної і програмної сумісності

1. Клієнт: React або React Native (Expo) – для PWA/мобільної версії.
2. Сервер: ASP.NET Core Web API (C#).
3. Комунікація: REST API, JSON, HTTPS.

4.6. Вимоги до транспортування і збереження

1. Дані зберігаються у БД Firebase.
2. Шифрування паролів.
3. Токенна автентифікація (JWT).
4. HTTPS-з'єднання.

5. Вимоги до програмної документації

1. Опис архітектури системи (Clean Architecture, клієнт-серверна модель).
2. Опис реалізованого функціоналу та його відповідності вимогам.
3. Опис структури бази даних (схеми, зв'язки).
4. Опис ключових алгоритмів та особливостей програмної реалізації (бекент та фронтент).
5. Інструкцію користувача (основні сценарії використання).
6. Результати тестування (опис проведеного мануального тестування).
7. Висновки та рекомендації щодо подальшого розвитку.

6. Техніко-економічні показники

1. Оцінка витрат на розробку (в контексті дипломної роботи – це переважно часові витрати студента).
2. Обґрунтування вибору технологій з точки зору доступності, популярності та наявності ресурсів для розробки.
3. Потенційна корисність та ефективність застосунку для організації онлайн-навчання.

7. Стадії і етапи розробки

7.1. Передбачені стадії розробки

Дослідження поняття органайзерів, аналіз аналогів, формулювання вимог до системи.

Розробка архітектури (Clean Architecture), проєктування бази даних, проєктування користувацького інтерфейсу (UI/UX).

Налаштування проєкту .NET та бази даних.

Реалізація моделей, міграцій, контролерів, маршрутів API.

Реалізація логіки автентифікації, онбордінг, категоріями та ШІ помічник.

Налаштування проєкту React Native.

Розробка компонентів UI, сторінок, маршрутизації.

Реалізація логіки взаємодії з API, управління станом.

Інтеграція Backend та Frontend.

Тестування та налагодження: Проведення мануального тестування функціоналу, UI, інтеграції.

Підготовка документації: Написання пояснювальної записки до дипломної роботи.

7.2. Необхідні сторінки та структурні елементи на сайт

Назва елементу	Опис
HomePage (Головна сторінка)	Відображення загальної інформації про платформу, можливо, список категорій та фільтрів.
LoginPage (Сторінка входу)	Форма для введення email та пароля для автентифікації. Або вибір авторизації через соціальні мережі
RegisterPage (Сторінка реєстрації)	Форма для створення нового облікового запису.
PlanPage (Сторінка задач)	Відображення списку всіх задач з можливістю пошуку та фільтрації.

ProfilePage (Сторінка профілю користувача)	Відображення інформації про користувача та можливість відредагувати дані.
Category/plan (Сторінка задач в певній категорії)	Відображення списку всіх задач в певній категорії з можливістю пошуку та фільтрації.
SettingsPage (Налаштування акаунту)	Редагування застосунку.

8. Порядок контролю і приймання

1. Контроль за виконанням кожного етапу розробки здійснюється науковим керівником.

2. Приймання роботи відбувається шляхом захисту курсової роботи, що включає:

   a. Демонстрацію працездатності мобільного застосунку "Elegix".

   b. Перевірку відповідності реалізованого функціоналу вимогам даного ТЗ.

   c. Оцінку якості програмної реалізації та дизайну.

   d. Аналіз представленої програмної документації (пояснювальної записки).

Додаток Б

Інструкція користувача

Для ефективного використання мобільного застосунку "Elegix" та доступу до інструментів саморозвитку, планування цілей і звичок – спочатку потрібно зареєструватися або увійти до свого облікового запису.

Після успішної автентифікації вам відкриються всі можливості платформи. Через інтуїтивну навігацію ви зможете обрати 3 життєві сфери, над якими хочете працювати, створювати персоналізовані цілі та звички, планувати дії та відстежувати прогрес.

Завдяки вбудованому штучному інтелекту Elegix допоможе сформулювати правильну ціль, розбити її на кроки, запропонує ідеї звичок і підтримуватиме вашу мотивацію. Кожну ціль можна доповнити дедлайнами, мотиваційними нотатками, а звички – гнучким трекером із повтореннями та нагадуваннями.

У процесі саморозвитку ви зможете проходити тестування, чеклісти та діагностику прогресу, результати яких зберігатимуться у вашому профілі. Система автоматично обраховує досягнення, трекінг та динаміку вашого росту.

В особистому кабінеті користувача доступна можливість редагування профілю, оновлення обраних сфер, цілей, шаблонів звичок, а також перегляд повної історії досягнень та статистики.

Elegix – це ваш особистий простір розвитку, де кожен крок наближає вас до життя, яким ви пишаєтесь.

АНОТАЦІЯ

Корицького Е. Р. – **Розробка та проєктування мобільного застосунку «Elegix» для організації плану життя з використанням .NET, мови C# та бібліотеки React Native**

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

У дипломній роботі досліджено сучасні підходи до саморозвитку, систем постановки цілей і трекінгу звичок. Проведено аналіз популярних мобільних рішень на кшталт Fabulous, Habitica та Notion, що дозволило сформувати функціональні та UX-вимоги до власного застосунку.

На основі аналізу було обґрунтовано вибір стеку технологій: .NET Core (C#) для серверної частини з побудовою RESTful API, React Native для кросплатформенного клієнтського застосунку, а також кастомної дизайн-системи UI-компонентів, адаптованої під стиль бренду та принципи Clean Architecture. Система включає модулі: реєстрації та автентифікації користувача, вибору життєвих сфер для розвитку, створення цілей, звичок і планів дій, а також відстеження прогресу. Реалізовано гнучкий механізм трекінгу, історії досягнень та інтеграцію зі штучним інтелектом для покращення постановки цілей.

В результаті розроблено функціональний прототип мобільного застосунку «Elegix», який може використовуватись як персональний інструмент саморозвитку або як база для масштабування в комерційний продукт.

Ключові слова: мобільний застосунок, саморозвиток, .NET, C#, React Native, RESTful API, Clean Architecture, трекінг звичок, постановка цілей.