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. Клієнтська частина.
(тема)

Виконала:
здобувач 4 року навчання
групи ПЗПІ-21-1

Надія ДАШКО
(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)
«___» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дашко Надії Михайлівні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для планування та моніторингу виконання особистих задач і досягнень. Клієнтська частина.
Затверджена наказом по університету від 19.05.2025 №397Ст
2. Термін подання студентом роботи до екзаменаційної комісії 11.06.2025
3. Вихідні дані до роботи *Розробити клієнтську частину вебзастосунку, що включає функціонал реєстрації, авторизації, управління задачами, взаємодії між користувачами, чат з асистентом та перегляду статистики. Реалізацію здійснити з використанням мови JavaScript, бібліотеки React, адаптивної верстки та інтеграції з серверним API.*
4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, прийняті програмні рішення, реалізація клієнтської частини, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2025	*виконано*
2	Створення специфікації ПЗ	10.04.2025	*виконано*
3	Проектування ПЗ	17.04.2025	*виконано*
4	Розробка ПЗ	30.04.2025	*виконано*
5	Тестування ПЗ	17.05.2025	*виконано*
6	Оформлення пояснювальної записки	21.05.2025	*виконано*
7	Підготовка презентації та доповіді	25.05.2025	*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: 101 с., 28 рис., 16 джерел.

АВТОРИЗАЦІЯ, ВЕБЗАСТОСУНОК, ВЗАЄМОДІЯ, ДОСЯГНЕННЯ, ДРУЗІ, ЗАДАЧІ, ІНТЕРФЕЙС, КОРИСТУВАЧ, МОНІТОРИНГ, НАВІГАЦІЯ, ПЛАНУВАННЯ, ПОВІДОМЛЕННЯ, РЕЄСТРАЦІЯ, СТАТИСТИКА, UI/UX.

Об'єктом дослідження є програмна система для особистого планування, відстеження прогресу та досягнень користувача в рамках інтерактивного застосунку з можливістю соціальної взаємодії. Дослідження включало аналіз потреб користувача, особливостей побудови інтерфейсів користувача для систем самоконтролю, а також реалізації зручного та адаптивного вебінтерфейсу з урахуванням інтеграції із зовнішніми API.

Метою кваліфікаційної роботи є розробка функціональної та зручної клієнтської частини вебзастосунку, що дозволяє користувачам створювати та моніторити особисті завдання, отримувати власні досягнення, взаємодіяти з іншими користувачами та отримувати системні повідомлення.

Методи розробки базуються на сучасних технологіях фронтенду, зокрема бібліотеці React.js у поєднанні з препроцесором стилів LESS та менеджером маршрутизації React Router. Для налаштування структури проєкту та оптимізації збірки використовувався інструмент Vite. Уся клієнтська частина взаємодіє з серверною частиною через REST API, запити до якої реалізовано за допомогою fetch. Розробка велась у локальному середовищі розробки Visual Studio Code з використанням системи контролю версій Git.

У результаті виконаної роботи було було реалізовано функціональний інтерфейс вебзастосунку, що забезпечує користувачеві можливість керування особистими задачами, досягненнями та статистикою, спостереження за власним прогресом, а також базову соціальну взаємодію у межах системи.

ABSTRACT

AUTHENTICATION, WEB APPLICATION, INTERACTION, ACHIEVEMENTS, FRIENDS, TASKS, INTERFACE, USER, MONITORING, NAVIGATION, PLANNING, NOTIFICATIONS, REGISTRATION, STATISTICS, UI/UX.

The object of the research is a software system for personal planning, tracking user progress and achievements within an interactive application that enables social interaction. The study involved analysing user needs, the peculiarities of building user interfaces for self-monitoring systems, as well as implementing a user-friendly and adaptive web interface considering integration with external APIs.

The aim of this qualification work is to develop a functional and user-friendly client side of the web application that allows users to create and monitor personal tasks, receive their own achievements, interact with other users, and receive system notifications.

The development methods are based on modern frontend technologies, in particular the React.js library in combination with the LESS style preprocessor and the React Router routing manager. For structuring the project and optimising the build, the Vite tool was used. The entire client side interacts with the server side via a REST API, with requests implemented using fetch. The development was conducted in the local development environment Visual Studio Code, using the Git version control system.

As a result of the completed work, a functional interface of the web application was implemented, providing the user with the ability to manage personal tasks, achievements, and statistics, monitor their own progress, as well as basic social interaction within the system.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

DOM – Document Object Model

ESLint – ECMAScript Linter

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JWT – JSON Web Token

JSON – JavaScript Object Notation

LESS – Leaner Style Sheets

OAuth – Open Authorization

SPA – Single Page Application

UI – User Interface

URL – Uniform Resource Locator

UX – User Experience

XML – eXtensible Markup Language

ВСТУП

У сучасному інформаційному суспільстві стрімко зростає потреба у засобах, що дозволяють людині ефективно організовувати власний час, розподіляти задачі, відстежувати прогрес у досягненні особистих цілей та підвищувати загальну продуктивність. У відповідь на ці виклики активно розвивається сегмент цифрових інструментів особистого планування, що поєднують функціональність класичних органайзерів, систем самоорганізації та аналітики. Зокрема, вебзастосунки стають зручним середовищем для взаємодії з такими системами, оскільки забезпечують доступ з будь-якого пристрою та дозволяють гнучко адаптувати функціонал під потреби конкретного користувача.

Одним із ефективних підходів до формування залученості користувача є застосування елементів гейміфікації – інтеграція балів, досягнень, рівнів складності, а також соціальних механік, що забезпечують емоційну підтримку та зворотний зв’язок. Такі інструменти дозволяють підсилити мотивацію до виконання завдань і сприяють збереженню дисципліни в довгостроковій перспективі. Разом із тим, особливу роль відіграє інтерфейс, що має бути інтуїтивно зрозумілим, адаптивним і орієнтованим на зручність користувача.

У межах даної кваліфікаційної роботи реалізовано клієнтську частину програмної системи для планування та моніторингу особистих задач і досягнень. Основна увага приділена розробці інтерфейсу користувача, його взаємодії із серверною частиною через API, реалізації логіки обробки задач, побудови візуалізацій прогресу та інтеграції елементів соціальної взаємодії. В рамках роботи також враховано вимоги до адаптивності, швидкодії та підтримки сучасних браузерів. Розробка здійснювалася з використанням бібліотеки React, що забезпечує модульність, перевикористовуваність компонентів та легкість масштабування в подальшому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної галузі

У сучасному динамічному суспільстві, що характеризується високою інтенсивністю інформаційних потоків і багатозадачністю, здатність до ефективного планування та самоменеджменту є визначальним чинником успішності як у професійній, так і в особистій діяльності. Люди, залучені до освітніх, кар’єрних чи особистісних проєктів, потребують зручних інструментів для постановки цілей, організації повсякденних справ, контролю часу та моніторингу досягнень.

З огляду на це, зростає попит на інтерактивні цифрові системи, що допомагають користувачам структурувати свої завдання, планувати короткі і довгострокові цілі, а також підтримувати мотивацію шляхом відображення прогресу та результатів. Такі системи забезпечують користувачеві не лише функціонал для ведення задач, але й розширені можливості з аналізу власної ефективності, завдяки використанню статистичних даних, графіків та візуалізації результатів.

Особливу роль у таких системах відіграє гейміфікація процесу, тобто використання ігрових механік у неігровому контексті для підвищення залученості, мотивації та рівня взаємодії користувача із застосунком. Впровадження балів, віртуальних нагород, рівнів складності задач, щоденних викликів, досягнень та таблиць лідерів сприяє формуванню в користувачів внутрішньої мотивації до регулярного виконання завдань і досягнення поставлених цілей. Такий підхід також дозволяє користувачам відчути поступ, навіть за виконання дрібних дій, що особливо важливо в контексті довготривалого планування [1].

Предметна область охоплює широкий спектр функціональних компонентів: створення, редагування та категоризацію задач; відображення динаміки виконання у вигляді діаграм і показників; реалізацію гейміфікаційних елементів, зокрема системи балів, рівнів, досягнень і таблиці лідерів; push-сповіщення з нагадуваннями, мотиваційними повідомленнями та викликами; соціальну

взаємодію через перегляд акаунтів інших користувачів, додавання в друзі й обмін активністю.

У контексті розробки вебзастосунку особливе значення має реалізація зручного й адаптивного інтерфейсу, що забезпечує позитивний користувацький досвід, швидкий відгук системи на дії користувача та ефективну синхронізацію даних між клієнтською та серверною частинами.

Таким чином, програмна система для планування та моніторингу виконання особистих задач і досягнень постає як багатофункціональний сервіс, що об’єднує інструменти організації часу, зворотного зв’язку, гейміфікації та соціальної підтримки, спрямовані на підвищення продуктивності, залученості та особистої відповідальності користувача.

1.2 Огляд існуючих систем

Перейдемо до огляду наявних аналогів до системи що розроблюється. Конкуренція у сфері продуктивних застосунків еволюціонувала, змістивши фокус на гейміфікацію та індивідуалізований підхід. Спершу на ринку домінували прості трекери завдань, але з появою гейміфікованих інструментів мотивація через ігрові елементи стала ключовим трендом.

Першим і головним конкурентом розроблюваного застосунку є Habitica (див. рис. 1.1). Це ігровий інструмент, що застосовує принципи гейміфікації для покращення повсякденних звичок і підвищення продуктивності користувачів у реальному житті. Система перетворює звички, щоденні завдання та інші справи на ігрові елементи, зокрема на віртуальних «монстрів», яких необхідно подолати шляхом успішного виконання відповідних активностей [2].

У процесі досягнення цілей користувач просувається по грі, отримуючи винагороди та досягнення, що слугують додатковим мотиваційним стимулом. Водночас, невиконання завдань або помилки у реальному житті призводять до негативних наслідків для ігрового персонажа, що символізує відкат у прогресі. Habitica займає значну частку ринку гейміфікованих інструментів для продуктивності, будучи одним із найвідоміших рішень у цій сфері. Згідно з

відкритими джерелами, застосунок активно використовують понад 4 мільйони користувачів по всьому світу, що робить його важливим гравцем на ринку інструментів для трекінгу звичок і завдань.

Для залучення користувачів Habitica активно використовує командні випробування, групові «челленджі» та спільноту. Колективна мотивація та соціалізація, реалізована у формі групових завдань і рейтингових таблиць, стимулює користувачів продовжувати роботу над звичками. У розроблюваному застосунку соціальна складова відрізняється: фокус на персоналізованій взаємодії через додавання друзів та відправлення запитів, що краще підходить для індивідуального використання.

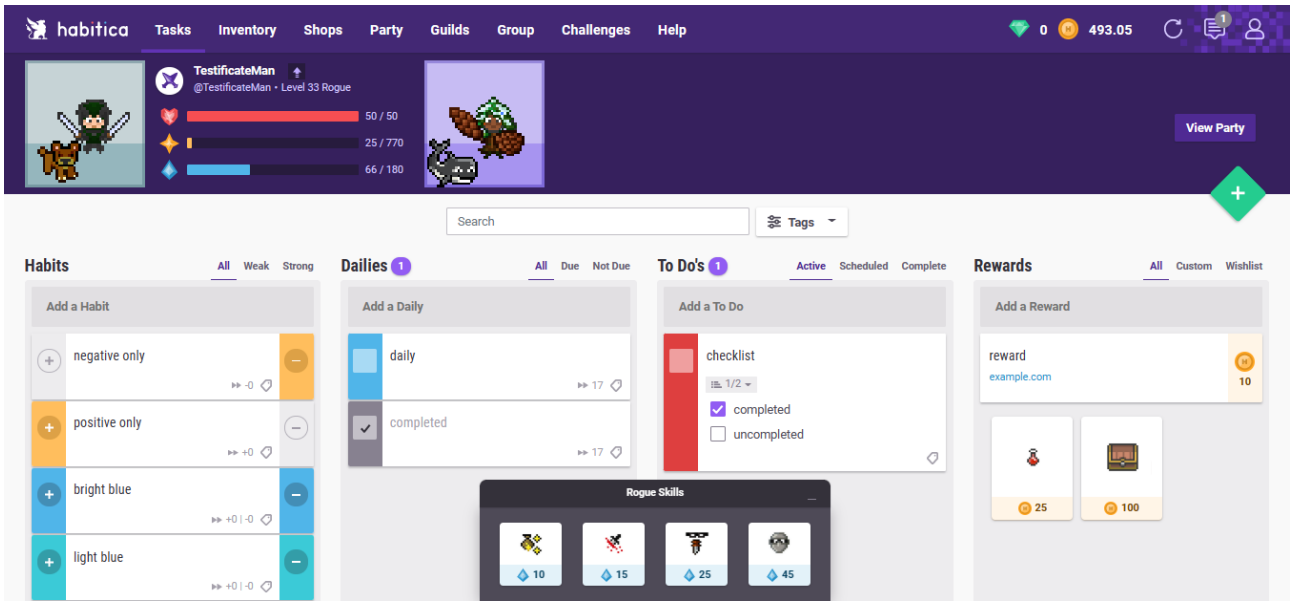

Рисунок 1.1 – Головна сторінка Habitica (рисунок виконано самостійно)

Тепер розглянемо відмінності функціоналу. Наш застосунок і застосунок конкурента мають спільну мету – допомагати користувачам у досягненні цілей та розвитку навичок, проте підходи до реалізації дещо відрізняються.

Конкурент робить акцент на повній гейміфікації та візуальній мотивації: користувачі отримують віртуальні нагороди, предмети, а також можуть брати участь у групових випробуваннях. Натомість наш застосунок пропонує глибокий аналіз прогресу, індивідуальні функції та розширену систему трекінгу навичок, що більше підходить для користувачів, орієнтованих на аналітику й детальну

організацію.

Однією основних з переваг розроблюваного застосунку в порівнянні з конкурентом є наявність графіків і діаграм, які дозволяють чітко відслідковувати прогрес у розвитку навичок і досягненні цілей. Крім того, генерація звітів (щотижневих або щомісячних) дає змогу аналізувати витрачений час і коригувати плани.

Також відрізняється класифікація створюваних завдань. У Habitica вона здійснюється переважно за часовими критеріями, тоді як у розроблюваній системі передбачено класифікацію як за часом, так і за тематичними категоріями. Конкурентний застосунок акцентує увагу на мотиваційній складовій через віртуальні нагороди, інвентар та рівні, що створює ефект гейміфікації та заохочує користувачів до регулярного виконання завдань. Це може свідчити про наявність різного підходу до цільової аудиторії: розроблювана система орієнтована на більш структуровану персональну організацію часу та прогресу.

У частині соціальної взаємодії функціональність розроблюваної системи передбачає не лише базові можливості, притаманні подібним застосункам, але й спрямована на формування повноцінної мережі персональних контактів між користувачами. Реалізовано механізм надсилання та обробки запитів у друзі, перегляду профілів інших учасників, а також підтримки індивідуальних соціальних зв’язків. Кожен користувач має змогу формувати власне коло контактів, що сприяє персоналізованій взаємодії, обміну досвідом і взаємній підтримці в досягненні цілей.

Натомість у застосунку Habitica основний соціальний механізм реалізовано у вигляді групових викликів і гільдій (див. рис. 1.2), що орієнтовані на кооперативну взаємодію та створення командної мотивації. Однак така модель не передбачає побудови індивідуальних взаємин між користувачами, зокрема, відсутня функція додавання друзів або приватного обміну активністю.

Таким чином, запропонований у розроблюваній системі підхід має на меті не лише стимулювання користувача через зовнішні чинники, але й створення простору для сталих соціальних зв’язків, що базуються на довірі, виборі контактів

та індивідуальній комунікації. Це відрізняє проєкт від конкурентного рішення, у якому акцент зміщено на колективну участь та командну динаміку, залишаючи поза увагою аспекти персонального соціального середовища.

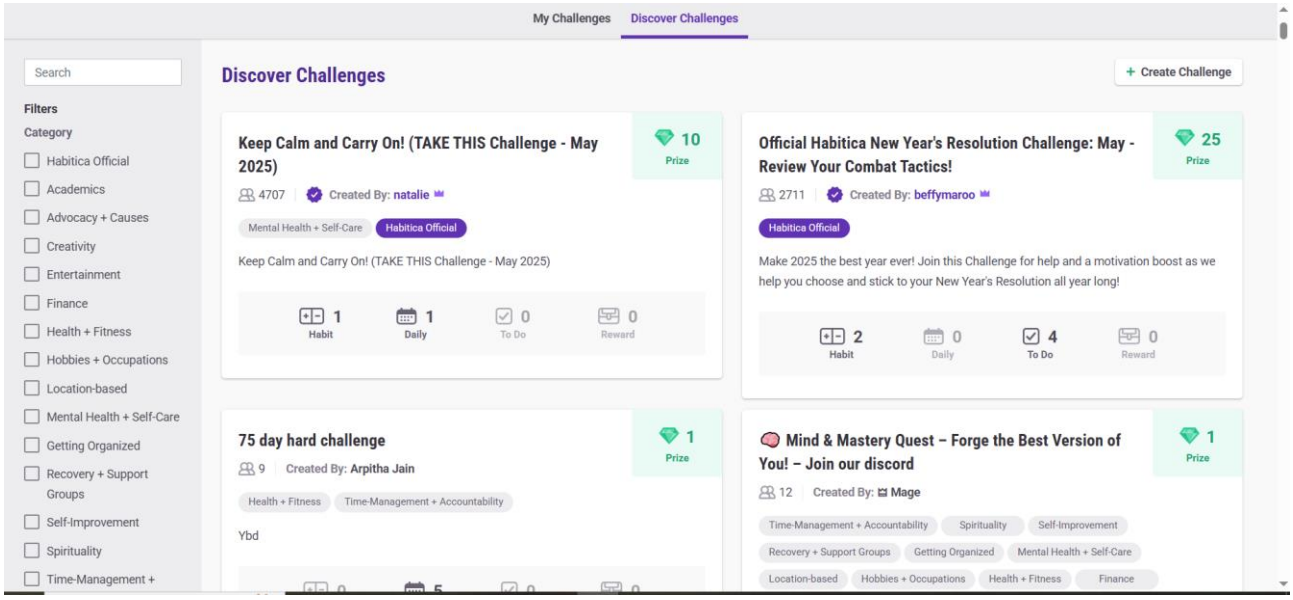

Рисунок 1.2 – Спільні випробовування (рисунок виконано самостійно)

Загалом можна сказати, що основна відмінність між розроблюваним застосунком та його конкурентом полягає в унікальності реалізованих функціональних можливостей і векторі взаємодії з користувачем. У той час як створювана система робить акцент на простоті, структурованості, зручності планування особистих завдань і досягненні цілей шляхом персоналізованої взаємодії, конкурентний додаток, зокрема Habitica, побудований навколо яскраво вираженої концепції гейміфікації повсякденного життя.

Гейміфікація у Habitica реалізована як розгалужена система ігрових механік, що створює у користувача враження занурення в рольову гру. Користувач має змогу збирати інвентар, отримувати й використовувати віртуальні предмети, такі як яйця, еліксири, обладунки, зброя тощо (див. рис. 2.3). Ці елементи не лише виконують декоративну чи символічну функцію, а й безпосередньо впливають на прогрес користувача, розширюючи можливості персоналізації персонажа, взаємодії у команді, виконання квестів і участі в боях проти віртуальних ворогів.

Крім того, система винагород у Habitica має гнучкий і глибокий характер: за

кожне виконане завдання користувач отримує досвід, золото, а іноді – випадкові призи у вигляді предметів. Такі стимули створюють високу залученість і підтримують мотивацію виконувати навіть рутинні дії. У результаті застосунок перетворює повсякденне планування на ігровий процес, який супроводжується почуттям досягнення та прогресу.

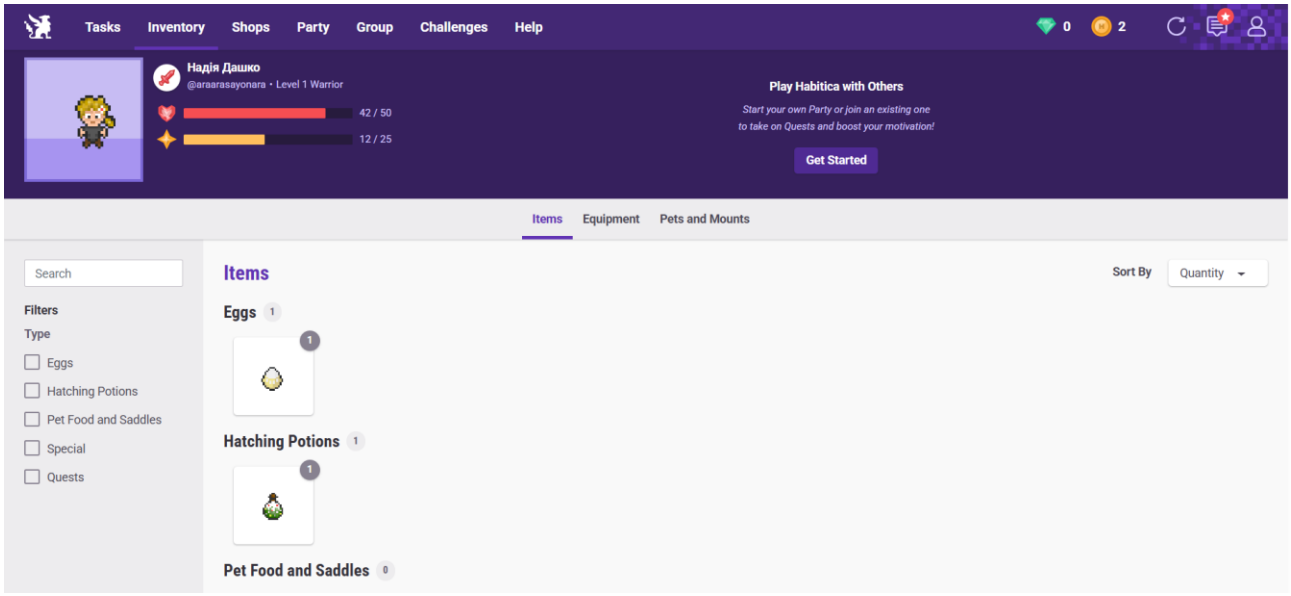

Рисунок 1.3 – Інвентар (рисунок виконано самостійно)

Крім того, одним із ключових елементів гейміфікації у Habitica є можливість кастомізації власного ігрового персонажа. Користувач має змогу змінювати зовнішній вигляд свого героя, обираючи стать, зачіску, колір шкіри, очей, одягу та інші візуальні характеристики, що дозволяє створити персоналізоване втілення себе у віртуальному середовищі. Згодом, у процесі досягнення прогресу, відкривається доступ до нових елементів кастомізації, включно з обладунками, зачісками, капелюхами, аксесуарами та навіть супутниками – так званими «pet», які супроводжують героя.

Функція налаштування персонажа відіграє не лише естетичну роль, а й формує емоційний зв’язок користувача з додатком, посилюючи ефект залученості до процесу самоменеджменту. Такий підхід сприяє формуванню стійкої звички використовувати додаток щоденно, адже навіть незначні досягнення винагороджуються можливістю змінити або покращити вигляд персонажа.

На рисунку 1.4 наведено приклад інтерфейсу, що демонструє візуальні можливості кастомізації та доступний набір параметрів, які користувач може змінювати відповідно до власних уподобань. Така деталізація гейміфікованого підходу значно відрізняє Habitica від класичних систем управління завданнями, де візуальна персоналізація зазвичай відсутня або мінімальна.

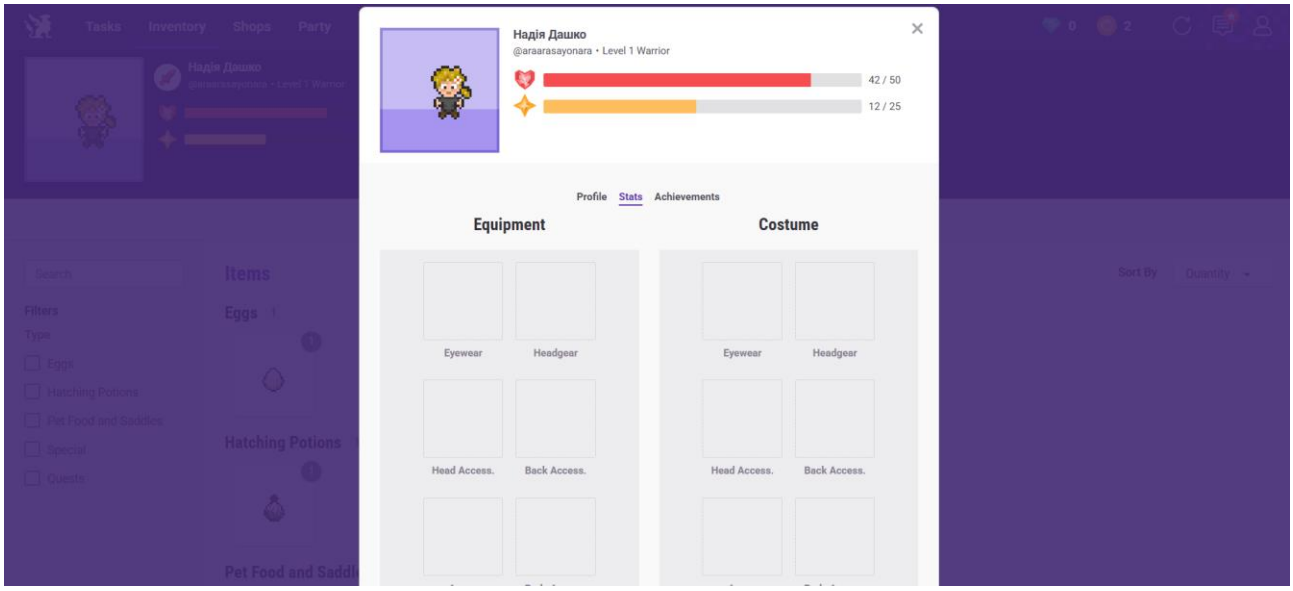

Рисунок 1.4 – Кастомізація персонажу (рисунок виконано самостійно)

Натомість розроблюваний застосунок пропонує наступні функції.

Першою є функція «Підтримати друга», яка додає емоційну складову до досягнення цілей. Ця особливість дозволяє користувачам отримувати мотиваційні нагадування у важливий момент, що може стати додатковим стимулом. У конкурента таких персоналізованих рішень немає, проте він компенсує це командними активностями та випробуваннями, де користувачі можуть по-іншому взаємодіяти зі спільнотою.

Додатковою перевагою застосунку є вбудований AI-асистент, який надає персоналізовані поради щодо розвитку навичок і досягнення цілей. Це важливий елемент для користувачів, які шукають конкретні рекомендації та підтримку. Конкурент, хоч і пропонує цікаві гейміфіковані рішення, але не має такого індивідуального підходу.

Наступним, а також одним із добре відомих і функціонально зрілих конкурентів розроблюваної системи є Trello – інструмент для управління

проєктами, заснований на методології канбан [3]. Канбан – це метод візуального управління завданнями, який походить із виробничої практики компанії Toyota, а згодом був адаптований для сфер управління проєктами та розробки програмного забезпечення. Основною ідеєю методу є візуалізація робочого процесу шляхом представлення задач у вигляді карток, розміщених на дошці, що поділена на колонки відповідно до стадій виконання (наприклад: «Заплановано», «У процесі», «Виконано»). Такий підхід дозволяє швидко оцінити стан завдань, обсяг поточного навантаження та виявити «вузькі місця» в робочому потоці. Канбан сприяє покращенню прозорості процесів, оптимізації розподілу ресурсів і підвищенню продуктивності як в індивідуальному, так і в командному використанні. Застосунок орієнтований переважно на командне використання, хоча має й достатньо можливостей для індивідуального планування. Основна парадигма роботи у Trello полягає в організації завдань у вигляді карток, які групуються за списками, що відображають етапи виконання. Завдяки простоті візуального представлення, система користується популярністю серед широкого кола користувачів – від студентів до команд розробників.

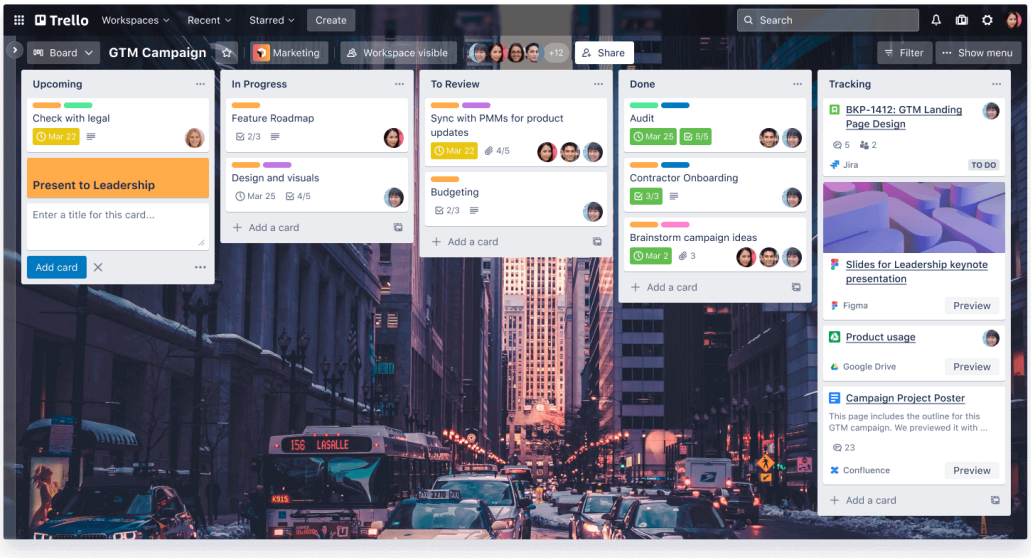

Рисунок 1.5 – Інтерфейс Trello (рисунок виконано самостійно)

На рівні клієнтської частини Trello реалізує інтерфейс з високим ступенем інтерактивності, де основний фокус зміщено на drag-and-drop механізми переміщення задач між списками, модальне редагування контенту та кастомізацію

кожної картки. UI побудований так, аби бути мінімалістичним, але функціональним: користувач має змогу додавати теги, дедлайни, вкладення та коментарі без складної навігації. Проте ця модель взаємодії фокусується на горизонтальному переміщенні задач у межах процесу, не надаючи достатнього акценту на глибинному відстеженні персонального прогресу чи аналітичних аспектах виконання завдань.

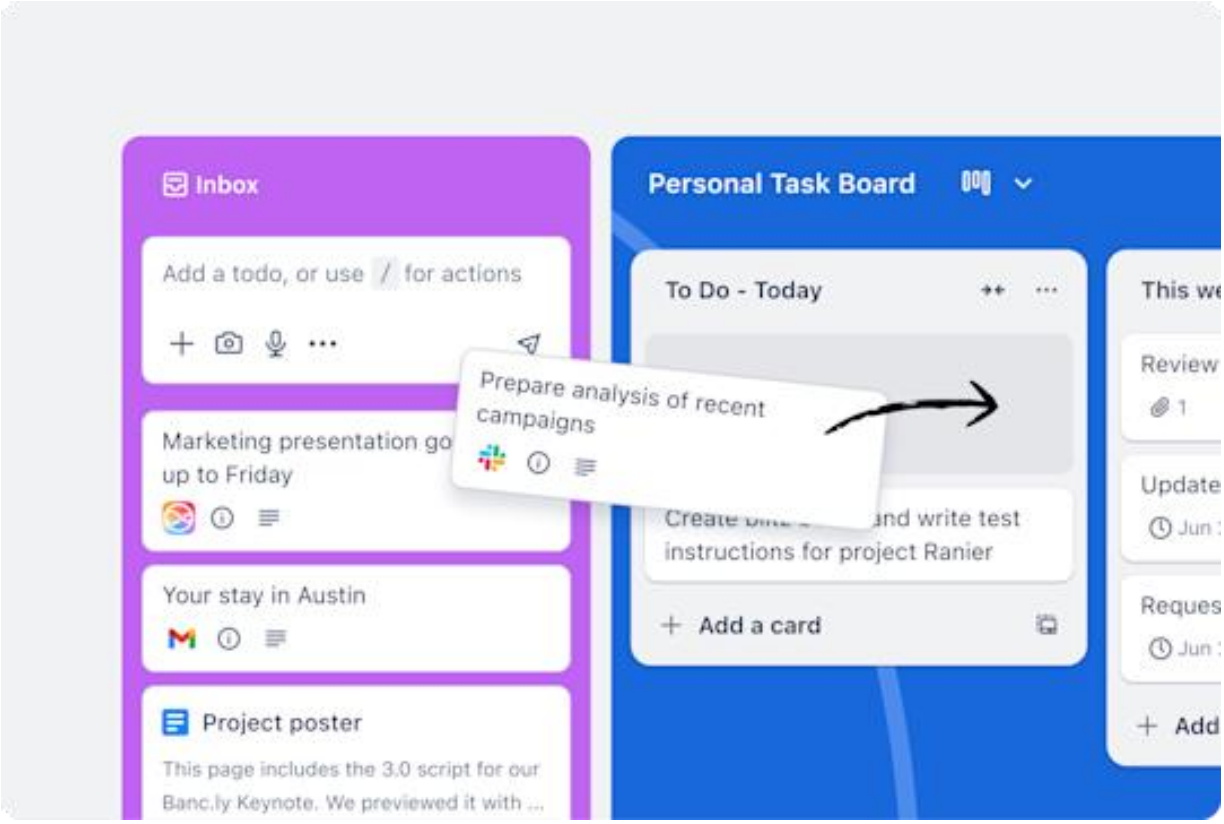

Рисунок 1.6 – Механізм drag-and-drop (рисунок виконано самостійно)

Розроблювана система, на відміну від Trello, орієнтована на комплексний моніторинг особистих цілей та досягнень із застосуванням гейміфікованого підходу. Інтерфейс користувача у цій системі реалізується як інструмент не лише для фіксації задач, але й для формування персоналізованої взаємодії завдяки візуальним індикаторам прогресу, аналітичним панелям і мотиваційним повідомленням. Зокрема, графіки прогресу за категоріями, візуальні індикатори виконання та накопичувані бали відображаються у спеціалізованих розділах особистого кабінету. Система рівнів, досягнень і рейтингових таблиць інтегрується безпосередньо в інтерфейс користувача, забезпечуючи додаткову мотивацію, на

відміну від Trello, де подібні механізми відсутні.

З погляду UX/UI, розроблювана система вирізняється вищим ступенем візуальної динаміки: реалізовано анімовані реакції інтерфейсу на дії користувача, адаптивне представлення аналітичної інформації та можливість налаштування візуального середовища відповідно до категорій завдань. Якщо у Trello інтерфейс зорієнтований на уніфіковану роботу з кількома дошками одночасно, то в аналізованій системі реалізується персоналізований підхід до організації життєвих задач, із можливістю гнучкої категоризації цілей за темами, пріоритетами та часовими межами.

Також варто зазначити, що Trello не містить функціональності мотиваційного чи соціального супроводу користувача. Його інтерфейс зберігає нейтральність щодо процесів виконання, не втручаючись у мотиваційні аспекти. Натомість розроблювана система включає механізм push-сповіщень з мотиваційними цитатами, можливістю отримання викликів та персоналізованих повідомлень, а також інструменти соціальної взаємодії, зокрема перегляд профілів інших користувачів і формування соціальних зв’язків. Таким чином, інтерфейс виконує не лише інформаційну, але й емоційну та підтримуючу функцію. Він створює відчуття комфорту під час взаємодії, сприяє формуванню позитивного користувацького досвіду та підвищує мотивацію до регулярного використання системи.

Узагальнюючи, можна відзначити, що хоча Trello є потужним засобом керування завданнями, його функціональна модель зорієнтована на загальну організацію робочих процесів, тоді як розроблювана система зосереджується на глибокій підтримці особистого розвитку, аналітики та мотиваційної динаміки користувача через функціонально насичений та візуально адаптивний клієнтський інтерфейс.

Розглянемо інший застосунок з контролю виконання задач ClickUp (див. рис. 2.7). ClickUp є одним із провідних інструментів на ринку управління проєктами та трекінгу завдань, з особливим акцентом на корпоративний та командний сегмент. Завдяки широкому функціоналу та високій універсальності, цей продукт здобув

популярність серед понад 8 мільйонів користувачів по всьому світу, що робить його одним із ключових гравців на ринку інструментів для продуктивності та планування завдань [4].

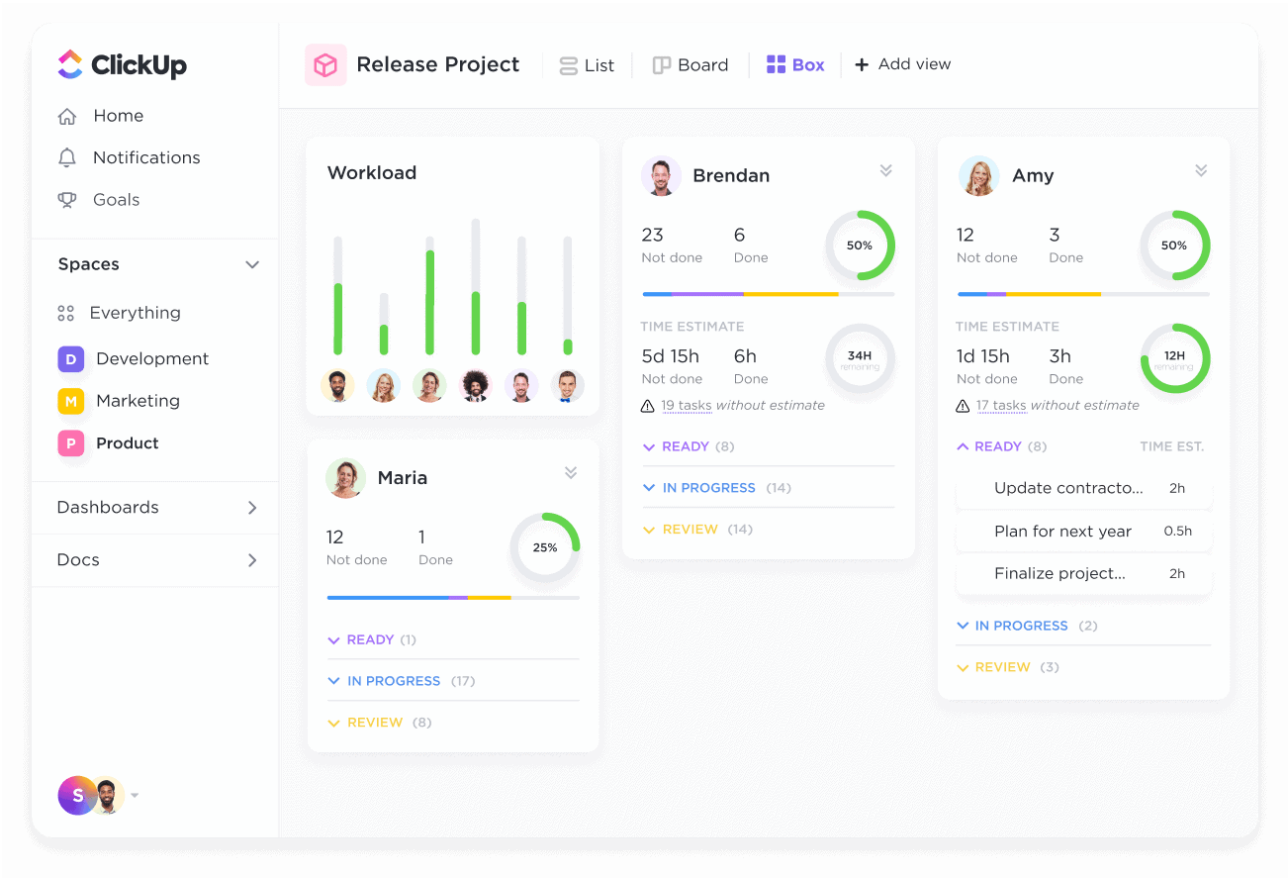

Рисунок 1.7 – Головна сторінка ClickUp (рисунок виконано самостійно)

Ринок управління цілями й завданнями за останні роки значно розвинувся, перетворюючись із простих інструментів на багатофункціональні платформи з налаштовуваними інтерфейсами. ClickUp успішно адаптувався до цих змін, додавши можливості для інтеграції завдань із цілями та спільною роботою, що особливо цінують бізнес-користувачі. Крім того, ClickUp активно монетизує свою платформу через систему підписок, пропонуючи різні рівні тарифів для різних категорій користувачів. Для корпоративних клієнтів компанія також надає масштабовані та безпечні рішення для управління великими проєктами.

Основні конкурентні переваги ClickUp включають: широкий набір функцій для управління цілями, завданнями та комунікацією в межах єдиної платформи, шаблони для різних підходів (SMART Goals, OKR), кастомізацію робочих

процесів, що приваблює як корпоративних клієнтів, так і окремих користувачів, а також функції сповіщень та нагадувань, які допомагають контролювати дедлайни. Платформа також інтегрується з популярними сервісами, такими як Google Drive, Jira, Figma, GitHub та іншими, що забезпечує зручність у використанні для різних типів команд.

ClickUp має кілька унікальних функцій, що виділяють його серед конкурентів. До таких належать мультифункціональність, яка поєднує цільовий трекінг, управління завданнями та аналітику в одній платформі, а також можливість адаптувати робочий процес до будь-якого стилю проєктного менеджменту (Kanban, Gantt, OKR). Інтерактивні віджети та Dashboards дозволяють користувачам аналізувати прогрес завдань через діаграми та візуалізації, а функції ClickUp Chat і обговорень завдань створюють інтегроване середовище для командної роботи.

Таким чином, ClickUp є універсальним інструментом для управління проєктами, орієнтованим на командну роботу, масштабування та структуризацію завдань. Платформа надає можливість налаштовувати робочі процеси, використовувати шаблони для досягнення цілей, а також здійснювати аналіз прогресу через інтерактивні панелі та візуальні звіти. Основний акцент робиться на вимірюванні продуктивності, залежностях завдань і співпраці всередині команд.

Розроблюваний додаток, на відміну від ClickUp, зосереджується на індивідуальному підході до досягнення цілей і розвитку навичок. Він орієнтований не на командну взаємодію, а на особисту ефективність, самоорганізацію та контроль власного прогресу. Застосунок пропонує емоційно-мотиваційні функції, такі як персоналізовані нагадування, віртуальні нагороди та підтримка через інтерфейс взаємодії з користувачем. Також реалізовано інструменти для глибокого аналізу прогресу, з можливістю побудови графіків, виведення статистичних показників та застосування AI-асистента для надання рекомендацій. У той час як ClickUp є багатофункціональним рішенням для командного менеджменту та організації спільної роботи. Тобто не орієнтоване на персоналізоване планування, детальний трекінг власної діяльності та підтримку мотивації.

1.3 Потреби клієнта або ринку

1.3.1 Опис потреб

Потреби клієнтів на ринку управління задачами та особистою продуктивністю зростають, і вони мають високі вимоги до функціональності таких систем. Клієнти прагнуть знайти інструменти, які дозволяють ефективно організовувати та виконувати задачі, мінімізуючи час на їх планування і відстеження. Вони шукають рішення, які зможуть допомогти виконанню рутинних операцій, даючи змогу зосередитися на важливіших для себе моментах.

Однією з основних потреб є можливість чітко відслідковувати виконання задач, своєчасно реагувати на їх невиконання або прострочення, а також автоматичне нагадування про важливі етапи або дедлайни. Клієнти шукають системи, які дозволяють не тільки ставити задачі, але й бачити прогрес у їх виконанні через зручні візуалізації – графіки, діаграми, інтерактивні інтерфейси. Це дозволяє користувачам ефективно планувати свій час і бачити, які задачі вимагають найбільше уваги.

Крім того, важливою вимогою є простота у використанні та адаптивність системи. Клієнти хочуть мати інструмент, який не потребує значних витрат часу на освоєння і дозволяє зручно налаштувати процеси під свої індивідуальні потреби. Вони шукають інтерфейси, які є інтуїтивно зрозумілими, що дозволяє швидко отримувати доступ до всіх функцій без необхідності навчання.

Мотиваційна складова також стає важливою частиною ринку. Клієнти хочуть, щоб системи не лише допомагали у плануванні, але й активно підтримували мотивацію до виконання завдань. Інструменти гейміфікації, такі як бали, рівні, досягнення, є важливою складовою для багатьох користувачів, які шукають додаткові стимули для постійного досягнення своїх цілей.

Враховуючи ці потреби, розроблюваний додаток має забезпечити інтеграцію функцій для ефективного управління задачами, надання аналітики виконання, а також включати мотиваційні елементи для підтримки користувачів у процесі досягнення їхніх цілей.

1.3.2 Цільова аудиторія

Цільова аудиторія розроблюваного додатка охоплює широке коло користувачів, які мають різні потреби та рівні досвіду в управлінні своїм часом та завданнями. Вона включає активних людей, що шукають ефективні інструменти для самоорганізації, зокрема професіоналів, студентів, підприємців та всіх, хто прагне покращити свою продуктивність і досягати особистих та професійних цілей.

Серед представників цієї аудиторії можна виділити дві основні категорії: новачки та досвідчені користувачі. Новачки – це люди, які вперше стикаються з подібними інструментами для планування та управління задачами. Вони можуть бути менш досвідченими в організації свого часу, тому потребують простих, інтуїтивно зрозумілих інтерфейсів і базових функцій для ефективного впорядкування своїх завдань. Для них важлива легкість у використанні додатка та наявність простих шаблонів для початку роботи.

Досвідчені користувачі, в свою чергу, мають сформовані підходи до організації та планування свого часу. Вони звикли до використання різноманітних методик самоорганізації, таких як списки завдань, трекери прогресу, методи планування та гейміфікації. Ці користувачі шукають систему, яка б надавала більше можливостей для налаштування та персоналізації робочого процесу. Вони зацікавлені у функціях, які дозволяють детальніше відслідковувати прогрес, аналізувати результати та налаштовувати інтерфейс під свої індивідуальні потреби.

Також важливу роль у цільовій аудиторії відіграють люди, які прагнуть розвивати нові навички або формувати корисні звички. Це можуть бути користувачі, які займаються самовдосконаленням, вивчають нові сфери діяльності або бажають змінити свої повсякденні звички. Для цієї групи важливою є наявність мотиваційних інструментів, які б підтримували їх на шляху до досягнення особистих цілей. Це можуть бути нагадування, візуальні індикатори прогресу, рівні, досягнення та інші елементи, що стимулюють до дії.

Крім того, ця програма буде корисна для тих, хто шукає баланс між особистим життям і роботою. Користувачі, які мають насичений графік і

потребують ефективних інструментів для управління завданнями, а також тих, хто прагне більш раціонально розподіляти свій час між різними активностями. Для цієї категорії користувачів важлива не тільки простота у використанні, але й наявність функцій для детального аналізу та відстеження часу, витраченого на виконання конкретних завдань.

У цілому, цільова аудиторія цього додатка різноманітна, і кожна група користувачів має свої специфічні вимоги та очікування щодо функціональності системи. Розроблюваний додаток має стати універсальним інструментом для людей різного віку та професій, котрі прагнуть підвищити свою продуктивність і ефективно управляти особистими задачами та досягненнями.

1.4 Виявлення та вирішення проблеми

Сучасний ритм життя, постійний потік інформації та необхідність виконання багатьох завдань одночасно створюють значне навантаження на користувачів, що часто призводить до неефективного використання часу та зниження продуктивності. Багато людей стикаються з проблемами, пов'язаними з організацією задач та відстеженням прогресу у їх виконанні. Зокрема, відсутність чіткої структури у розподілі завдань та недостатня видимість прогресу сприяють накопиченню завдань, що в свою чергу призводить до стресу та фрустрації. У результаті, користувачі не можуть ефективно реалізувати свої цілі, що знижує їхню мотивацію та продуктивність. Іншою проблемою є недостатня підтримка мотивації, що є особливо актуальним у разі довгострокових проєктів або складних задач. Без належного моніторингу прогресу та відсутності чітких орієнтирів для досягнення цілей, люди можуть втратити зацікавленість у процесі виконання завдань, що знижує їхню ефективність. Крім того, часто люди не мають інструментів для своєчасного виявлення та коригування помилок у процесі виконання, що призводить до затримок і зниження результативності.

Отже, основною проблемою є відсутність інтегрованих систем, які б не лише дозволяли організувати завдання, але й надавали б користувачеві інструменти для детального моніторингу прогресу, надавали персоналізовані рекомендації та

активно підтримували мотивацію. Більшість існуючих рішень зосереджена лише на управлінні завданнями, не надаючи достатньої підтримки в процесі досягнення цілей або у впровадженні змін у звичку користувачів. Це і є основною проблемою, яку має вирішити розроблюваний додаток.

Для вирішення вищезгаданих проблем, розроблюваний додаток пропонує інтегровану систему управління особистими задачами, яка враховує не лише планування, а й детальний моніторинг виконання завдань, забезпечує мотиваційний супровід та дає можливість зворотного зв'язку в реальному часі. Однією з основних складових вирішення є гнучке управління задачами. Додаток дозволяє користувачам створювати, редагувати та класифікувати задачі за різними критеріями, такими як терміни, пріоритети та категорії. Це дозволяє уникнути хаосу у списках завдань та забезпечує зручну структуру для їх виконання. Користувач може чітко бачити весь обсяг роботи та розподіляти завдання по днях, що допомагає зменшити стрес і підвищити ефективність роботи.

Іншою важливою складовою є візуалізація прогресу. Додаток використовує графіки та діаграми для відображення прогресу виконання завдань, що дає користувачам можливість оцінити, скільки завдань виконано, а скільки ще залишилось. Це дозволяє користувачам зрозуміти, на якому етапі вони знаходяться, і створює відчуття контролю та завершеності. Візуалізація також допомагає знизити рівень невизначеності, який часто є однією з причин відкладання важливих справ.

Ще однією важливою складовою є мотивація користувачів через гейміфікацію. Додаток інтегрує елементи гейміфікації, такі як бали, рівні, досягнення та таблиці лідерів. Це стимулює користувачів до регулярного виконання завдань і дає їм відчуття прогресу навіть при виконанні дрібних задач. Користувачі отримують віртуальні нагороди та рівні, що додає елемент змагання і знижує ризик втоми від повторення однотипних завдань. Відчуття досягнення та успіху важливе для збереження мотивації, і ці елементи забезпечують додаткову підтримку в процесі виконання завдань.

Для користувачів, які шукають індивідуальний підхід до виконання задач,

додаток пропонує AI-асистента, що надає персоналізовані рекомендації для організації робочого процесу. AI-асистент допомагає користувачам налаштувати пріоритети завдань, оптимізувати їхнє виконання та коригувати стратегії досягнення цілей. Він також підтримує користувачів у моменти, коли потрібна додаткова мотивація або поради щодо ефективного виконання задач.

Також важливою складовою є профілактика забування та пропуску термінів. Додаток автоматично генерує нагадування та сповіщення про наближення дедлайнів, що дає змогу користувачам своєчасно реагувати на будь-які зміни у планах. Це зменшує ймовірність виникнення затримок і сприяє своєчасному виконанню завдань. Завдяки цьому користувачі мають змогу підтримувати високий рівень організованості та відповідальності.

Розроблюваний додаток також включає можливість для соціальної взаємодії між користувачами. Це дає змогу додавати друзів, переглядати їхні досягнення та взаємодіяти у межах системи. Користувачі можуть брати участь у спільних викликах, обмінюватися досвідом і підтримувати один одного на шляху до досягнення цілей. Така соціальна складова додає елемент спільного розвитку і підтримки, що є додатковим стимулом до виконання завдань.

Загалом, розроблюваний додаток допомагає вирішити основні проблеми, пов'язані з неорганізованістю, відсутністю мотивації та недостатнім моніторингом прогресу, надаючи користувачам інструмент для ефективного управління завданнями, досягненнями та розвитку особистих навичок.

1.5 Постановка задачі

Основною задачею є розробка програмної системи, а саме створення ефективного інструменту для управління особистими задачами, який допоможе користувачам організувати свої завдання, відстежувати прогрес, зберігати мотивацію та досягати поставлених цілей. Застосунок має бути універсальним, простим у використанні та адаптованим під індивідуальні потреби користувачів різного віку, професій та рівня досвіду у використанні технологій.

Особливу ж увагу потрібно приділити клієнтській частині додатка, оскільки

саме вона визначатиме взаємодію користувача з системою. Завдання полягає в розробці інтуїтивно зрозумілого та адаптивного інтерфейсу, який дозволить користувачам легко орієнтуватися в додатку без потреби в додаткових інструкціях чи складному навчанні. Клієнтська частина має бути оптимізованим для різних браузерів, і забезпечувати швидку реакцію на дії користувача.

Першочерговою задачею є реалізація функцій для створення, редагування та категоризації завдань, щоб користувач міг чітко структурувати свою роботу, розподіляти задачі за пріоритетами та термінами. Усі ці функції мають бути представлені в простому й зручному інтерфейсі, який мінімізує час на введення та редагування даних. Крім того, важливо забезпечити відстеження прогресу виконання завдань, використовуючи візуалізацію через графіки, діаграми та інші інтерактивні елементи, що дозволять користувачам бачити їхні досягнення в реальному часі. Іншою важливою задачею є створення системи мотивації через гейміфікаційні елементи, такі як бали, рівні та досягнення. Це дозволить користувачам не лише організовувати свої задачі, а й отримувати стимул для регулярного виконання завдань. Додаток має надавати інтерактивний зворотний зв'язок за допомогою таких елементів, як сповіщення та нагадування про важливі етапи, дедлайни та терміни. Всі ці функції повинні бути представлені в динамічному і зрозумілому інтерфейсі, що мотивує користувача до постійного виконання завдань. Окрім цього, необхідно створити чат з персональним помічником у вигляді AI-асистента, що допомагатиме користувачам організовувати завдання в залежності від їхніх індивідуальних потреб та пріоритетів. Основною задачею є також забезпечення простоти у використанні системи та створення інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам без труднощів освоїти всі функції додатка та максимально ефективно ним користуватися.

У підсумку, задача полягає в тому, щоб розробити додаток, який буде поєднувати всі ці функції в зручному та доступному інтерфейсі, при цьому надаючи користувачам максимальну свободу в налаштуванні інтерфейсу та взаємодії з додатком. Цей підхід дозволить зробити систему максимально

адаптованою до потреб різних категорій користувачів, підвищуючи їхню продуктивність та мотивацію до виконання задач. Важливо, щоб інтерфейс був гнучким і масштабованим, а користувачі мали змогу змінювати структуру та вигляд елементів відповідно до власних уподобань. Крім того, система має забезпечувати інтуїтивну навігацію, швидкий доступ до ключових функцій і підтримку на різних пристроях. Усе це сприятиме формуванню стабільної звички до використання додатку та підвищенню його ефективності в довготривалій перспективі.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Постановка мети

Метою даної кваліфікаційної роботи є розробка клієнтської частини програмної системи для планування та моніторингу виконання особистих задач і досягнень.

Застосунок має забезпечити користувачів ефективними інструментами для організації їхніх завдань, відстеження прогресу і досягнення поставлених цілей. Система повинна бути адаптована до індивідуальних потреб користувачів, що мають різні рівні досвіду в управлінні часом та завданнями. Основна мета полягає в створенні інтуїтивно зрозумілого та зручного інтерфейсу, який дозволяє користувачам швидко та ефективно взаємодіяти з додатком. Функціональність системи повинна включати можливість створення, редагування та категоризації завдань, візуалізацію прогресу, систему нагадувань і сповіщень, а також мотиваційні елементи, такі як: бали, рівні та досягнення. Клієнтська частина повинна бути оптимізована для роботи на різних пристроях та платформах, забезпечуючи швидку реакцію і адаптацію до потреб користувачів [5].

2.2 Загальний опис

У межах даної кваліфікаційної роботи реалізується клієнтська частина веборієнтованої програмної системи, призначеної для організації, планування та контролю виконання особистих задач і досягнень користувача. Клієнтська частина взаємодіятиме з сервером, отримуючи необхідні дані та передаючи їх для відображення в інтерфейсі користувача.

Мають бути реалізовані наступні функції:

а) Реєстрація та авторизація користувачів:

1) Користувачі матимуть можливість зареєструватися та авторизуватися у системі. Всі дані користувачів, такі як ім’я, електронна пошта та паролі, будуть зберігатися на сервері.
2) Клієнтська частина взаємодіятиме з сервером для перевірки облікових

даних користувачів, що дозволить здійснити авторизацію та доступ до персонального контенту.

б) Профіль користувача:
   1) Клієнтська частина надасть користувачам можливість редагувати свої персональні дані, такі як ім’я, фото профілю тощо. Всі зміни зберігатимуться на сервері і будуть доступні для користувача після авторизації.
   2) Користувачі також зможуть переглядати свої досягнення та статистику виконаних завдань, які будуть отримуватися з серверу в реальному часі для відображення на інтерфейсі.

в) Управління задачами:
   1) Клієнтська частина дозволить користувачам створювати, редагувати та категоризувати завдання. Усі зміни будуть зберігатися на сервері.
   2) Дані про завдання, їх категорії, терміни виконання будуть передаватися на сервер для збереження та подальшого відображення у клієнтському інтерфейсі.
   3) Користувачі зможуть працювати з завданнями на клієнтській стороні, а дані синхронізуватимуться з сервером для збереження і подальшої обробки.

г) Аналітика та відстеження прогресу:
   1) Прогрес користувачів щодо виконання завдань буде візуалізуватися через графіки та діаграми.
   2) Клієнтська частина буде запитувати дані про виконані завдання з сервера та на основі цих даних генерувати відповідні візуалізації. Це дозволить користувачам відстежувати свої досягнення в реальному часі.

д) Гейміфікація:
   1) Всі елементи гейміфікації, такі як бали, рівні, досягнення та таблиці лідерів, будуть зберігатися на сервері. Клієнтська частина буде взаємодіяти з сервером для отримання актуальних даних про

досягнення користувачів та їх позицію в таблиці лідерів, а також для оновлення рівнів та нагород.

е) Push-сповіщення:

1) Сповіщення, що надходять користувачам про дедлайни, мотиваційні цитати та виклики від друзів, будуть генеруватися на сервері та доставлятися на клієнтську частину для відображення. Ці сповіщення будуть синхронізовані з даними на сервері, щоб забезпечити своєчасність і точність повідомлень.

ж) Соціальна взаємодія:

1) Клієнтська частина дозволить користувачам переглядати акаунти інших, додавати та видаляти друзів. Всі ці дії будуть зберігатися на сервері, що дозволить синхронізувати соціальну активність користувачів між різними пристроями.

Усі зазначені функціональні можливості мають бути реалізовані у межах клієнтської частини програмного додатка, яка відіграє ключову роль у забезпеченні взаємодії між користувачем та системою. Саме клієнтська частина відповідає за побудову, відображення та оновлення інтерфейсу, через який здійснюється доступ до основних можливостей: перегляду профілю, керування задачами, відстеження прогресу, взаємодії з досягненнями та обробки повідомлень. Вона функціонуватиме як динамічний фронтенд, що отримує необхідні дані від серверної частини шляхом надсилання HTTP-запитів до API. Такий підхід дозволить забезпечити актуальність і цілісність даних, а також ефективну синхронізацію інформації між пристроями.

У результаті, незалежно від того, на якому пристрої здійснюється вхід – персональному комп’ютері, планшеті чи мобільному телефоні – користувач матиме постійний доступ до персоналізованого контенту, включаючи задачі, статистику, досягнення, повідомлення та інші інтерактивні елементи. Забезпечення стабільної та узгодженої роботи клієнтської частини гарантує, що зміни, внесені користувачем, будуть коректно збережені, а інтерфейс завжди відображатиме актуальний стан даних, що є критично важливим для підтримання довіри до

системи та її ефективного використання.

2.3 Загальні обмеження

Клієнтська частина програмної системи реалізована на мові програмування JavaScript [6]. Використовується бібліотека React (v19) для створення компонентного інтерфейсу та Vite як інструменту для збірки й розробки [7]. Архітектура застосунку орієнтована на одно-сторінкову структуру (SPA), із використанням React Router для навігації між маршрутами. Для інтеграції з обліковими записами Google застосовується бібліотека @react-oauth/google, яка забезпечує базовий механізм автентифікації.

Застосунок розгортається у середовищі браузера і потребує стабільного інтернет-з'єднання для отримання та надсилання даних до серверної частини через HTTP API-запити. Уся бізнес-логіка, пов’язана зі збереженням, обробкою й аналітикою даних, виконується на сервері, тоді як клієнтська частина відповідає за візуалізацію, взаємодію з користувачем та ініціацію запитів до API.

Система підтримується на сучасних браузерах, зокрема:

- Google Chrome (версії не старші за дві актуальні);
- Mozilla Firefox;
- Microsoft Edge;
- Safari.

Мінімальні вимоги до клієнтського середовища:

- Операційна система: Windows 10/11, macOS 11+, Linux (із підтримкою сучасних браузерів);
- Процесор: 2-ядерний, не нижче 2.0 GHz;
- Оперативна пам’ять: від 4 GB;
- Екран: мінімальна роздільна здатність 1024×768;
- Інтернет-з’єднання: не менше 5 Мбіт/с.

Поза онлайном застосунок працювати не може, оскільки всі динамічні дані завантажуються з серверної частини, а локальне кешування реалізоване лише для

технічних цілей. Відсутність повноцінної офлайн-підтримки зумовлює залежність функціоналу від постійного доступу до мережі. У випадку втрати з'єднання користувач позбавлений можливості взаємодіяти з основними розділами інтерфейсу, що вимагає передбачення відповідних повідомлень про помилки та механізмів повторної синхронізації після відновлення доступу до сервера.

2.4 Припущення та залежності

Під час розробки клієнтської частини програмної системи зроблено низку припущень, які впливають на логіку функціонування додатка та його взаємодію з серверною частиною.

Передбачається, що серверна частина уже реалізована або буде реалізована у відповідності до REST-архітектури з доступом до основних ресурсів (користувачі, завдання, досягнення, сповіщення тощо) через стандартизовані HTTP-запити. Клієнтська частина функціонує як інтерфейс для обробки та відображення цих даних, і без доступного API її повноцінна робота є неможливою.

Очікується, що сервер надає:

- стабільний та захищений API для аутентифікації користувача (зокрема підтримку OAuth 2.0 для Google-авторизації);
- доступ до бази даних із задачами, категоріями, статистикою, досягненнями;
- механізм обробки push-сповіщень і взаємодії між користувачами (друзі, виклики, таблиця лідерів).

Передбачається, що всі відповіді серверу надходять у форматі JSON, а авторизація користувачів реалізована через токенну модель (наприклад, з використанням JWT), яку клієнтська частина зберігає в локальному сховищі або оперативній пам’яті.

Крім того, існує залежність від зовнішніх бібліотек, зокрема:

- React – для побудови інтерфейсу користувача;
- React Router DOM – для маршрутизації;
- @react-oauth/google – для авторизації через Google;

- Axios або Fetch API – для взаємодії з сервером;
- Recharts або аналог – для візуалізації статистики;
- CSS або UI-бібліотеки – для реалізації адаптивного і зручного дизайну.

Також припускається, що користувач має сучасний браузер, який підтримує ES6+ та актуальні веб-API, включно з fetch, localStorage.

Таким чином, успішна робота клієнтської частини залежить від наявності стабільного інтернет-з'єднання, коректної роботи серверної частини, актуальності зовнішніх бібліотек та стандартів веббраузерів.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML-проєктування

Клієнтська частина системи орієнтована на взаємодію кінцевого користувача з ключовими функціями програмного забезпечення. Для відображення функціональності з точки зору користувача доцільно використати діаграму прецедентів, яка ілюструє основні сценарії взаємодії.

Користувач системи має змогу виконувати такі дії:

- зареєструвати новий обліковий запис та здійснити вхід;
- створювати, редагувати, переглядати й категоризувати особисті задачі;
- відслідковувати прогрес виконання задач та переглядати статистику;
- переглядати таблицю лідерів і власні досягнення;
- отримувати push-сповіщення про дедлайни, мотиваційні повідомлення або виклики;
- взаємодіяти з іншими користувачами – додавати друзів, переглядати їхні профілі.

На рисунках 3.1 та 3.2 наведені діаграми прецедентів, які відображають основні випадки використання для неавторизованого та авторизованого користувача відповідно:

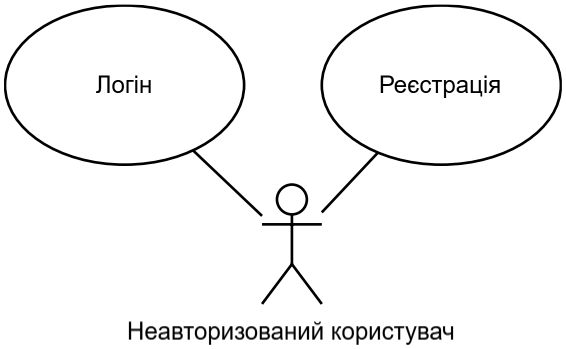

Рисунок 3.1 – Діаграма прецедентів для неавторизованого користувача (рисунок виконано самостійно)

Перша діаграма демонструє базові можливості, доступні без входу в систему, зокрема перегляд сторінки авторизації, реєстрації, а також можливість входу через

обліковий запис Google (що також відноситься до логіну). Друга діаграма охоплює ширший спектр дій, які стають доступними після успішної автентифікації: створення, редагування та видалення задач, перегляд статистики, керування друзями, надсилання запитів, а також доступ до персоналізованого профілю користувача.

Рисунок 3.2 – Діаграма прецедентів для авторизованого користувача (рисунок виконано самостійно)

Зображені діаграми дозволяють систематизувати функціональність відповідно до рівня доступу користувача, що є важливим на етапі подальшого моделювання логіки взаємодії та реалізації інтерфейсних рішень. Прецеденти охоплюють усі ключові можливості, передбачені технічним завданням, і демонструють розмежування функцій за категоріями – від управління задачами до

соціальної взаємодії. На їх основі можливо сформувати окремі модулі застосунку, визначити зв’язки між компонентами та забезпечити узгодженість під час реалізації. Діаграма прецедентів також слугує основою для формування технічної документації та є орієнтиром для майбутньої розробки інтерфейсів.

3.2 Проєктування архітектури

Розпочнемо проєктування з обрання архітектурної моделі для клієнтської частини програмної системи Naviria. Було обрано архітектурну модель, засновану на принципах одно-сторінкового застосунку (SPA) з використанням бібліотеки React. Такий підхід дає змогу реалізувати чітку логічну сегментацію інтерфейсу на незалежні компоненти, що взаємодіють через маршрутизацію, стан та обробку подій. Ключовим проєктним рішенням стало поділ клієнтської частини на функціональні області:

- Навігація та інтерфейсна оболонка;
- Аутентифікація;
- Контентні модулі;
- Взаємодія з API (через сервіси, із суворим поділом запитів/відповідей);
- Управління станом (через локальні хуки та контекст, із можливістю масштабування до глобального state manager).

Кожна з функціональних частин передбачає окрему відповідальність, що відповідає принципу Single Responsibility. Цей принцип полягає в тому, що кожен компонент або модуль має виконувати лише одну чітко визначену функцію та змінюватися лише з однієї причини – якщо змінюються вимоги до цієї функції. У контексті клієнтської частини це означає, що, наприклад, UI-компоненти повинні відповідати виключно за візуальне відображення та взаємодію з користувачем, а логіка запитів до API або обробка помилок мають бути винесені в окремі сервіси. Такий підхід покращує читабельність коду, спрощує тестування, зменшує зв’язаність частин системи та забезпечує кращу підтримку в довгостроковій перспективі.

Взаємодія з сервером у процесі проєктування передбачає роботу через уніфіковану точку обміну – API-клієнт, винесений в окремий модуль, що ізолює логіку запитів від UI-рівня. Цей підхід відповідає принципам чистої архітектури.

Власне проєктування було орієнтоване на забезпечення реактивності інтерфейсу, що реалізується через керування станом (useState/useEffect) та динамічне рендерування даних. Крім того, було враховано адаптивність інтерфейсу, що закладається в систему стилів і структурне розбиття верстки.

Для глибшого розуміння логіки роботи клієнтської частини програмної системи доцільно створити діаграми активності, які демонструють послідовність дій користувача на деяких основних сторінках вебдодатка. Такі діаграми дозволяють візуалізувати типові сценарії взаємодії з інтерфейсом, відображаючи переходи між станами, виклики подій та обробку відповідей. Це сприяє формуванню цілісного уявлення про поведінку системи на рівні користувацького досвіду та полегшує проєктування інтерфейсної логіки.

Розпочнемо огляд із діаграми активності для логіну та реєстрації (див. рис. 3.3).

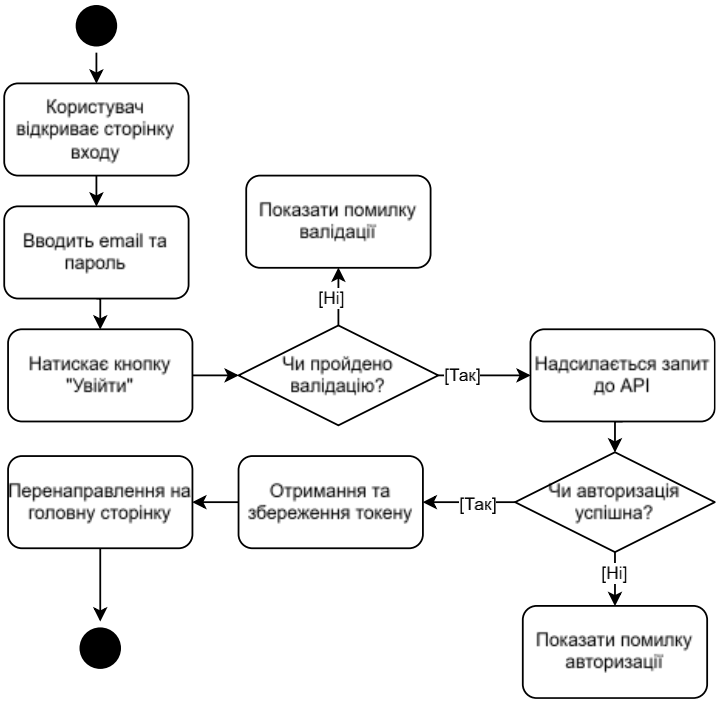

Рисунок 3.3 – Діаграма активності: логін та реєстрація (рисунок виконано самостійно)

Ця діаграма активності демонструє загальну логіку обробки форми входу або реєстрації у клієнтській частині застосунку. Вона охоплює послідовність дій, які відбуваються з моменту відкриття відповідної сторінки до завершення процесу авторизації. Основні етапи процесу:

1. Ініціалізація – користувач відкриває відповідну сторінку форми входу або реєстрації, після чого відображається інтерфейс для введення даних;
2. Введення даних – користувач заповнює обов'язкові поля, серед яких можуть бути email, пароль, підтвердження пароля, нікнейм, стать, дата народження тощо (залежно від типу форми);
3. Валідація на клієнті – перевіряється правильність і повнота введених даних:
   - якщо дані некоректні (наприклад, порожні поля, неправильний формат email чи слабкий пароль), користувач бачить відповідне повідомлення про помилку;
   - якщо валідація проходить успішно, формується запит на сервер через API;
4. Надсилання запиту до API – на цьому етапі відбувається обмін даними з серверною частиною:
   - у разі помилки (неправильний пароль, невірний email, вже зареєстрований обліковий запис тощо) користувач отримує повідомлення про помилку входу або реєстрації;
   - якщо авторизація або реєстрація проходить успішно, сервер повертає токен, який система зберігає у локальному сховищі браузера (наприклад, localStorage);
5. Завершення – після збереження токена виконується автоматичне перенаправлення користувача на головну сторінку застосунку або інший попередньо визначений маршрут (наприклад, особистий кабінет).

Цей процес є універсальним для обох форм – як входу, так і реєстрації, із незначними відмінностями у перевірці на стороні серверної частини. Діаграма активності відображає логіку дій, орієнтовану на швидкий зворотний зв'язок із користувачем та мінімізацію очікувань.

Перейдемо до огляду діаграми активності для створення задач(див. рис. 3.4). Ця діаграма відображає послідовність дій користувача та системи, починаючи з ініціації процесу створення нової задачі. Наочно показано, як здійснюється заповнення обов’язкових полів форми, перевірка введених даних, взаємодія з API та обробка відповіді сервера. Завершенням процесу є або успішне збереження задачі у системі, або виведення повідомлення про помилку у разі некоректного введення або збоїв під час запиту.

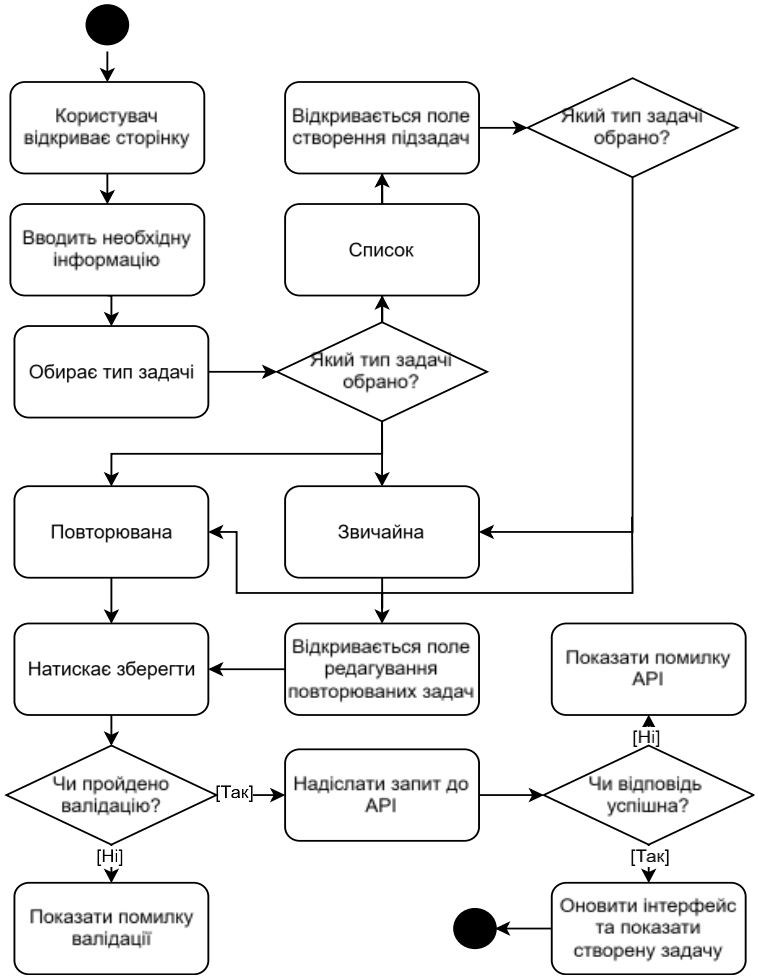

Рисунок 3.4 – Діаграма активності: створення задачі (рисунок виконано самостійно)

Наведена діаграма активності відображає повну послідовність дій користувача під час створення нової задачі у клієнтській частині програмного забезпечення. Сценарій починається з відкриття відповідного розділу, де користувач ініціює процес створення, натискаючи кнопку «Створити нову задачу».

Далі виконується заповнення обов’язкових та додаткових полів: назва задачі, опис, категорії, дата, час, рівень пріоритету, а також вказуються додаткові параметри, такі як відображення прогресу чи наявність нагородження. Ключовим етапом є вибір типу задачі. Якщо обрано тип «Звичайна» або «Повторювана», користувач завершує заповнення та переходить до збереження задачі. У випадку вибору типу «Список» відкривається додатковий інтерфейс для створення підзадач. При цьому для кожної підзадачі користувач може окремо задати назву, дату, час, пріоритет, а також тип задачі – але лише між варіантами «Звичайна» або «Повторювана». Тип «Список» у підзадачах не допускається, що обмежує вкладеність лише до двох рівнів.

Після завершення заповнення форми та підзадач (якщо такі є), користувач натискає кнопку збереження. У цей момент запускається механізм клієнтської валідації. У разі виявлення помилок користувач отримує відповідні повідомлення, які зупиняють виконання сценарію. Якщо валідацію пройдено успішно, формується запит до API серверної частини. У відповідь система або підтверджує створення задачі, або повідомляє про помилку (наприклад, у разі проблеми з сервером). У разі успішного збереження задачі, інтерфейс оновлюється, і нова задача з’являється в активному списку користувача.

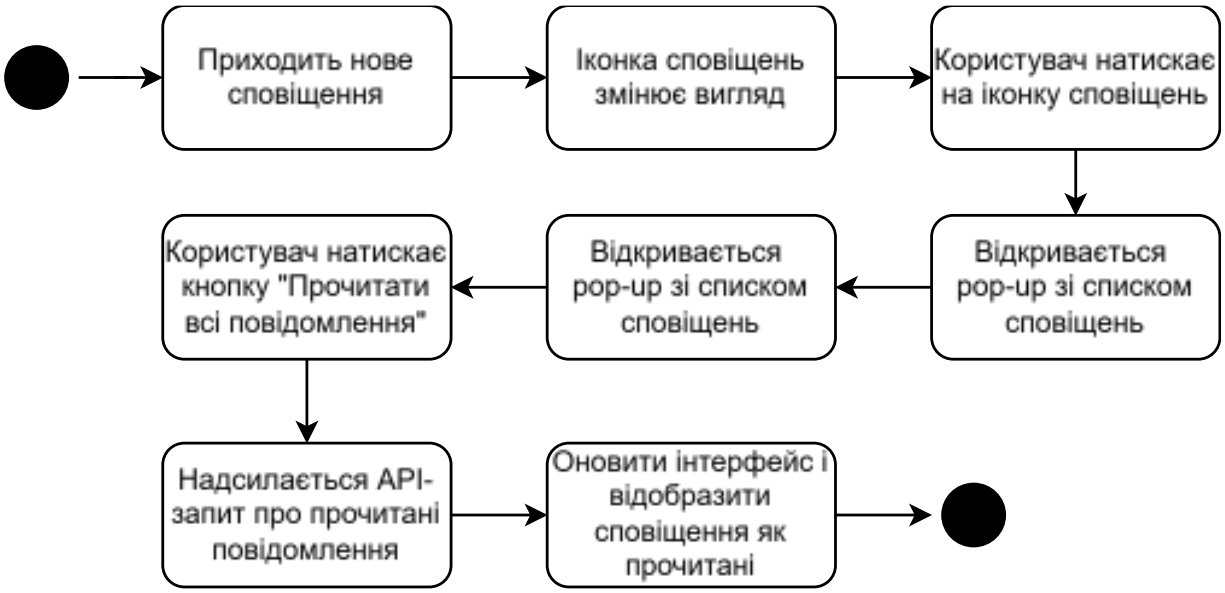

Рисунок 3.5 – Діаграма активності: отримання push-сповіщень (рисунок виконано самостійно)

На діаграмі можна побачити, що після надходження сповіщення користувач відкриває pop-up вікно, в якому відображається список повідомлень. Усі сповіщення поділяються на нові та прочитані, що реалізовано через різні візуальні стилі, наприклад, зміну кольору або акцентного маркування.

У межах інтерфейсу користувач має змогу переглядати вміст повідомлень, а також натиснути кнопку «Прочитати всі», що активує запит до серверного API з метою оновлення статусу повідомлень у базі даних. У відповідь клієнтська частина отримує підтвердження, після чого інтерфейс оновлюється, і всі повідомлення позначаються як прочитані.

3.3 UI/UX-проєктування інтерфейсу

UI/UX-проєктування є одним із ключових етапів розробки клієнтської частини програмного забезпечення, оскільки саме від нього залежить зручність, доступність та ефективність взаємодії користувача із системою. Грамотно спроєктований інтерфейс дозволяє забезпечити інтуїтивну навігацію, логічне розташування елементів, візуальну послідовність та відповідність очікуванням користувачів [8]. У межах цієї роботи особливу увагу приділялося створенню адаптивного інтерфейсу, що відповідає сучасним принципам дизайну та забезпечує зручну роботу на різних типах пристроїв.

Розробка UX-рішень базувалася на аналізі типових сценаріїв використання, а також передбачала врахування поведінкових шаблонів користувачів, зокрема мінімізацію кількості дій для досягнення цілей, оптимізацію форм введення та надання швидкого зворотного зв’язку.

У межах UI/UX-проєктування було створено низку каркасних макетів майбутніх сторінок, що відображають логіку інтерфейсу, структуру розміщення елементів та сценарії взаємодії користувача із системою. Макети охоплюють ключові інтерфейси, зокрема сторінки входу, реєстрації, управління задачами, перегляду статистики та сповіщень. Вони були виконані у стилі lo-fi wireframe, що дозволяє зосередитися на функціональній структурі без деталізації графічного оформлення. Такий підхід дав змогу на ранньому етапі виявити потенційні UX-

проблеми, забезпечити зручність навігації, візуальну послідовність та підготувати основу для подальшого візуального дизайну клієнтської частини. Крім того, каркасні схеми стали зручним інструментом для обговорення рішень у команді, спрощуючи узгодження функціональних вимог між розробниками, дизайнерами та потенційними користувачами. Це дозволило досягти більшої відповідності між очікуваннями цільової аудиторії та реалізованим інтерфейсом.

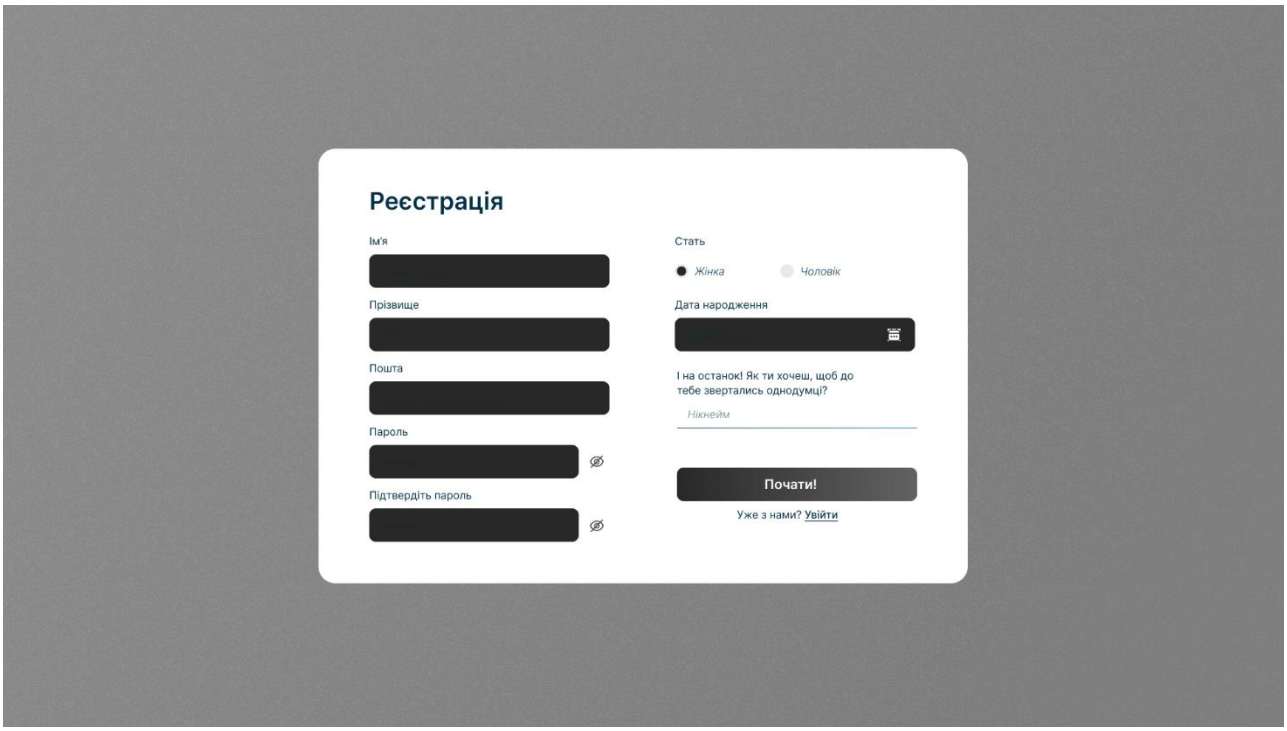

Рисунок 3.6 – Макет сторінки реєстрації (рисунок виконано самостійно)

На рисунку можемо побачити каркасний макет (wireframe) інтерфейсу сторінки реєстрації користувача, який є складовою клієнтської частини програмної системи. Даний макет демонструє структуру основних елементів форми, необхідної для первинного збору персональних даних користувача при створенні облікового запису.

Інтерфейс включає стандартний набір полів введення: ім’я, прізвище, адреса електронної пошти, пароль та підтвердження пароля. Також передбачено вибір статі шляхом перемикачів (radio-buttons), вказання дати народження через відповідний інтерфейсний компонент (date picker), а також необов’язкове текстове поле для введення імені або нікнейму, яким користувач бажає, щоб до нього

зверталися в системі. У нижній частині форми розміщено кнопку дії з закликом до реєстрації, а також додаткове текстове посилання для переходу до форми входу, якщо користувач вже має обліковий запис.

Макет виконано з дотриманням принципів візуальної ієрархії, симетрії та адаптивності з метою забезпечення інтуїтивної взаємодії користувача з формою реєстрації.

Принцип візуальної ієрархії реалізовано через логічне упорядкування елементів згори донизу відповідно до природного сценарію заповнення форми: спочатку основні персональні дані (ім’я, прізвище, пошта), потім автентифікаційна інформація (пароль), далі – додаткові атрибути (стать, дата народження, нікнейм). Заголовок форми та кнопка дії виділені більшим шрифтом і розташовані на відповідних стратегічних позиціях, що привертає увагу користувача.

Принцип симетрії втілено у рівномірному розміщенні полів по вертикальній осі, з однаковими відступами, що формує візуальний порядок і спрощує сприйняття інтерфейсу. Вирівнювання елементів по одній вісі створює баланс, зменшуючи когнітивне навантаження на користувача.

Принцип адаптивності реалізовано за рахунок компактної й блокової структури інтерфейсу, що дозволяє масштабувати макет під різні типи пристроїв (наприклад, елементи розташовані у єдиній колонці, з урахуванням мобільної зручності). Таке рішення гарантує однакову функціональність і збереження логіки подання інформації незалежно від розміру екрана.

Усі інші макети в межах кваліфікаційної роботи побудовані за аналогічними принципами – візуальної ієрархії, симетрії та адаптивності – із дотриманням узгодженого стилю розміщення елементів, послідовності взаємодії та структури інтерфейсу. Такий підхід дозволив забезпечити цілісність дизайну та легкість сприйняття інформації незалежно від конкретної сторінки або функції. Враховуючи повторюваність дизайнерських підходів у побудові всіх макетів, подальший детальний опис кожного з них не є необхідним і не наводиться окремо.

Перейдемо до огляду наступного макету, що зображений на рисунку 3.7. Даний макет призначений для забезпечення авторизації користувачів шляхом

введення облікових даних або за допомогою стороннього сервісу – зокрема, Google-акаунта. Макет також побудований із дотриманням ключових принципів UI/UX-дизайну: використано вертикальне вирівнювання елементів, достатні відступи між блоками, логічну ієрархію подання інформації та акцентування дій за допомогою стилізованих кнопок.

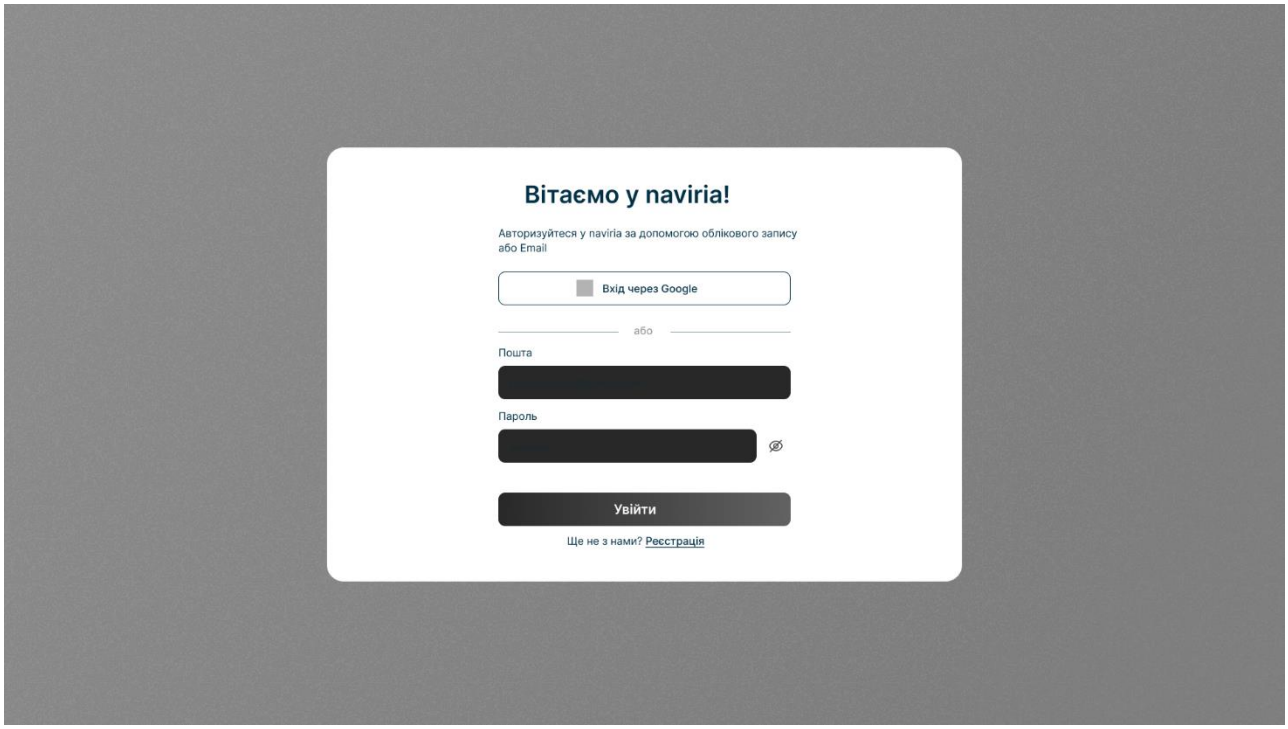

Рисунок 3.7 – Макет сторінки логіну (рисунок виконано самостійно)

Центральним елементом є поле входу через Google, що візуально виділене в окремий блок і супроводжується альтернативним варіантом – автентифікація за електронною поштою та паролем. Така структура дозволяє швидко обрати зручний спосіб входу, підвищуючи загальну доступність інтерфейсу. У нижній частині форми реалізовано кнопку входу «Увійти», а також гіперпосилання для переходу на сторінку реєстрації у випадку, якщо користувач ще не має облікового запису. Обидва елементи мають контрастне візуальне оформлення, що спрощує навігацію та сприяє швидкому залученню користувача до подальшої взаємодії із системою.

Варто зазначити, що структура даної форми зберігає візуальну та функціональну узгодженість із макетом сторінки реєстрації, наведеним раніше. Зокрема, спільними є типографіка, розміщення основних елементів, стилізація

полів і кнопок, що забезпечує цілісність інтерфейсу та покращує користувацький досвід за рахунок послідовного візуального стилю. Така узгодженість у проєктуванні сприяє зниженню когнітивного навантаження на користувача та прискорює адаптацію до системи.

Перейдемо до наступного макету. На рисунку 3.8 зображено каркасний макет інтерфейсу розділу «Персональний помічник», який виконує функцію чату між користувачем і віртуальним асистентом.

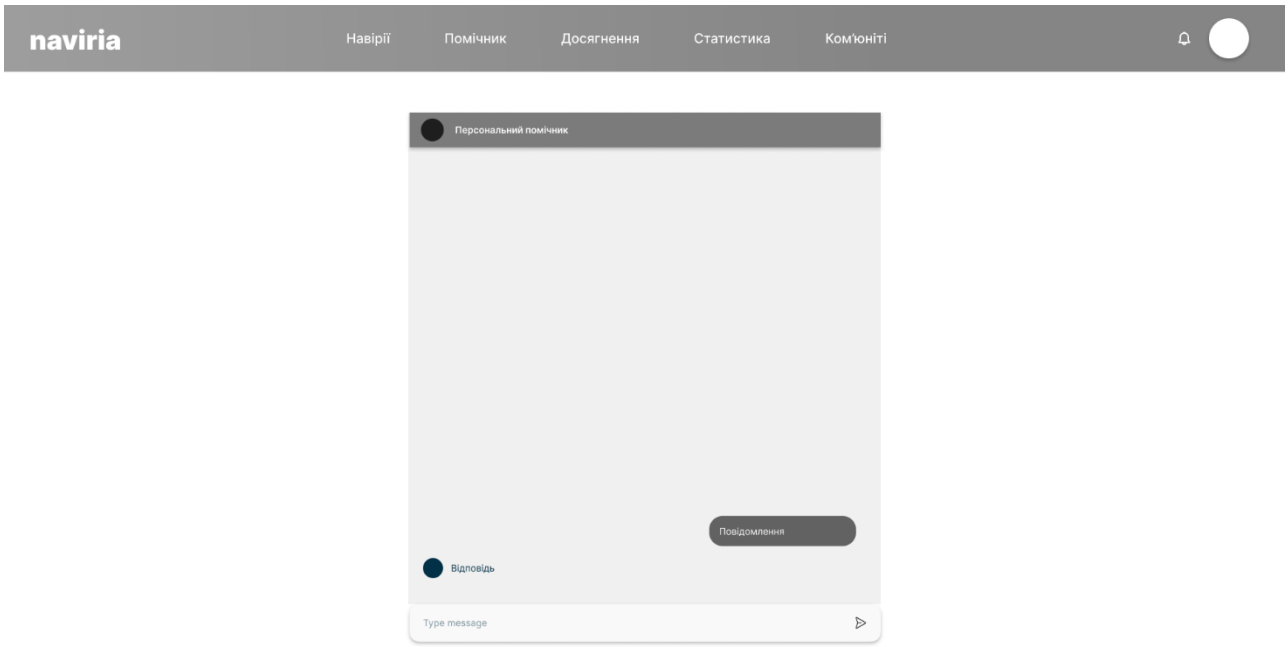

Рисунок 3.8 – Макет сторінки «Персональний помічник» (рисунок виконано самостійно)

Інтерфейс складається зі стандартного поля виведення повідомлень, області введення тексту та панелі з навігацією у верхній частині сторінки. Усі елементи компонуються у вертикальному макеті, що забезпечує звичну логіку діалогу та відповідає сучасним уявленням про зручний чат-інтерфейс.

З огляду на функціональне призначення даного інтерфейсу – спілкування з персональним помічником, реалізованим на базі моделі ChatGPT – структура діалогу, послідовність повідомлень та їх формат максимально наближені до інтерфейсу класичного ChatGPT. Це зроблено свідомо для забезпечення користувачам звичного досвіду взаємодії з текстовим генератором відповідей.

Однак на рівні повноцінного візуального дизайну (у фінальній версії інтерфейсу) чат буде стилістично адаптовано під загальний візуальний стиль вебзастосунку – із використанням відповідної кольорової палітри, типографіки та іконографії, характерних для всієї системи. Такий підхід дозволить зберегти баланс між функціональною звичністю та візуальною узгодженістю всіх модулів застосунку.

Наступним макетом є сторінка з головним функціоналом усього вебдодатку, а саме «Навірії» що є відображенням різноманітних задач (див. рис. 3.9).

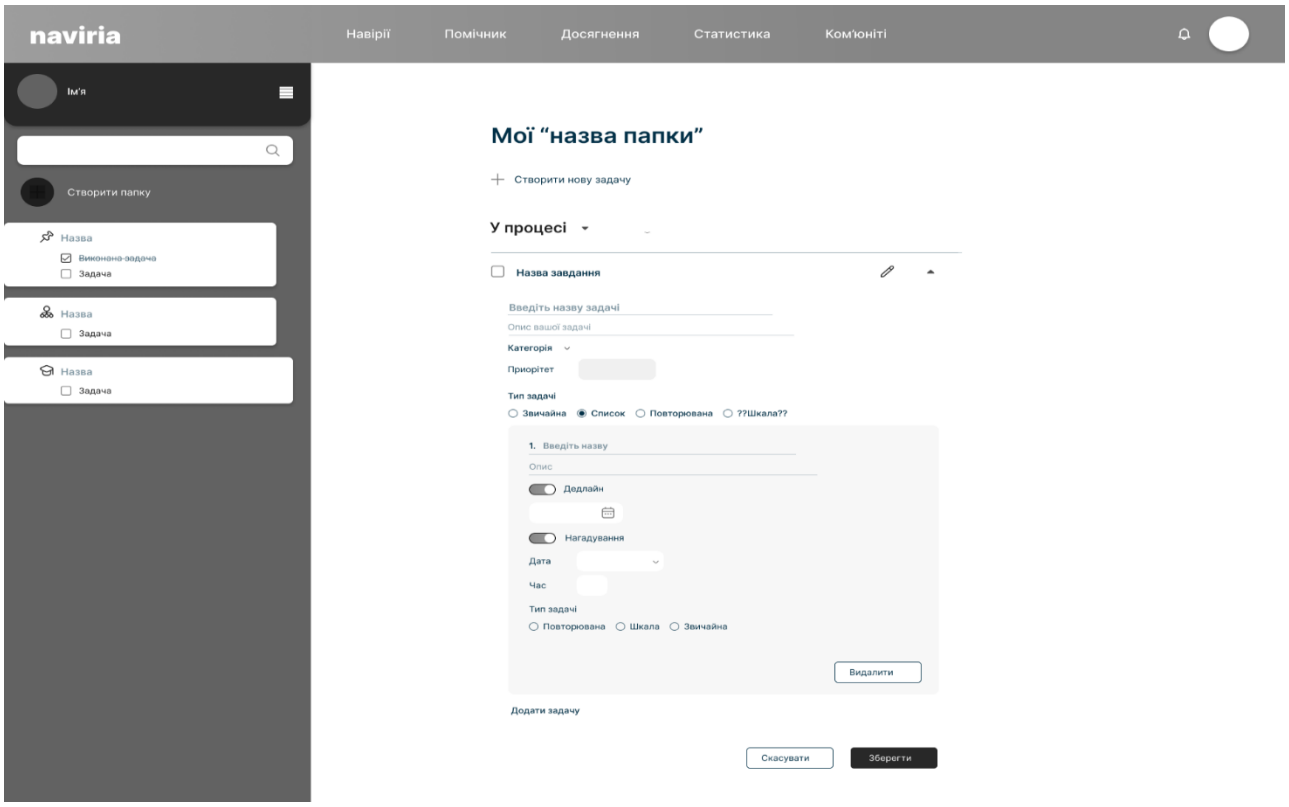

Рисунок 3.9 – Макет сторінки «Навірії» (рисунок виконано самостійно)

Інтерфейс реалізовано у вигляді двоколонкової структури: зліва розташована панель навігації з переліком створених папок і задач, праворуч – основна робоча область для створення, перегляду та редагування вибраної задачі.

У лівій частині передбачено можливість створення нових папок, а також доступ до існуючих – кожна з них відображає пов’язані з нею задачі у вигляді окремих блоків.

Права частина містить функціональну форму редагування задачі. Користувач може ввести назву, опис, обрати категорії, встановити пріоритет, тип задачі

(звичайна, список, повторювана), а також додати підзадачі. Для кожної підзадачі доступні окремі налаштування: дедлайн, нагадування, дата та час виконання, тип. Взаємодія реалізована через стандартні інтерфейсні елементищо забезпечує зручність та швидкість заповнення.

У нижній частині форми передбачено кнопки збереження або скасування змін, а також опція для видалення підзадачі. Загальний дизайн макета є функціонально насиченим, однак структуровано впорядкованим, що дозволяє користувачу ефективно керувати задачами різного рівня складності у межах єдиного інтерфейсу.

Наступною не менш важливою сторінкою є «Ком'юніті», детальніше оглянемо її макет зображений на рисунку 3.10.

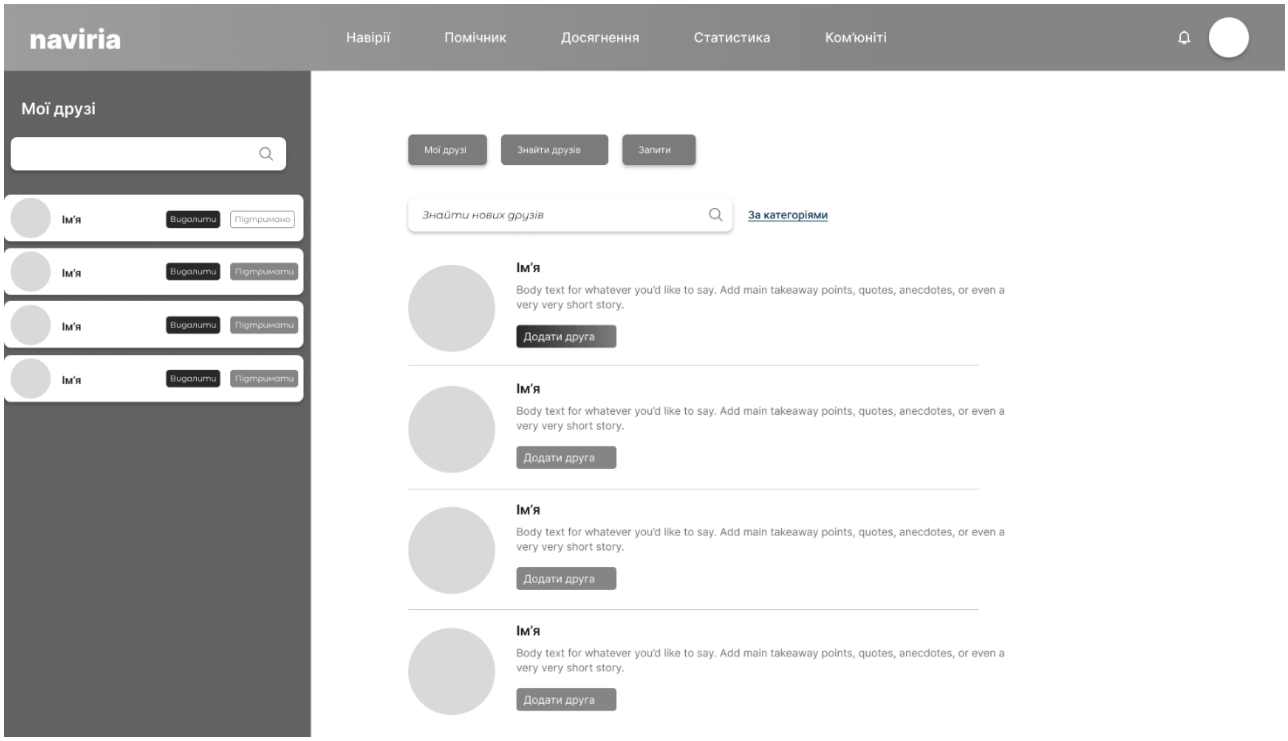

Рисунок 3.10 – Макет сторінки «Ком'юніті» (рисунок виконано самостійно)

Ця сторінка виконує функцію централізованого керування контактами, включаючи перегляд існуючих друзів, пошук нових, а також обробку вхідних запитів на додавання. Основна логіка інтерфейсу реалізована через систему вкладок у верхній частині центральної області: «Мої друзі», «Знайти друзів» та «Запити». У залежності від обраної вкладки змінюється відповідний контент і

функціональність.

У вкладці «Мої друзі» відображається список підтверджених контактів, розміщений у лівій бічній панелі, де для кожного користувача доступні дії, зокрема «Видалити» та «Підтримати». У режимі «Знайти друзів» центральна область заповнюється профілями користувачів, яких можна додати до списку друзів – для цього передбачено кнопку «Додати друга». Присутня також текстова панель пошуку та опція фільтрації «За категоріями», що дозволяє структурувати великий масив результатів. У вкладці «Запити» (не показано на зображенні, але передбачено макетом) користувач отримує доступ до переліку запитів на додавання у друзі з можливістю їх прийняття або відхилення.

Тепер рознлянемо каркасний макет сторінки профілю користувача, яка відображає основну публічну інформацію, доступну для перегляду іншим користувачам у межах застосунку (див. рис. 3.11).

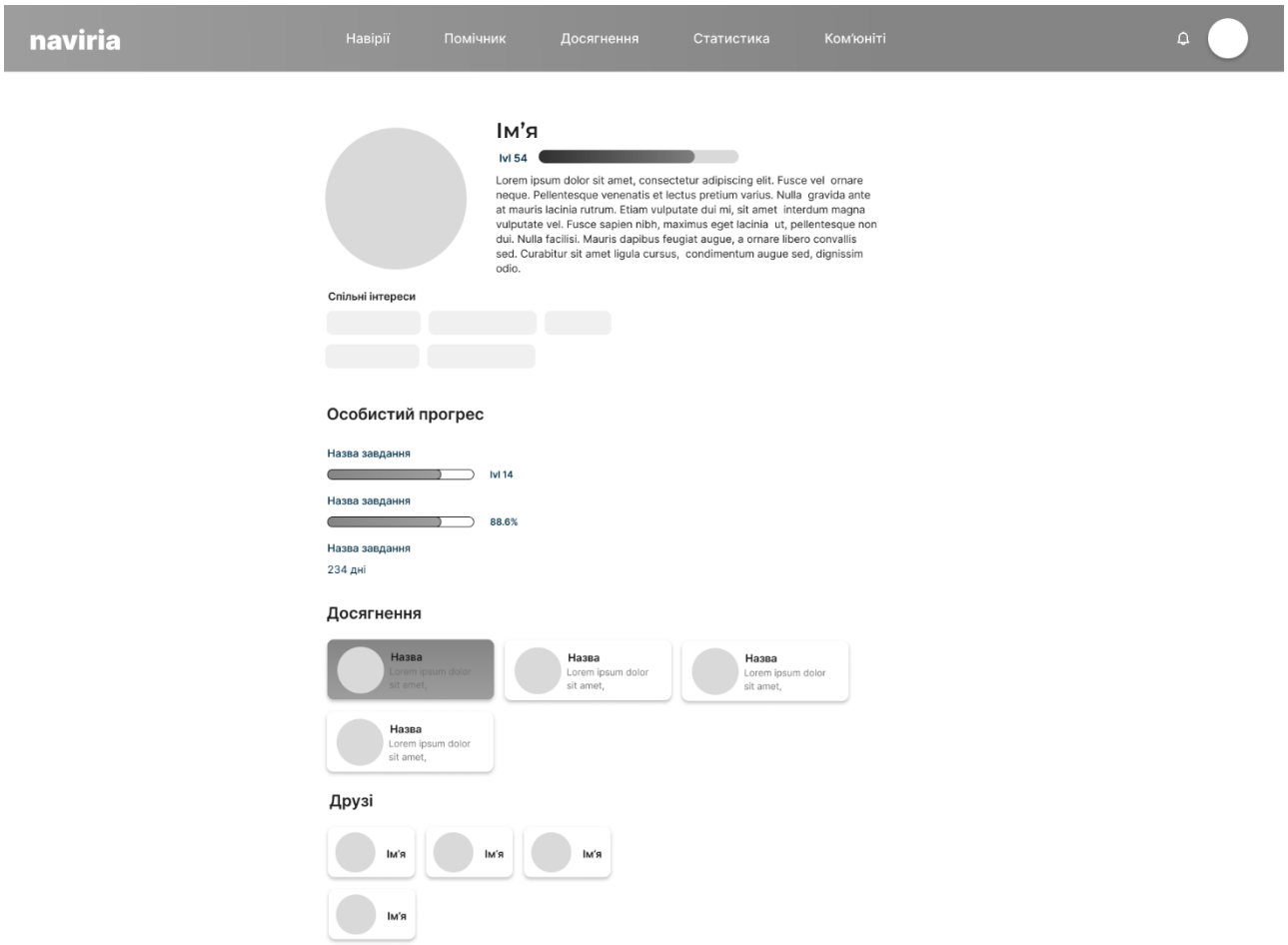

Рисунок 3.11 – Макет сторінки профілю (рисунок виконано самостійно)

Можна побачити, що у верхній частині розміщено аватар користувача, ім’я,

рівень, короткий опис та список спільних інтересів (буде відсутній на сторінці особистого профілю), представлений у вигляді тегів.

Нижче розташований блок «Особистий прогрес», у якому відображаються активні задачі або напрямки, над якими працює користувач, із зазначенням поточного рівня, відсотка виконання або кількості днів активності. Далі наведено секцію «Досягнення», що містить перелік віртуальних нагород, отриманих користувачем у процесі взаємодії з системою. Візуально досягнення подаються у вигляді плиток з назвою, коротким описом та індикатором отримання.

У нижній частині інтерфейсу розміщується розділ «Друзі», в якому виводяться аватари користувачів, що перебувають у списку контактів. Кожен елемент представлений у компактному форматі з іменем, без додаткових інтерактивних дій у рамках цього макету.

Наступною є сторінка досягнень, зображена на рисунку 3.12.

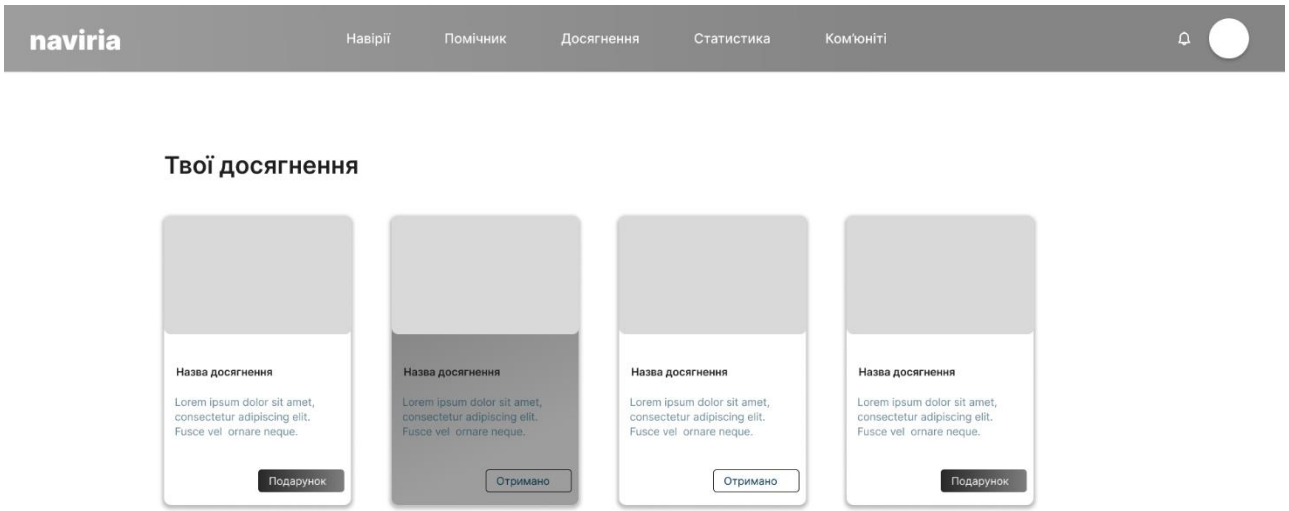

Рисунок 4.12 – Макет сторінки «Досягнення» (рисунок виконано самостійно)

Інтерфейс представлений у вигляді сітки з картками, кожна з яких містить назву досягнення, опис і кнопку дії. Така структура дозволяє користувачу швидко ознайомитися з наявними досягненнями та взаємодіяти з ними без зайвих переходів. Картки візуально відрізняються залежно від їх рідкості: рідкісні досягнення мають більш яскраве оформлення або декоративні елементи, що підкреслюють їхню унікальність. На картках наявна кнопка отримання подарунку,

яка змінює вигляд при натисканні, сигналізуючи про успішне виконання дії та формуючи відчуття винагороди у користувача.

І останнім макетом є сторінка зі статистикою. На рисунку 3.13 зображено каркасний макет цієї сторінки, що складається з кількох аналітичних блоків, зокрема діаграм активності, графіків виконання задач і підрахунку прогресу за обраний період. Блоки організовано у логічні секції, що дозволяє швидко орієнтуватися в даних та робить інтерфейс інформативним і зручним для аналізу власної продуктивності. Розглянемо їх детальніше.

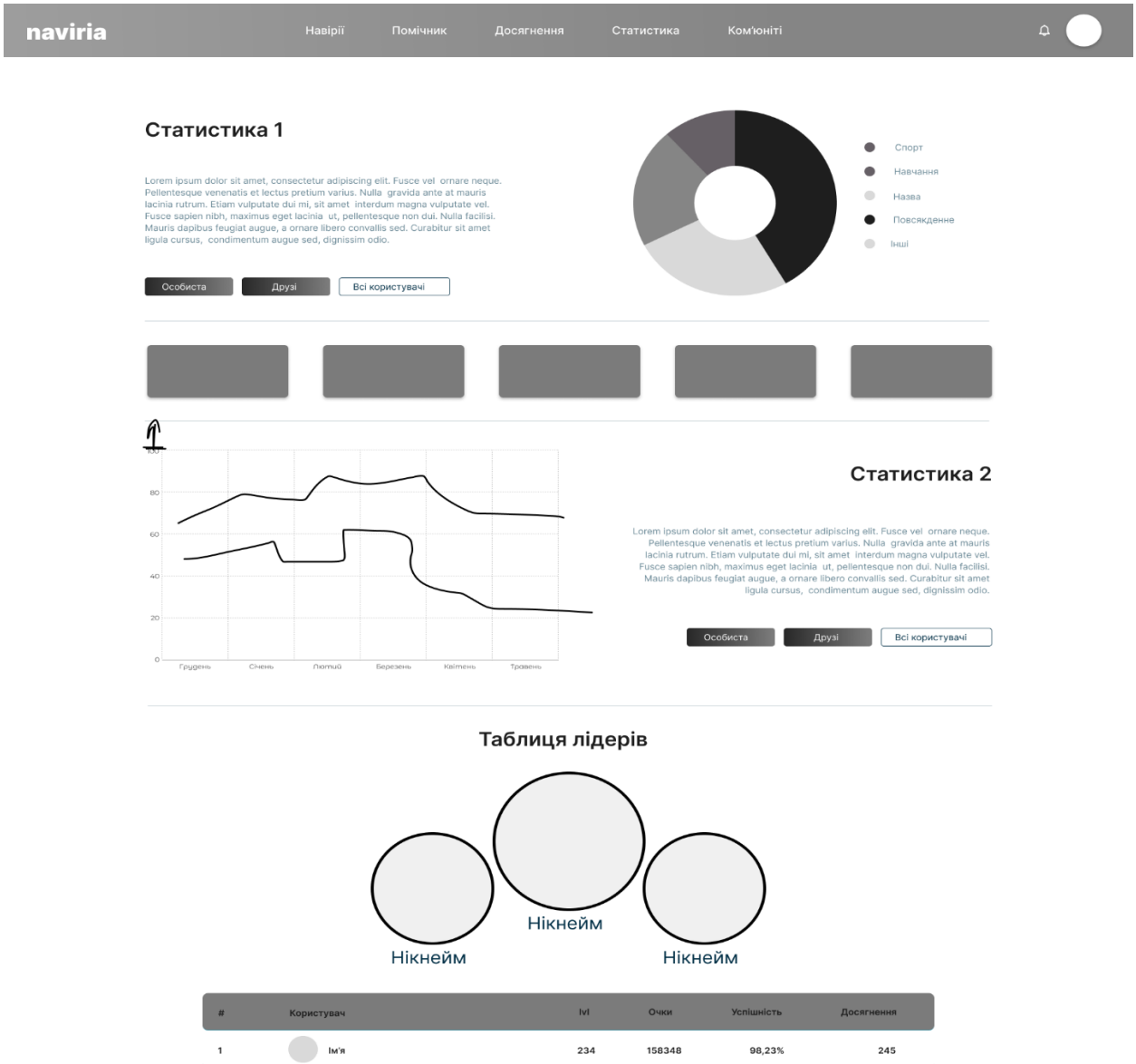

Рисунок 3.12 – Макет сторінки «Статистика» (рисунок виконано самостійно)

У верхній частині інтерфейсу розташовано кругову діаграму для

відображення розподілу активностей за категоріями, а також текстовий опис і перемикачі фільтрації за типом даних: «Особиста», «Друзі», «Всі користувачі».

Нижче блоки з аналітичними картками, призначеними для виведення узагальнених метрик або коротких звітів.

У центральній частині сторінки розміщено лінійну діаграму для візуалізації динаміки певних показників у часі, зліва від якої вказано вісь значень, а знизу – місяці. Поруч – ще один блок статистичного опису з аналогічними перемикачами фільтрації.

У нижній частині макету представлено розділ «Таблиця лідерів», що складається з двох частин: візуального представлення трьох топових користувачів у вигляді кругових аватарів з нікнеймами та детальної таблиці з позиціями, іменами, рівнями, кількістю очок, відсотком успішності та числом досягнень. Така структура дозволяє охопити як загальні аналітичні дані, так і елементи гейміфікації через рейтинг користувачів.

У результаті проведеного UI/UX-проєктування було сформовано структуровану систему каркасних макетів основних інтерфейсних сторінок застосунку. Створені прототипи відображають логіку взаємодії користувача з функціоналом системи, забезпечують послідовність, узгодженість і зручність навігації. Спроєктовані рішення лягли в основу подальшої реалізації клієнтської частини, визначаючи розміщення елементів, принципи візуальної ієрархії та сценарії використання, що дозволить забезпечити ефективну та комфортну роботу з програмним продуктом.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Обґрунтування вибору платформи для клієнтської частини

У процесі розробки клієнтської частини інформаційної системи було прийнято рішення обрати платформу веббраузера як основне середовище функціонування користувацького інтерфейсу. Такий вибір зумовлений низкою технічних, економічних та експлуатаційних чинників, що забезпечують високу ефективність розробки, зручність експлуатації та широку доступність для кінцевих користувачів.

Передусім, слід зазначити, що веббраузер є універсальним програмним середовищем, яке підтримується на переважній більшості сучасних пристроїв та операційних систем, зокрема Windows, Linux, macOS, Android та iOS. Така кросплатформеність дозволяє охопити максимальну кількість користувачів без необхідності створення окремих нативних застосунків для кожної платформи. Таким чином, використання веббраузера сприяє зменшенню витрат на розробку та обслуговування програмного забезпечення. Окрім цього, веббраузерна платформа не потребує встановлення додаткового програмного забезпечення на пристрій користувача, що значно знижує бар’єр входу. Користувачеві достатньо мати доступ до інтернету та сучасного браузера, щоб розпочати роботу з системою. Такий підхід не лише підвищує рівень доступності, але й позитивно впливає на користувацький досвід, оскільки усуває необхідність у додаткових налаштуваннях або оновленнях на стороні клієнта.

З технічної точки зору, сучасні веббраузери повною мірою підтримують передові стандарти веб-технологій [9], включаючи HTML5, CSS3 та ECMAScript (JavaScript), що дозволяє створювати динамічні, інтерактивні та адаптивні інтерфейси користувача. Також браузери надають потужні вбудовані інструменти розробника, які істотно спрощують процес налагодження, тестування та оптимізації клієнтського коду, що є вагомим аргументом на користь цієї платформи.

Важливо також врахувати, що обрана платформа повністю відповідає концепції клієнт-серверної архітектури, яка передбачає відокремлення логіки представлення від логіки обробки даних. У рамках цього підходу клієнтська частина, що виконується у веббраузері, забезпечує візуалізацію даних, отриманих через API-запити до серверної частини системи. Це забезпечує масштабованість рішення, його гнучкість та можливість подальшого розвитку без зміни фундаментальної архітектури.

Окремої уваги заслуговує аспект оновлення клієнтської частини. Веб-платформа дозволяє реалізовувати централізоване оновлення застосунку: усі зміни впроваджуються на сервері та стають миттєво доступними для всіх користувачів без потреби у ручному оновленні з боку клієнта. Це сприяє оперативному усуненню помилок, швидкому впровадженню нового функціоналу та загальному підвищенню рівня інформаційної безпеки.

Таким чином, використання веббраузера як платформи для клієнтської частини програмного забезпечення є цілком виправданим та раціональним вибором, що базується на об’єктивних технічних та організаційних перевагах. Такий підхід дозволяє реалізувати ефективну, масштабовану та зручну в експлуатації систему, що відповідає сучасним вимогам до розробки веб-застосунків.

4.2 Вибір інструментів програмної реалізації

У процесі створення клієнтської частини програмного забезпечення було обрано мову програмування JavaScript, що є найбільш поширеним і технічно доцільним рішенням у сфері веб-розробки. Такий вибір обумовлений її функціональними можливостями, глибокою інтеграцією з браузерами, а також порівняльними перевагами над альтернативними мовами, які потенційно могли бути використані.

JavaScript є єдиною мовою програмування, яка нативно підтримується всіма сучасними веб-браузерами. Це означає, що для її виконання не потрібно встановлення додаткових плагінів або компіляторів – код виконується

безпосередньо у середовищі браузера, що гарантує високу швидкодію та стабільну роботу застосунку на різних пристроях та операційних системах.

Крім нативної підтримки, JavaScript забезпечує асинхронну обробку подій (завдяки механізмам Promises, async/await, Event Loop) [10], що є критично важливим для побудови інтерактивних інтерфейсів, які реагують на дії користувача та взаємодіють із сервером у режимі реального часу. Також JavaScript має динамічну типізацію, що пришвидшує розробку, особливо на ранніх етапах створення прототипу інтерфейсу. З точки зору інструментальної підтримки, JavaScript має найбільшу екосистему бібліотек, фреймворків і інструментів, які активно підтримуються спільнотою та постійно оновлюються. Це значно зменшує час на реалізацію типових завдань, таких як валідація форм, анімації, робота з API, локалізація тощо.

У процесі вибору мови програмування також були розглянуті альтернативні варіанти, зокрема TypeScript, Dart, Python (через Transcrypt або Brython) та ClojureScript.

- TypeScript, попри свої переваги у вигляді статичної типізації, є надбудовою над JavaScript і потребує компіляції у нього [11]. Використання TypeScript доцільне у великих проєктах з високими вимогами до безпеки типів, однак на етапі базової реалізації клієнтської частини така складність була б надлишковою;
- Dart, що використовується у фреймворку Flutter Web, вимагає специфічного середовища виконання та має нижчу сумісність із існуючими бібліотеками JavaScript. Крім того, браузерна підтримка Dart є обмеженою у порівнянні з JavaScript;
- Python, хоч і є популярною мовою загального призначення, не призначений для прямого виконання в браузері. Існують проєкти для транспіляції Python у JavaScript (наприклад, Brython), однак вони не забезпечують повної сумісності та не є промисловими стандартами;
- ClojureScript і подібні мови використовуються в нішевих задачах і мають круту криву навчання, що є недоцільним у рамках командного проєкту з

обмеженими ресурсами.

Таким чином, вибір JavaScript як основної мови програмування для реалізації клієнтської частини є логічно обґрунтованим. Він базується на нативній сумісності з браузерами, розвиненій екосистемі, підтримці асинхронної логіки, широкому розповсюдженні, доступності документації та практичному досвіді розробників. Усі ці чинники дозволяють забезпечити ефективну, гнучку та масштабовану реалізацію інтерфейсу користувача.

Одним із ключових завдань під час розробки клієнтської частини інформаційної системи є вибір ефективного інструменту для побудови інтерфейсу користувача. Зважаючи на необхідність створення інтерактивного, гнучкого та масштабованого веб-застосунку, було обрано бібліотеку React, яка надає сучасні засоби для реалізації компонентної архітектури, управління станом та оптимізованого рендерингу інтерфейсів. Такий вибір зумовлений низкою технічних, функціональних і практичних переваг. Опишемо їх детальніше.

React є бібліотекою з відкритим кодом, яка розробляється та підтримується компанією Meta, а також великою спільнотою розробників. Вона забезпечує компонентний підхід до побудови інтерфейсу, що дозволяє розділити програму на незалежні, повторно використовувані елементи з чітко визначеною логікою та станом. Така структура істотно покращує підтримуваність коду, сприяє його повторному використанню та забезпечує зручність масштабування.

Однією з ключових особливостей React є використання віртуального DOM (Document Object Model)[12] – механізму, який дозволяє мінімізувати кількість реальних змін у DOM браузера, тим самим підвищуючи продуктивність інтерфейсу. Це особливо важливо при реалізації динамічних сторінок із великою кількістю взаємодій користувача, де традиційний підхід до оновлення DOM може призводити до помітних затримок. React також забезпечує уніфіковану модель обробки стану та подій, що дозволяє передбачувано керувати поведінкою інтерфейсу. Завдяки таким концепціям, як односпрямований потік даних та можливість централізованого управління станом (через useState, useReducer або

зовнішні бібліотеки), забезпечується чіткість і прозорість логіки роботи застосунку.

З технічної точки зору, React має низький поріг входу для розробників, які вже володіють JavaScript, і дозволяє поступово ускладнювати проєктну архітектуру без необхідності одразу впроваджувати складні інструменти. При цьому React легко інтегрується з іншими бібліотеками та API, підтримує серверний рендеринг, адаптивні інтерфейси та розширення функціоналу шляхом використання численних плагінів.

При розгляді альтернатив, таких як Vue.js та Angular, було враховано декілька ключових факторів. Vue.js, хоча і має подібний компонентний підхід, менш поширений у великих комерційних проєктах, а його екосистема менш стандартизована. Angular є потужним фреймворком, однак він має високу складність, жорстку структуру, а також вимагає додаткового часу на освоєння. На відміну від них, React надає більшу гнучкість і краще підходить для поступового впровадження у проєкт.

Таким чином, вибір бібліотеки React як основного інструменту для побудови інтерфейсу користувача є обґрунтованим з точки зору продуктивності, масштабованості, підтримуваності та відповідності сучасним практикам розробки веб-застосунків. Її використання дозволяє створити ефективну, модульну та зручну для супроводу клієнтську частину системи, яка відповідає поставленим вимогам.

Для організації процесу розробки та збірки клієнтської частини програмного забезпечення було обрано інструмент Vite [13]. Це сучасний білдер, що забезпечує високу швидкодію, зручну конфігурацію та ефективну інтеграцію з сучасними JavaScript-бібліотеками. Вибір Vite є обґрунтованим з погляду як продуктивності, так і архітектурної відповідності вимогам проєкту.

Однією з головних переваг Vite є його швидкий старт та миттєве оновлення під час розробки, що досягається завдяки використанню нативних можливостей браузера (ES-модулів) та асинхронного завантаження залежностей. На відміну від класичних білдерів, таких як Webpack, які виконують повну попередню збірку проєкту, Vite застосовує принцип «on-demand compilation», що значно скорочує час

запуску проєкту та знижує навантаження на систему розробника. Ще однією важливою перевагою є простота конфігурації. У типовому випадку Vite потребує мінімального налаштування для початку роботи, а завдяки офіційним плагінам – зокрема @vitejs/plugin-react – інтеграція з бібліотекою React відбувається автоматично та без ускладнень. Це дозволяє зосередитися безпосередньо на реалізації бізнес-логіки замість витрат часу на налаштування складного інструментарію. Крім того, Vite підтримує оптимізовану збірку для продакшну, використовуючи сучасний компілятор Rollup у якості бекенду. Це забезпечує компактність та продуктивність підсумкового коду, зменшуючи обсяг JavaScript-бандлів та прискорюючи завантаження сторінки на стороні користувача.

При виборі інструменту також були розглянуті альтернативні рішення, такі як Webpack, Parcel та Create React App (CRA). Webpack, незважаючи на свою потужність і гнучкість, має складну конфігурацію та демонструє нижчу швидкість у режимі розробки. Parcel позиціонується як «zero-config» білдер, однак має меншу гнучкість у порівнянні з Vite. Інструмент Create React App є традиційним рішенням для створення застосунків на React, проте має менш гнучку архітектуру і не підтримує сучасні оптимізації за замовчуванням, зокрема модульну структуру залежностей, яку реалізує Vite.

Загалом, використання Vite дозволяє суттєво оптимізувати процес розробки, підвищити ефективність роботи та забезпечити кращу продуктивність готового застосунку. Усі ці чинники роблять його доцільним вибором як для невеликих проєктів, так і для масштабованих клієнтських систем.

Для забезпечення високої якості програмного коду, його читабельності та підтримуваності в рамках проєкту було обрано інструмент статичного аналізу – ESLint. Його основною метою є виявлення синтаксичних і логічних помилок, а також дотримання єдиного стилю програмування в межах команди.

ESLint є одним із найпоширеніших та найбільш гнучких інструментів для перевірки JavaScript-коду [14]. Його конфігурація дозволяє як застосування стандартних наборів правил (@eslint/js), так і підключення плагінів, що враховують специфіку проєкту. У межах цієї системи були використані eslint-plugin-react-hooks

та eslint-plugin-react-refresh для забезпечення коректного використання React-специфічних конструкцій і підтримки гарячого оновлення під час розробки. Інструмент інтегрується з популярними середовищами розробки та виконує перевірку коду в режимі реального часу, що сприяє зниженню кількості помилок і покращенню командної роботи. У порівнянні з альтернативами, такими як JSHint або StandardJS, ESLint має ширший функціонал і підтримку плагінів, що робить його найбільш придатним для потреб сучасної фронтенд-розробки.

Таким чином, ESLint дозволяє автоматизувати контроль якості коду, підтримувати узгодженість стилю в межах команди та запобігати типовим помилкам ще до етапу тестування або запуску застосунку.

У проєкті також застосовуються додаткові бібліотеки, які розширюють функціональні можливості клієнтської частини та полегшують реалізацію окремих підсистем. Їхнє використання дозволяє зосередитися на логіці застосунку, уникаючи ручної реалізації повторюваних або складних рішень, що вже стандартизовані у професійному середовищі.

Однією з таких бібліотек є Recharts, яка використовується для побудови графіків і діаграм [15]. Цей інструмент базується на компонентному підході та інтегрується з React, що забезпечує гнучкість у створенні візуалізацій даних без потреби писати низькорівневий код. Recharts підтримує широкий спектр графічних елементів, зокрема стовпчикові, лінійні, кругові та комбіновані діаграми. Окрім того, вона надає зручні засоби для створення інтерактивних елементів: тултіпів (підказок), легенд, механізмів масштабування, а також анімацій, що покращують користувацький досвід.

Це дозволяє ефективно відображати аналітичну або статистичну інформацію у зрозумілому вигляді, сприяє візуальному аналізу прогресу користувача та підвищує загальну інформативність інтерфейсу. Використання Recharts значно спрощує процес побудови візуальних звітів, дає змогу адаптувати їх до різних сценаріїв застосування та легко масштабувати при розширенні функціоналу.

Для реалізації автентифікації користувачів через обліковий запис Google було використано бібліотеку @react-oauth/google. Вона є офіційно підтримуваним

клієнтським обгортанням для інтеграції з сервісом Google OAuth 2.0 [16], що дозволяє швидко впровадити механізм авторизації за допомогою зовнішнього постачальника ідентифікації. Підключення цієї бібліотеки потребує мінімального налаштування, забезпечує стабільну взаємодію з Google API та дозволяє уникнути складнощів із ручною реалізацією протоколів авторизації.

Бібліотека гарантує безпечну обробку авторизаційних токенів, автоматичне оновлення сесій, відповідність сучасним вимогам до захисту персональних даних та можливість гнучкої конфігурації авторизаційного процесу. Крім того, вона підтримується Google, що знижує ризики втрати актуальності чи виникнення критичних помилок у майбутньому.

Таким чином, вибір цих інструментів обумовлений прагненням до швидкої розробки, масштабованості проєкту, зручності в підтримці та актуальності технологій. Усі використані бібліотеки мають широку підтримку спільноти, докладну документацію та доведену ефективність у багатьох реальних проєктах. Це робить їх оптимальним рішенням для реалізації клієнтської частини застосунку з високими вимогами до якості, зручності та безпеки.

5 РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

5.1 Структура проєкту

Клієнтська частина розроблюваного програмного забезпечення реалізована у вигляді односторінкового, побудованого з використанням бібліотеки React. Такий архітектурний підхід забезпечує високу продуктивність, зручну взаємодію з інтерфейсом без потреби в перезавантаженні сторінки, а також сприяє легкому масштабуванню функціональності та підтримці коду.

Структура проєкту сформована відповідно до принципів модульності та розподілу відповідальностей, що дозволяє ізолювати різні аспекти логіки та інтерфейсу в межах окремих директорій. Уся вихідна логіка розміщена в каталозі src, який є основним робочим простором фронтенд-розробки (див. рис. 5.1).

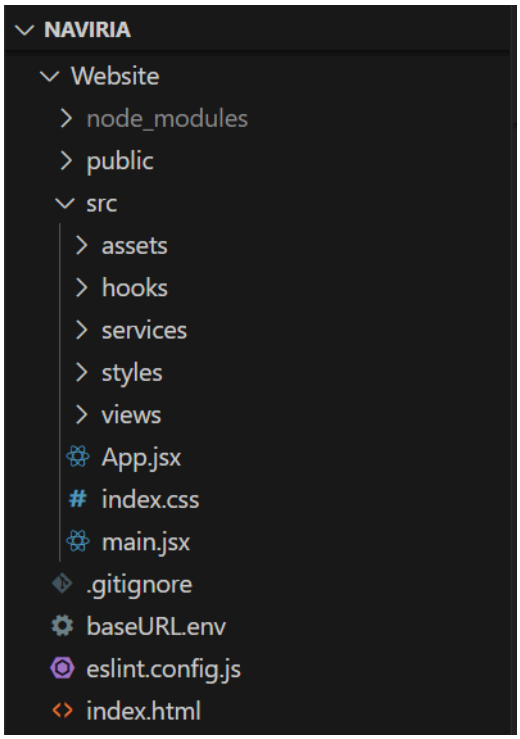

Рисунок 5.1 – Структура проєкту (рисунок виконано самостійно)

У директорії assets зберігаються статичні ресурси, зокрема графічні зображення, які використовуються в інтерфейсі. Папка hooks містить користувацькі React-хуки, що інкапсулюють окремі поведінкові сценарії, зокрема авторизацію користувача та приховання елементів інтерфейсу. Такий підхід

дозволяє повторно використовувати логіку та підтримувати її у відокремленому вигляді.

Особливу роль у проєкті відіграє каталог services, де зосереджена логіка взаємодії з серверною частиною через API. Для кожного напряму (реєстрація, профіль, друзі, завдання, статистика тощо) реалізовано окремі модулі-сервіси, що відповідають за відправлення та обробку HTTP-запитів. Це дозволяє дотримуватись принципу інкапсуляції та забезпечує зручну підтримку і розширення функціоналу.

Стилізація інтерфейсу згрупована в окремій папці styles, де зберігаються як CSS, так і LESS-файли. Структура стилів організована за функціональними блоками, що відповідають окремим модулям або сторінкам застосунку, що спрощує внесення змін до зовнішнього вигляду інтерфейсу без ризику порушення глобального стилю.

Передбачено також наявність директорії views, яка, згідно з типовими практиками, містить основні візуальні компоненти рівня сторінок. Такі компоненти відповідають за компоновку внутрішніх блоків інтерфейсу та обробку їхнього стану.

Головний компонент застосунку реалізовано у файлі App.jsx, де визначається базове компонування інтерфейсу та налаштовується маршрутизація. Початкова ініціалізація React-застосунку здійснюється у main.jsx, де React-компонент монтується в DOM-дерево браузера. Загальні стилі проєкту підключені через index.css, а глобальні константи оформлені у файлах const.less та const.css.

Загалом, така архітектура клієнтської частини забезпечує високу структурованість, зручність у розробці, тестуванні та масштабуванні проєкту, а також відповідає актуальним вимогам до сучасних веб-застосунків.

5.2 Взаємодія з користувачем та інтерфейс

Інтерфейс користувача відіграє ключову роль у забезпеченні зручності та ефективності роботи з програмним забезпеченням. У межах реалізації клієнтської частини було розроблено низку інтерфейсних компонентів, що забезпечують

цілісну, логічну та інтуїтивно зрозумілу взаємодію користувача із системою. Побудова інтерфейсу здійснена з використанням компонентного підходу, властивого бібліотеці React, що дає змогу поділити застосунок на незалежні частини з чітко визначеною відповідальністю. Реалізовано стандартні елементи взаємодії, зокрема форми введення даних, кнопки дій, випадаючі списки, повідомлення про помилки, таблиці зі статистикою тощо. Компоненти, що відповідають за ці функції, розміщені у відповідних підкаталогах проєкту та мають зрозумілу ієрархію, що забезпечує зручність навігації в кодовій базі.

Тепер розглянемо найцікавіші приклади реалізації. Розпочнемо огляд зі сторінки реєстрації, що зображена на рисунку 5.2.

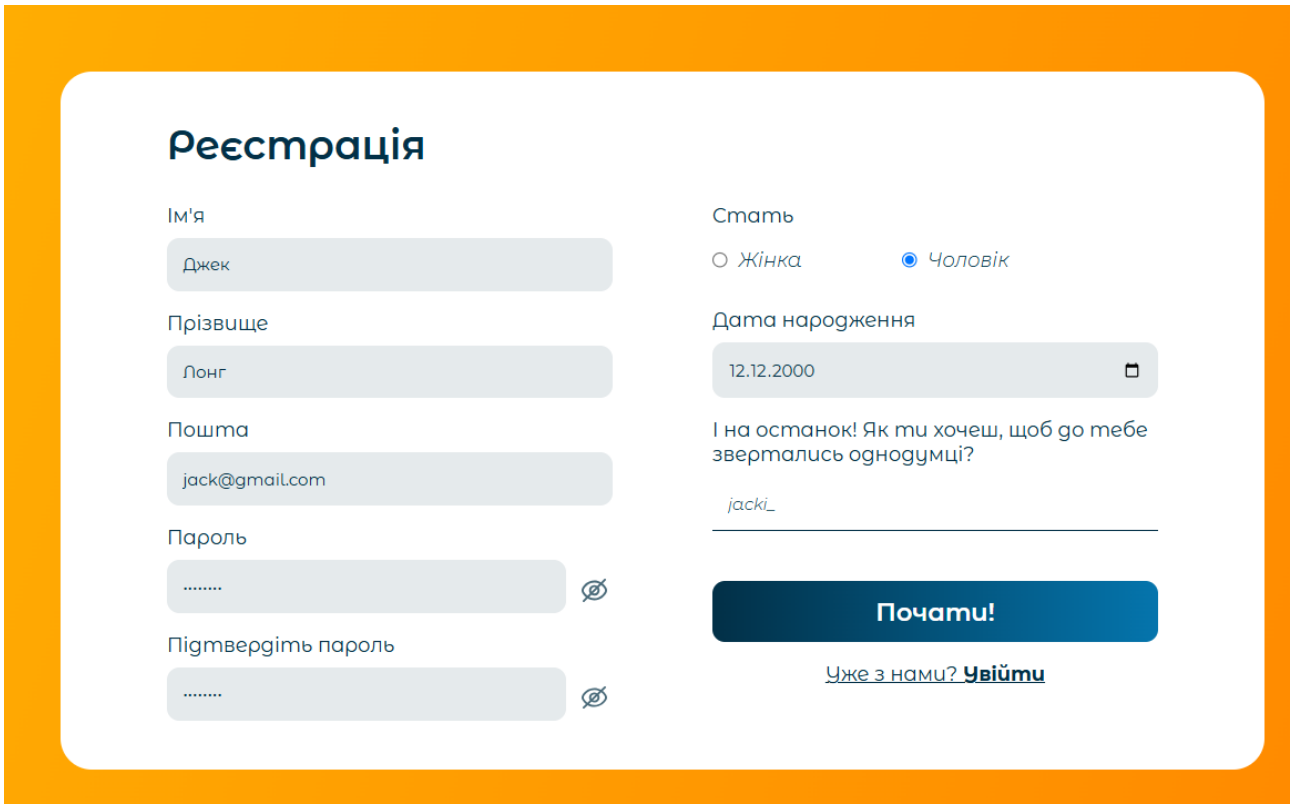

Рисунок 5.2 – Сторінка реєстрації (рисунок виконано самостійно)

Сторінка реєстрації є яскравим прикладом застосування принципів чіткого компонування. Візуально форма поділена на дві колонки, що створює зрозумілу логіку заповнення: зліва розміщені поля для введення особистих даних (ім’я, прізвище, пошта, пароль), праворуч – радіокнопки вибору статі, дата народження, нікнейм та кнопка надсилання. Така структура реалізована у вигляді верстки з

класами column1, column2, що дозволяє гнучко управляти виглядом компонента через CSS/LESS (див. лістинг В.1, додаток В).

Вся стилізація інтерфейсу винесена в окремі стилізовані файли, зокрема registration.less, які містять правила форматування шрифтів, полів, відступів, кольорів, станів кнопок тощо. Стилі згруповані за логікою компонента, що полегшує підтримку зовнішнього вигляду та унеможливлює конфлікти з іншими частинами застосунку. Приклад стилізації цього компоненту можна побачити у лістингу В.2, додаток В.

Перейдемо до огляду хедеру, який зображений на рисунку 5.3.

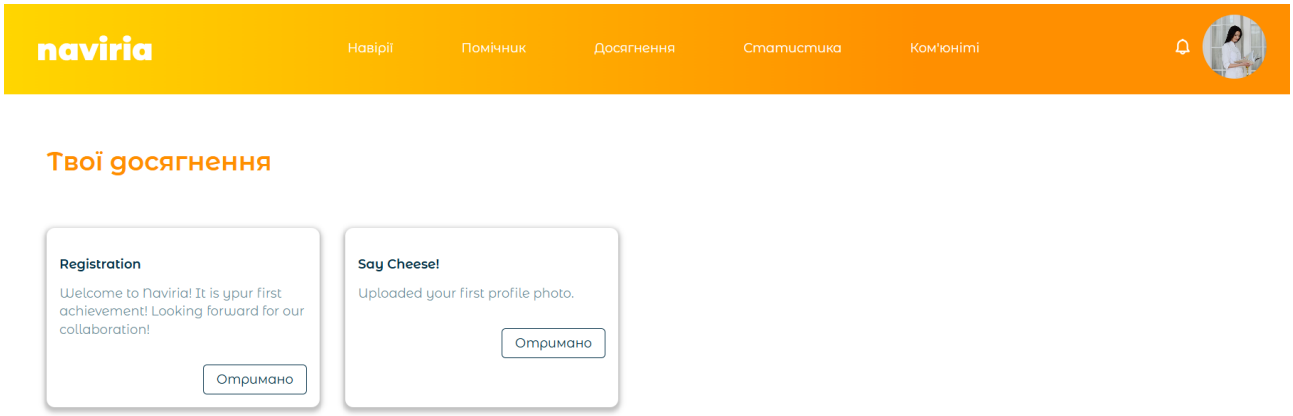

Рисунок 5.3 – Хедер (рисунок виконано самостійно)

Візуально хедер містить логотип, набір посилань (<Link> із React Router) на функціональні сторінки застосунку – «Навірії», «Помічник», «Досягнення», «Статистика», «Ком'юніті», а також панель дій, що включає кнопку сповіщень і зображення профілю користувача. Інтерфейс адаптовано під різні розміри екранів, елементи вирівняно за допомогою flex, а стилі оформлено в окремому файлі header.less (див. лістинг В.3, додаток В).

Компонент є динамічним: при натисканні на іконку сповіщень відбувається запит до API (getNotifications) і виведення повідомлень. Якщо серед них є нові (isNew: true), іконка змінюється з bell.svg на new-notif.svg, що візуально сигналізує користувачеві про непрочитану інформацію. Коли панель сповіщень відкривається, усі повідомлення автоматично позначаються як прочитані

(markAllNotificationsRead()), а при натисканні за межами блоку – приховуються. Реалізацію цієї частини можна побачити у лістингу В.4, додаток В.

Варто зазначити, що хедер також інтегрується з користувацькими даними: при завантаженні компонента відбувається асинхронний запит до getProfile, і, якщо користувач авторизований, у хедері відображається його аватар. Це створює відчуття персоналізації інтерфейсу та сприяє кращій орієнтації користувача в системі.

Так як поп-ап повідомлень тісно пов’язаний з хедером, далі розглянемо саме його (див. рис. 5.4).

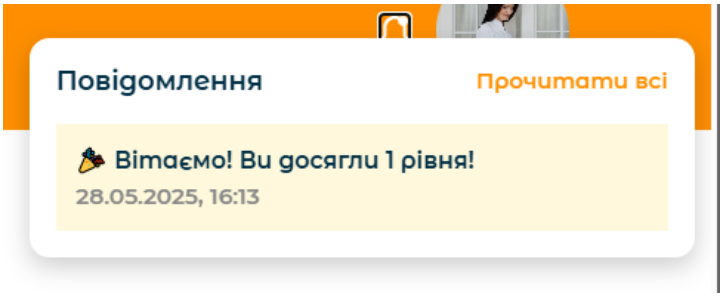

Рисунок 5.4 – Повідомлення (рисунок виконано самостійно)

Інтерфейс повідомлень реалізовано у вигляді окремого компонента Notifications, який приймає два параметри: масив повідомлень і функцію зворотного виклику onMarkRead. Компонент побудований з урахуванням принципів адаптивності й читабельності. Кожне повідомлення має текстову частину та мітку часу, яка форматована у зручному вигляді за локалізацією uk-UA. Реалізація цієї частини знаходиться у лістингу В.5, додаток В.

Повідомлення виводяться у зворотному хронологічному порядку, а нові повідомлення (з ознакою isNew: true) візуально виділяються через додатковий CSS-клас unread. Це дозволяє користувачу швидко зорієнтуватися в тому, які події є новими. Стилі компонента визначені у файлі notifications.less і включають, зокрема, кольорове маркування непрочитаних повідомлень, відступи, тінь контейнера та стилі кнопки «Прочитати всі». Розглянути частину написаних стилів можна у лістингу В.6, додаток В.

Інтерактивність реалізовано через кнопку «Прочитати всі», яка викликає функцію markAllNotificationsRead() з відповідного сервісу. Після успішного виконання запиту викликається передана ззовні функція onMarkRead, що дозволяє перезавантажити список або оновити інтерфейс батьківського компонента. Це забезпечує повну синхронізацію стану повідомлень між користувачем і сервером.

У випадку відсутності нових повідомлень, користувач бачить текстове повідомлення: «Немає нових повідомлень», що підтримує логіку прозорості системи та запобігає неправильному сприйняттю порожнього інтерфейсу.

Тепер розглянемо сторінку ком'юніті, вона зображена на рисунку 5.5.

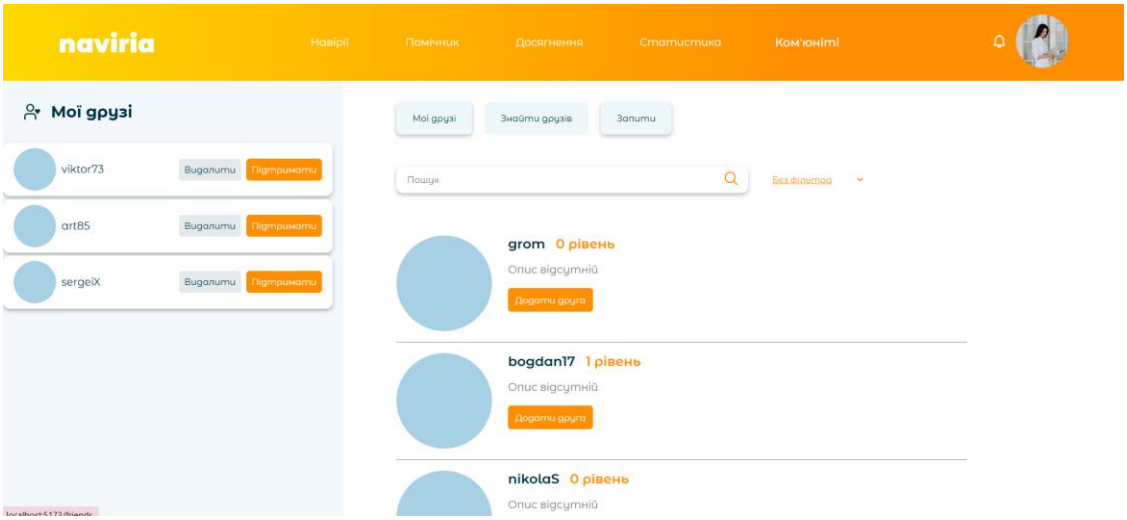

Рисунок 5.5 – Сторінка «Ком'юніті» (рисунок виконано самостійно)

Модуль «Ком'юніті» є однією з найбільш функціональних сторінок застосунку, реалізований у компоненті Friends.jsx. Його інтерфейс побудовано за табовою структурою, що забезпечує швидкий доступ до трьох основних режимів: «Мої друзі», «Знайти друзів» та «Запити». Перемикання між вкладками змінює вміст головної області, але зберігає загальну структуру сторінки (див. лістинг В.7).

Усі частини сторінки поділено на два ключові блоки: бічна панель (side-bar), що відображає перелік друзів користувача у скороченому вигляді, та основна область (main), яка динамічно змінюється залежно від активної вкладки. Для відображення даних використовуються структуровані блоки у форматі списків (.item, .friend, .info), які оформлено за допомогою класів із файлу friends.less (див. лістинг В.8, додаток В).

Побудова інтерфейсу базується на використанні компонентів та умовного рендерингу. Залежно від контексту, для кожного користувача можуть з’являтись різні кнопки дій: додати друга, прийняти чи відхилити запит, видалити друга тощо. Розмітка адаптивна: компонування здійснено через flex, а зображення, текст та кнопки підлаштовуються під ширину екрана.

Робота з формами та таблицями реалізована через інтерактивні елементи. Зокрема, форма пошуку та селектори фільтрації категорій дозволяють користувачу шукати друзів або запити за текстовим запитом та категорією інтересів. Пошуковий рядок (input) працює динамічно – введення запиту та натискання кнопки search-btn викликає відповідну функцію пошуку, а результат одразу відображається у таблиці користувачів. Для кожного запису у таблиці виводиться аватар, нікнейм, рівень користувача, опис та набір дій.

Кнопки дій (.add-friend, .accept, .reject, .remove) мають стилізований вигляд і змінюють свій зовнішній стан при наведенні або взаємодії. Вони також підключені до функцій асинхронної взаємодії з сервером, що дозволяє миттєво оновлювати інтерфейс без перезавантаження сторінки.

Тепер розглянемо сторінку «Навірії», що зображена на рисунку 5.6.

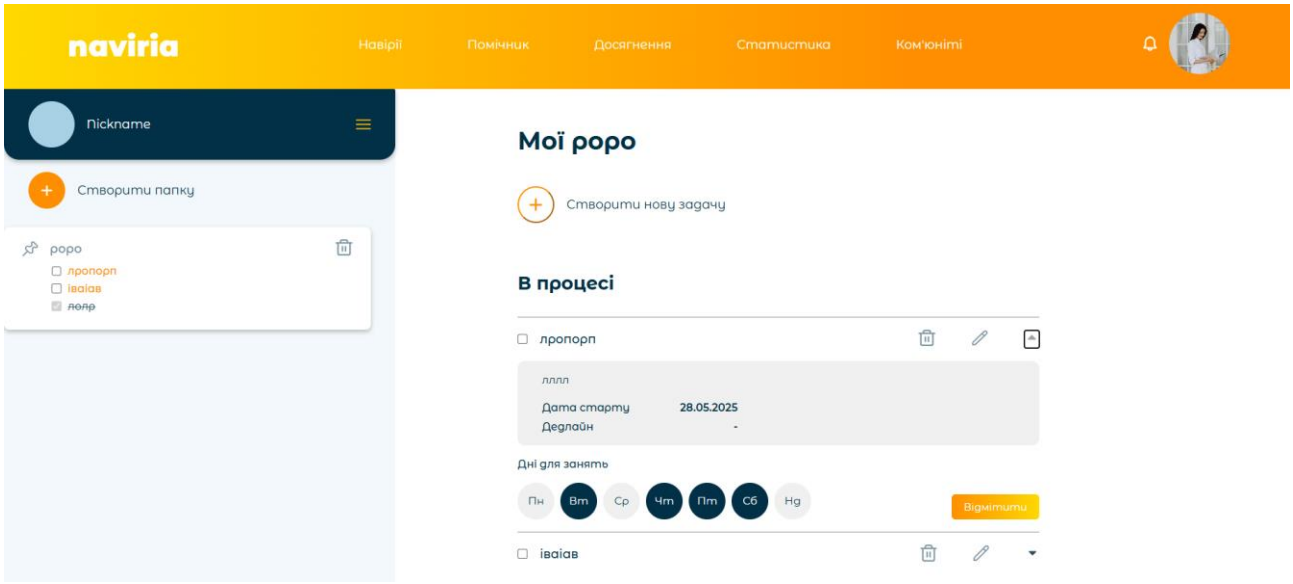

Рисунок 5.6 – Сторінка «Навірії» (рисунок виконано самостійно)

Сторінка «Навірії» є основним інструментом для управління задачами користувача. Інтерфейс реалізовано у вигляді двопанельного компонування:

ліворуч розміщується панель зі списком папок (груп задач), а праворуч – основний вміст із задачами обраної групи. Цей підхід забезпечує зручну навігацію, дозволяючи швидко перемикатися між категоріями завдань, які окреслює сам користувач. Подібна організація структури сприяє логічному впорядкуванню задач, що особливо важливо при роботі з великими обсягами інформації.

Компонентна структура сторінки побудована на використанні трьох основних компонентів: Tasks, Task та Subtasks. Компонент Tasks відповідає за загальне керування списками, обробку запитів до API, створення нових задач та зчитування даних. Кожне завдання виводиться через компонент Task, який залежно від типу задачі може мати різну структуру – проста, повторювана, зі шкалою або з підзадачами. Усі задачі мають уніфікований зовнішній вигляд із динамічним розширенням за потреби. Підзадачі, у свою чергу, відображаються через компонент Subtasks, який надає можливість встановлення статусу виконання кожного підпункту (див. лістинг В.9, додаток В).

Інтерфейс динамічний і адаптивний, реалізований із застосуванням гнучких контейнерів (flex) і умовного рендерингу, що забезпечує високий рівень інтерактивності. Наприклад, при створенні нової папки з’являється форма введення, яка автоматично замінює кнопку «Створити папку». Аналогічно, при розгортанні задачі – довантажується її розширена інформація без перезавантаження сторінки. Такий підхід значно покращує взаємодію з інтерфейсом і підвищує загальну зручність роботи.

Користувач може взаємодіяти із задачами за допомогою форм, чекбоксів, кнопок і полів введення. Для кожної задачі доступна можливість редагування, видалення, розгортання деталей або фіксації прогресу (наприклад, для повторюваних задач). У задачах зі шкалою (scale) реалізовано візуалізацію у вигляді заповненої горизонтальної шкали прогресу з цифровими значеннями поточного і цільового результату, що дає змогу швидко оцінити виконання у відсотковому вираженні. Усі елементи адаптовано під різні розміри екранів, що робить інтерфейс зручним як на десктопах, так і на мобільних пристроях. Таким

чином, сторінка «Навірії» поєднує багатофункціональність, інтуїтивну логіку та візуальну зрозумілість.

Форми на сторінці, зокрема TaskForm, TaskEditForm, а також форма створення папки, використовують інтуїтивний макет із підписами, плейсхолдерами та інформативними підказками. Переглянути частину коду для форм можна у лістингу В.10, додаток В. Для збереження змін використовуються кнопки з візуально вираженим станом (наприклад, неактивні при порожніх полях).

Особливу увагу в інтерфейсі приділено візуалізації даних, що реалізована на сторінці «Статистика»(див. рис. 5.7) за допомогою бібліотеки Recharts. Компонент Statistics відображає два основних типи графіків: кругову діаграму (PieChart) та лінійний графік (LineChart), кожен з яких надає користувачеві різні ракурси для аналізу своєї продуктивності.

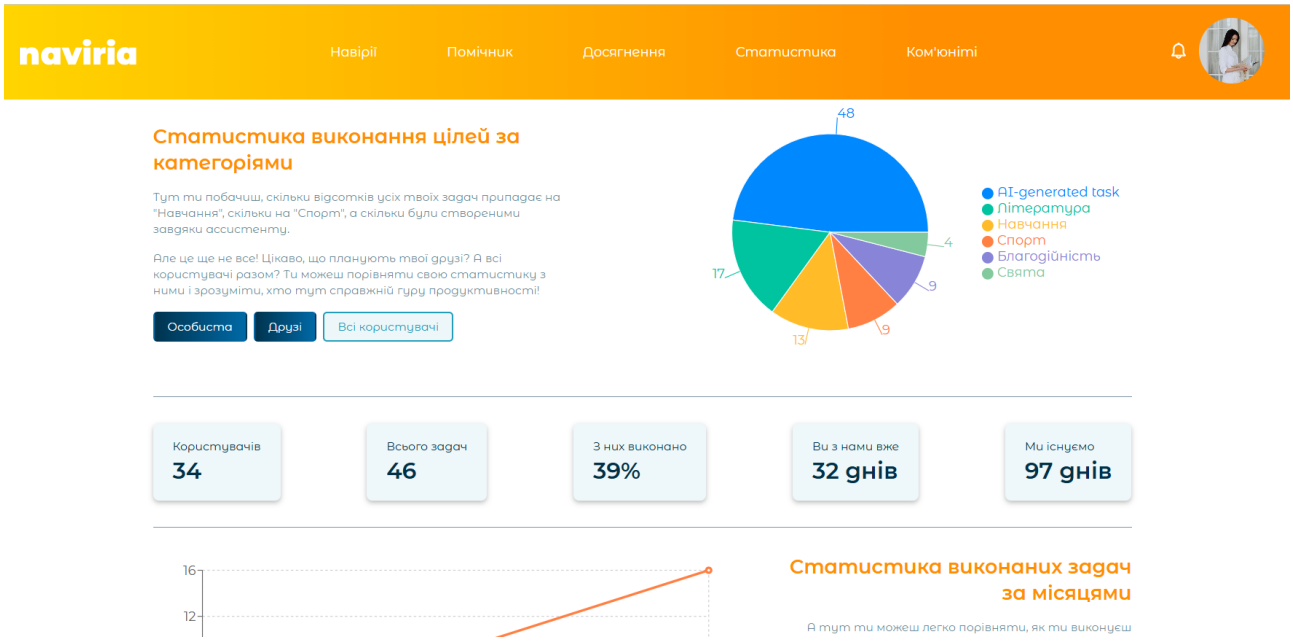

Рисунок 5.7 – Сторінка «Статистика» (рисунок виконано самостійно)

Кругова діаграма візуалізує розподіл задач за категоріями, наприклад: «Навчання», «Спорт», «Саморозвиток» тощо. Користувач може перемикатися між трьома режимами: особисті дані, дані друзів і глобальна статистика. Залежно від обраного режиму, на графік динамічно підвантажуються відповідні значення. Діаграма оформлена у фірмових кольорах, які автоматично призначаються кожному сектору з використанням попередньо визначеного масиву COLORS.

Підключено легенду (Legend) та підказки (Tooltip) для забезпечення кращого інтерфейсу.

Лінійний графік демонструє кількість виконаних задач по місяцях(див. рис. 5.8). Три лінії (користувача, його друзів і всіх користувачів платформи) дозволяють порівняти динаміку активності. Графік адаптований до різних розмірів екрану (ResponsiveContainer) і супроводжується пояснювальними елементами: підписами осей, легендою та інтерактивними точками. Структура даних формується програмно – дані з API об'єднуються у масив, де кожен елемент містить інформацію про конкретний місяць і відповідні значення для кожної категорії.

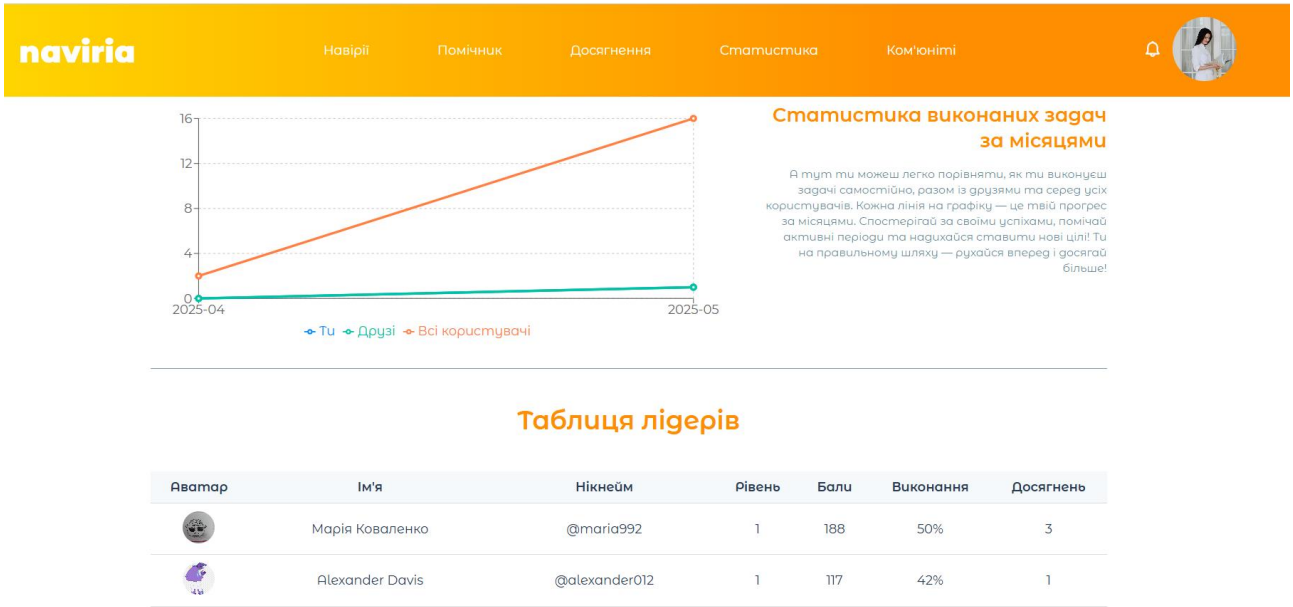

Рисунок 5.8 – Лінійна діаграма та таблиця лідерів (рисунок виконано самостійно)

Окрім графіків, у нижній частині сторінки відображається таблиця лідерів з показниками рівня, балів, відсотку виконання задач і кількості досягнень. Це є елементом візуальної гейміфікації, що підвищує мотивацію користувачів.

5.3 Інтеграція з зовнішніми сервісами

Робота клієнтської частини застосунку тісно пов'язана з інтеграцією з зовнішніми сервісами, зокрема бекенд-API, сервісами авторизації, статистики, взаємодії з друзями та візуалізації даних. Всі API-запити реалізовано за допомогою fetch з урахуванням налаштувань заголовків, обробки помилок і передачі

авторизаційних токенів. Крім цього у проєкті застосовується модульний підхід до організації запитів. Усі функції для взаємодії з API винесено у файли в директорії services.

Однією з ключових функцій є можливість авторизації за допомогою Google OAuth 2.0, реалізованої через бібліотеку @react-oauth/google (див. рис. 5.9). Користувач після входу за допомогою Google отримує авторизаційний токен, який надсилається до бекенду через функцію googleLogin у AuthServices.jsx, переглянути код цієї функції можна у лістингу В.11. Отриманий токен зберігається в localStorage, що дозволяє надалі додавати його до всіх запитів як заголовок Authorization.

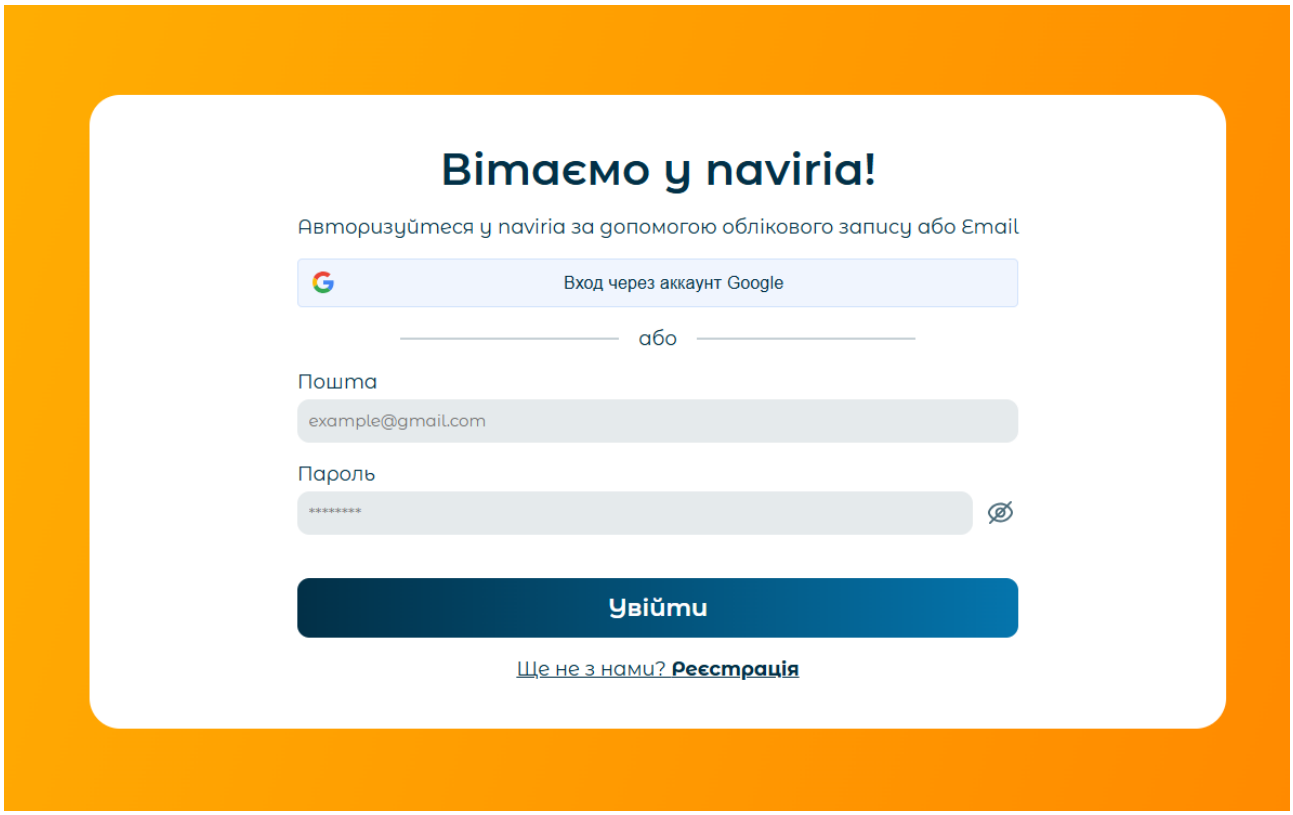

Рисунок 5.9 – Сторінка «Логін» (рисунок виконано самостійно)

Тепер розглянемо найцікавіші приклади реалізованих сервісів.

Сервіс AuthServices.jsx відповідає за автентифікацію користувачів. У ньому реалізовано методи для класичної форми входу (login), реєстрації (registration) та авторизації за допомогою облікового запису Google, що було описано вище. Функції обробляють відповіді від сервера, зберігають отримані токени доступу в

localStorage та, у разі потреби, витягують з токена userId. Додатково реалізовано функцію authHeaders(), що централізовано формує авторизаційні заголовки для всіх захищених запитів.

Сервіс AssistantChatServices.jsx реалізує інтеграцію з модулем чат-асистента (див. рис. 5.10). У ньому передбачено дві основні функції: fetchChatHistory() – для отримання історії взаємодій з асистентом, а також sendMessageToAssistant(message, isTaskRequest) – для надсилання нового повідомлення. Обидві функції використовують токен авторизації з localStorage, що дозволяє зберігати персоналізовану історію користувача.

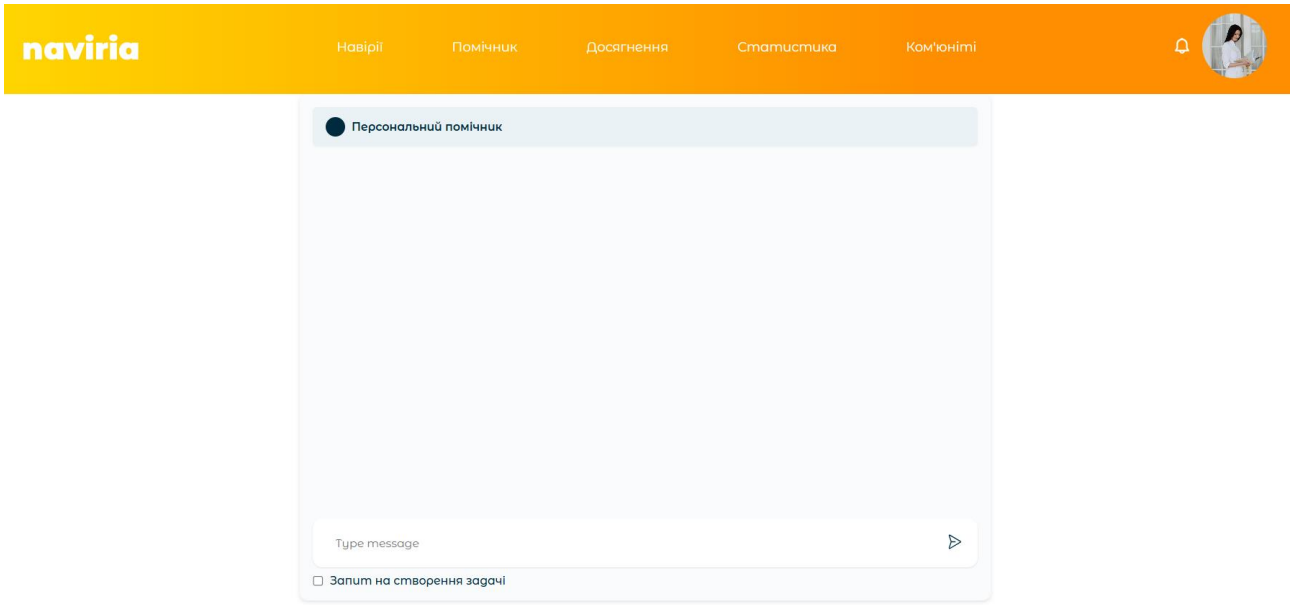

Рисунок 5.10 – Чат з персональним асистентом (рисунок виконано самостійно)

Наступним є сервіс FriendsServices.jsx, який відповідає за соціальну взаємодію між користувачами. Він містить функції для отримання списку друзів, пошуку потенційних контактів, обробки запитів у друзі, а також для прийняття, відхилення чи скасування таких запитів. Запити реалізовано з використанням динамічних параметрів (наприклад, userId) та опціональної фільтрації за категоріями інтересів. Усі запити забезпечені відповідною обробкою помилок, що гарантує стабільність роботи навіть за відсутності відповіді від сервера.

Сервіс StatisticsServices.jsx реалізує обмін даними зі статистичним модулем серверної частини. До його функцій належить отримання агрегованих показників –

загальна кількість задач, рівень користувача, виконані завдання тощо. Крім того, сервіс дозволяє формувати дані для візуалізації: кругових діаграм розподілу задач та лінійних графіків прогресу за місяцями. Реалізовано також окрему функцію для отримання рейтингової таблиці користувачів. Кожна функція працює з урахуванням прав доступу та виконує автентифіковані запити.

Перейдемо до TasksServices.jsx. Цей сервіс є одним із найбільш функціонально насичених. Він забезпечує повний цикл CRUD-операцій над задачами та підзадачами: створення (createTask, createSubtask), оновлення, видалення, а також спеціальні дії на кшталт check-in для повторюваних задач або збереження прогресу по шкалі, розглянути реалізацію деякий операцій можна у лістингу В.12, додаток В. Крім цього, у файлі міститься функція fetchCategories() для завантаження категорій задач(див. лістинг В.13, додаток В). Запити містять відповідні заголовки авторизації та включають обробку випадків, коли сервер повертає лише статус без тіла відповіді (наприклад, 204 No Content).

5.4 Забезпечення якості та стабільності коду

У розробці сучасних клієнтських застосунків якість програмного коду є не лише технічним критерієм, а й необхідною умовою для подальшого масштабування, підтримки та інтеграції нових функцій. У цьому проєкті досягнення високої якості коду забезпечується шляхом поєднання статичного аналізу, стандартизації стилю написання, структурної організації компонентів і реалізації механізмів захисту від помилок під час виконання.

Одним із основних інструментів контролю якості став ESLint – статичний аналізатор, який виконує перевірку вихідного коду JavaScript на відповідність узгодженим правилам. У конфігурації ESLint використано набір базових і додаткових правил, зокрема: обов’язкове використання строгого режиму (strict mode), перевірка наявності невикористаних змінних, уникнення дублювання імпортів, контроль вкладеності логічних блоків, дотримання правил форматування JSX. У разі виявлення порушення ESLint генерує повідомлення, що дозволяє

розробнику миттєво внести виправлення. Це сприяє зменшенню кількості логічних помилок ще до запуску застосунку.

Інструмент ESLint інтегровано у процес збирання застосунку через Vite, що дозволяє автоматично виконувати перевірку коду перед деплоєм. Крім того, розробники можуть запускати перевірку локально, що створює культуру постійного дотримання вимог до якості ще на ранніх етапах написання коду. Такий підхід формує спільну відповідальність за підтримку читаємості, безпеки та надійності програмного продукту.

Окрема увага приділяється структурній організації кодової бази. Компоненти інтерфейсу розміщено в директорії components, кожен з них має свою стилізацію (.less), що зберігається у відповідній папці styles. API-запити ізольовано у файлах services, що забезпечує інкапсуляцію логіки роботи з сервером і спрощує її повторне використання. Крім того, в окремих директоріях розміщено кастомні хуки (hooks) та допоміжні утиліти (utils). Завдяки цьому код логічно розділений, а компоненти мають чітку відповідальність, що відповідає принципам архітектури компонентно-орієнтованого програмування.

Для гарантування стабільності застосунку реалізовано систему обробки помилок. Усі запити, що виконуються через fetch, супроводжуються перевіркою статусу відповіді (res.ok) приклад якої можна побачити у лістингу В.14, додаток В, після чого за потреби викидається виняток з повідомленням про помилку. Наприклад, у разі невдалої авторизації користувача функція повертає текстову помилку, яка виводиться у відповідному інтерфейсному компоненті. Таким чином, користувач отримує зрозумілий фідбек, а система не допускає подальших збоїв через некоректні дані.

На рівні форм введення реалізовано локальну перевірку полів, яка не дозволяє відправляти дані без обов’язкових значень або за наявності очевидних помилок (наприклад, короткого пароля чи порожнього імені). Валідація доповнюється виведенням пояснювальних підказок, що забезпечує коректність взаємодії з користувачем та запобігає введенню неконсистентної інформації у систему (див. рис. 5.11).

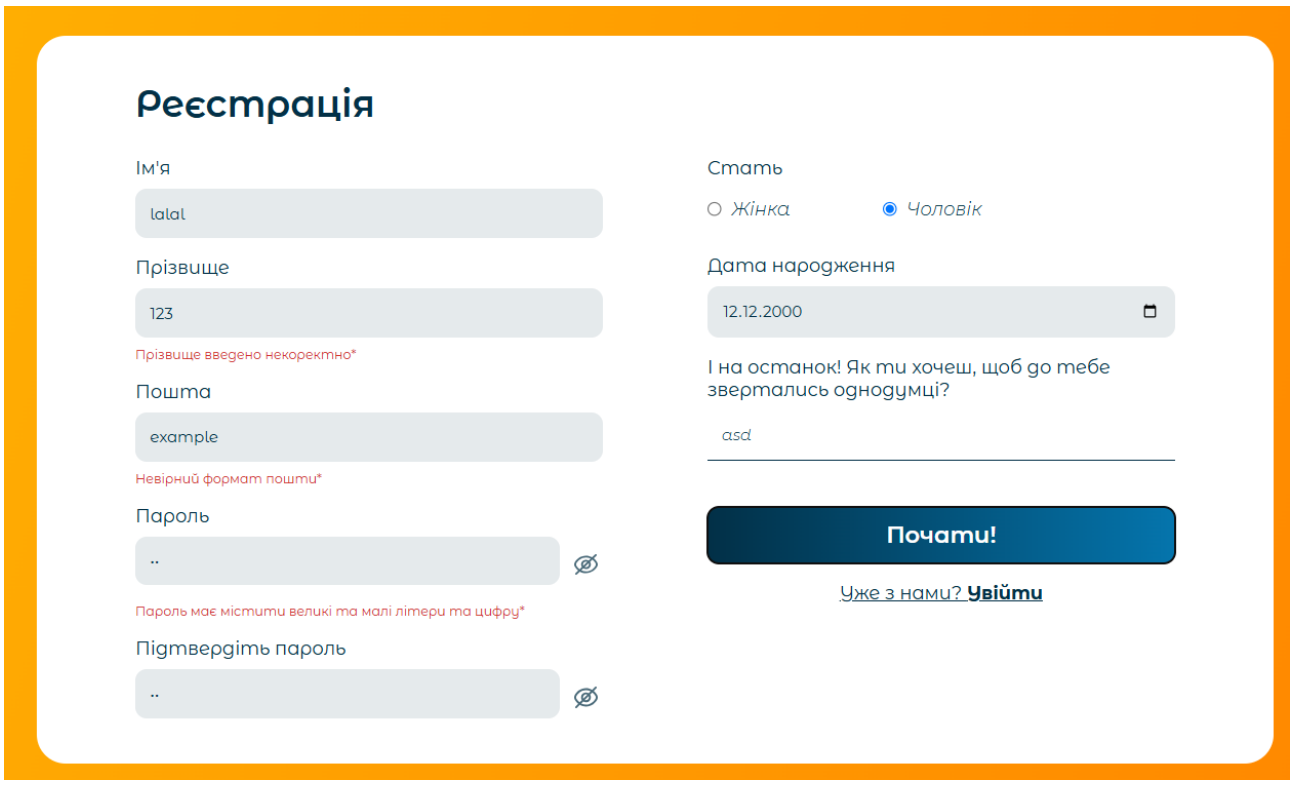

Рисунок 5.11 – Приклад валідації (рисунок виконано самостійно)

Дотримання єдиного стилю кодування підтримується не лише ESLint, а й домовленостями команди. Наприклад, усі функції мають приблизно однаковий синтаксис, змінні – зрозумілі імена, уникається зайве вкладення логіки, повторювані фрагменти винесено в утиліти. Такі підходи відповідають принципам чистого коду (Clean Code) і полегшують колективну роботу над проєктом.

У підсумку, поєднання інструментів статичного аналізу, архітектурної дисципліни, структурної чистоти та обробки винятків дозволило досягти високого рівня якості та стабільності коду. Це створює надійне підґрунтя для подальшого розвитку застосунку, впровадження нових функцій і підтримки великої бази користувачів без втрати продуктивності та керованості.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було здійснено розробку клієнтської частини інформаційної вебсистеми з використанням сучасного технологічного стеку. Основна увага приділялася побудові зручного та функціонального інтерфейсу користувача, реалізації ефективної взаємодії з серверною частиною, забезпеченню стабільності та якості коду, а також інтеграції зовнішніх сервісів.

У процесі роботи обґрунтовано вибір ключових технологій – мови програмування JavaScript, бібліотеки React для побудови інтерфейсу, інструменту збірки Vite та лінтера ESLint для контролю якості коду. Всі компоненти було реалізовано згідно з принципами модульності, інкапсуляції та повторного використання, що забезпечує гнучкість і масштабованість клієнтської частини.

Реалізовано низку функціональних сторінок: реєстрацію та вхід користувача, сторінку ком'юніті, задач та статистики. В інтерфейсі застосовано сучасні підходи до стилізації, адаптивності, побудови таблиць, форм і графічної візуалізації даних. Крім того, здійснено інтеграцію з зовнішніми сервісами, зокрема Google OAuth 2.0, що значно спрощує процес автентифікації користувачів.

Забезпечено захист доступу до конфіденційних даних, реалізовано перевірку прав доступу через токен-автентифікацію, а також механізми обробки помилок і валідації введених даних. Уся функціональність протестована в процесі інтерактивної перевірки та ручного тестування.

У результаті виконання роботи було досягнуто поставленої мети – розроблено повноцінну, зручну у використанні, стабільну та якісну клієнтську частину вебзастосунку, що відповідає вимогам сучасного програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cheng, V. W.-S., Davenport, T. A., Johnson, D., Vella, K., Mitchell, J., & Hickie, I. B. Gamification in apps and technologies for improving mental health and well-being: Systematic review. JMIR Mental Health. – 2019. – Vol. 6(6), Article e13717 [Електронний ресурс] – URL: https://mental.jmir.org/2019/6/e13717/ (дата звернення: 10.04.2025).

2. Habitica [Електронний ресурс] – URL: https://habitica.com/ (дата звернення: 15.04.2025).

3. Trello [Електронний ресурс] – URL: https://trello.com/ (дата звернення: 15.04.2025).

4. ClickUp [Електронний ресурс] – URL: https://clickup.com/ (дата звернення: 17.04.2025).

5. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) [Електронний ресурс] – URL: https://www.iso.org/standard/35733.html (дата звернення: 30.04.2025).

6. Mozilla Developer Network. JavaScript Guide [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення 02.05.2025).

7. React Documentation. Quick Start [Електронний ресурс] – URL: https://react.dev/learn (дата звернення: 02.05.2025).

8. Nasution, Winda Suci Lestari, and Patriot Nusa. UI/UX Design Web-Based Learning Application Using Design Thinking Method. ARRUS Journal of Engineering and Technology, Vol. 1 [Електронний ресурс] – URL: https://doi.org/10.1088/1742-6596/1845/1/012015 (дата звернення: 09.05.2025).

9. The web standards model [Електронний ресурс] – URL: https://developer.mozilla.org/enUS/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model (дата звернення: 12.05.2025).

10. Асинхронність в JS [Електронний ресурс] – URL: https://medium.com/@jstify.community (дата звернення: 15.05.2025).

11. TypeScript [Електронний ресурс] – URL: https://www.typescriptlang.org/ (дата звернення: 16.05.2025).

12. Document Object Model (DOM) [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (дата звернення: 17.05.2025).

13. Vite [Електронний ресурс] – URL: https://vite.dev/ (дата звернення: 18.05.2025).

14. ESLint [Електронний ресурс] – URL: https://eslint.org/ (дата звернення: 19.05.2025).

15. Recharts [Електронний ресурс] – URL: https://recharts.org/en-US (дата звернення: 20.05.2025).

16. Google OAuth 2.0 [Електронний ресурс] – URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 20.05.2025).

17. GitHub.NureDashkoNadiia/2025_B_PI_PZPI-21-1_Dashko_N_M. URL: https://github.com/NureDashkoNadiia/2025_B_PI_PZPI-21-1_Dashko_N_M