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 зберігання, спільного доступу та перевірки якості програмного коду. Клієнтська частина
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-4

Поліна ПАВЛЕНКО
(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 ст.викл. каф. ПІ
Марія ШИРОКОПЕТЛЄВА
(посада, Власне ім'я, ПРІЗВИЩЕ)

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

2025 р.

Харківський національний університет радіоелектроніки

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Павленко Поліні Олексіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для зберігання, спільного доступу та перевірки якості програмного коду. Клієнтська частина
Затверджена наказом по університету від 19.05. 2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 05.06.2025
3. Вихідні дані до роботи Розробити клієнтську частину застосунку для зберігання та управління фрагментами програмного коду. Реалізацію здійснити з використанням мови програмування JavaScript, бібліотеки React.js та суміжних інструментів (React Query, React Hook Form, Styled Components тощо).
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	07.04.2025	*виконано*
2	Створення специфікації ПЗ	11.04.2025	*виконано*
3	Проектування ПЗ	18.04.2025	*виконано*
4	Розробка ПЗ	29.04.2025	*виконано*
5	Тестування ПЗ	23.05.2025	*виконано*
6	Оформлення пояснювальної записки	28.05.2025	*виконано*
7	Підготовка презентації та доповіді	30.06.2025	*виконано*
8	Попередній захист	02.06.2025	*виконано*
9	Нормоконтроль, рецензування	02.06.2025	*виконано*
10	Здача роботи у електронний архів	03.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	03.06.2025	*виконано*

Дата видачі завдання « 7 » « квітня » 2025р.

Здобувач_____
(підпис)

Керівник роботи_____ ст.викл. каф. ПІ ШИРОКОПЕТЛЄВА М.С.
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, с. 90, рис. 26, табл. 1, джерел 10.

ЗБЕРІГАННЯ, КОМАНДНА РОЗРОБКА, ОРГАНІЗАЦІЯ, ФРАГМЕНТ КОДУ, FRONT-END РОЗРОБКА, JAVASCRIPT, REACT

Об'єктом дослідження виступають процеси зберігання та управління фрагментами програмного коду в середовищі командної розробки програмного забезпечення.

Метою роботи є створення клієнтської частини системи, що забезпечує ефективне управління фрагментами коду, включаючи їх додавання, редагування, збереження, пошук, а також генерацію нових кодових фрагментів на основі вже наявних. Система також сприяє аналізу якості коду, підтримує спільний доступ до фрагментів та оптимізує процеси розробки нових програмних продуктів.

У результаті роботи було реалізовано клієнтську частину програмної системи для організації та управління фрагментами кодуз використанням сучасної бібліотеки React, при цьому для керування асинхронною взаємодією з сервером застосовується React Query, а для створення стилізованих та підтримуваних інтерфейсів – styled-components.

ABSTRACT

STORAGE, TEAM DEVELOPMENT, ORGANIZATION, CODE ASSET, FRONT-END DEVELOPMENT, JAVASCRIPT, REACT

The object of research is the processes of storing and managing program code assets in the team-based software development environment. Modern problems associated with code duplication, difficulty in finding the necessary assets, and the complexity of reusing already created solutions are considered.

The purpose of the work is to create a client side of the system that provides efficient management of code assets, including adding, editing, saving, searching, and generating new code assets based on existing ones. The system also facilitates code quality analysis, supports asset sharing, and optimizes the development of new software products.

The client side of the system is developed using the modern React library, with React Query used to manage asynchronous interaction with the server and styled-components to create customized and maintainable interfaces. This technology stack provides flexibility, scalability, and ease of use. The interface is adapted for teamwork on large amounts of code, with an emphasis on quick search and visibility of the structure of the saved assets.

As a result of the work, the client side of the software system was implemented to organize and manage code fragments. The developed solution is an important component of the overall system architecture and provides intuitive access to tools that greatly facilitate the work of developers, improve the quality of the final product, and reduce the time for creating new software products.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SOLID – Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion

DRY – Don't Repeat Yourself

API – Application Programming Interface

ІШІ – Штучний Інтелект

UI /UX – User Interface/ User Experience

HTTP – Hypertext Transfer Protocol

JWT – JSON Web Token

SMTP – Simple Mail Transfer Protocol

UML – Unified Modeling Language

DOM – Document Object Model

ID – Identification

ВСТУП

Сучасні вимоги до розробки програмного забезпечення вимагають від команд не лише створення якісного коду, але й ефективного управління ним. Традиційні методи організації кодової бази часто виявляються недостатніми, що призводить до дублювання фрагментів коду, зниження продуктивності та збільшення часу на розробку нових проєктів. Розробники стикаються з труднощами при повторному використанні коду, його аналізі та контролі якості, що ускладнює спільну роботу та підвищує ризик помилок.

У таких умовах виникає потреба в інструментах, які дозволяють ефективно керувати кодовою базою. Це особливо актуально для компаній, які працюють над великими проєктами та мають значні об’єми повторюваного коду. Без належної системи управління кодом розробники можуть витрачати багато часу на пошук потрібних фрагментів, перевірку їх якості та інтеграцію в нові проєкти.

Ці проблеми акцентують увагу на важливості розробленої програмної системи для зберігання та управління фрагментами коду. Метою роботи є розробка клієнтської частини системи, яка надає можливість швидкого та ефективного доступу до даних, аналізу якості коду та його повторного використання. Вона дозволятиме розробникам легко додавати, редагувати та шукати фрагменти коду, а також отримувати рекомендації щодо покращення його якості. Також система має підтримувати спільний доступ до коду всередині компанії, що сприяє ефективній командній роботі та прискоренню процесу розробки. Вона також забезпечує механізм генерації нових проєктів на основі існуючих перевірених кодових фрагментів, що спрощує створення нових програмних продуктів та підвищує загальну ефективність роботи команд розробників.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У сучасній сфері розробки програмного забезпечення спостерігається тенденція до зростання кількості та складності проєктів, що вимагає від команд розробників не лише високого рівня технічної підготовки, а й використання ефективних інструментів для організації робочого процесу. Однією з ключових проблем галузі є управління програмним кодом, зокрема його повторне використання, перевірка якості, централізоване зберігання та швидкий доступ до потрібних фрагментів.

В умовах інтенсивної розробки велика частина коду, який створюють розробники, має повторюваний характер. При цьому у багатьох компаніях відсутні зручні засоби для централізованого зберігання фрагментів коду (code assets), які вже були реалізовані, протестовані та можуть бути повторно використані [1]. Це призводить до дублювання зусиль, втрати часу та зниження продуктивності.

З іншого боку, вимоги до якості програмного коду також постійно зростають. Недостатня модульність, надмірна складність, повторюваність або порушення архітектурних принципів можуть не лише ускладнювати підтримку програмного продукту, а й негативно впливати на його стабільність та масштабованість. Принципи SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) і DRY (Don't Repeat Yourself) стають стандартом у розробницькому середовищі, однак контроль за їх дотриманням досі часто виконується вручну або з використанням обмежених за функціональністю інструментів.

Водночас розвиток штучного інтелекту та технологій обробки природної мови відкрив нові можливості для автоматичного аналізу коду. Наприклад, за допомогою моделей від OpenAI API (Application Programming Interface) стало можливим автоматично оцінювати якість коду, знаходити дублікати або недоліки у структурі, а також генерувати нові рішення на основі існуючих фрагментів. Це

дозволяє не лише поліпшити якість продукту, а й скоротити час розробки за рахунок зменшення кількості рутинних завдань.

У контексті командної роботи все більше компаній прагнуть мати єдине середовище для зберігання та доступу до коду, яке дозволяє забезпечити корпоративний обмін напрацюваннями, швидко знаходити потрібні фрагменти, ділитися рішеннями між командами та проєктами.

Втім, більшість з існуючих рішень на ринку фокусуються лише на часткових аспектах, не забезпечуючи комплексного підходу до управління фрагментами коду, їхньої перевірки та генерації нових програмних продуктів на їх основі. Цей недолік створює передумови для створення нової системи, яка поєднає переваги наявних рішень та запропонує інноваційний функціонал.

1.2 Виявлення та вирішення проблем

1.2.1 Актуальні виклики в управлінні фрагментами коду

Фрагменти коду відіграють ключову роль як повторно використовувані одиниці логіки, що значно пришвидшують та оптимізують процес створення нових функціональних рішень. Незважаючи на це, в більшості випадків розробники зберігають свої фрагменти у неструктурованому вигляді – локально на комп’ютері, в особистих документах чи через сторонні додатки, які не передбачають командної взаємодії або перевірки якості коду. Це призводить до зниження ефективності розробки, дублювання зусиль та втрати напрацьованого досвіду всередині команди.

Крім того, стрімкий розвиток штучного інтелекту відкриває нові можливості для генерації, перевірки та оптимізації коду, проте значна частина наявних інструментів або не використовують ці можливості, або реалізують їх у вузькому контексті, що не дозволяє повністю інтегрувати ШІ (Штучний Інтелект) у щоденну практику розробника.

Ще однією серйозною проблемою є обмеження в корпоративному доступі до знань. У випадку командної роботи розробники часто стикаються з тим, що

фрагменти коду залишаються в особистому доступі, що ускладнює комунікацію, перевикористання та стандартизацію коду. Відсутність централізованого середовища, яке дозволяло б легко керувати доступом, також негативно впливає на якість командної роботи та безперервність знань у межах організації.

1.2.2 Аналіз актуальних рішень

Щоб краще зрозуміти потребу в новому рішенні, варто звернутися до аналізу найближчих конкурентів, які діють у цьому сегменті ринку. Одним із таких рішень є SnippetsLab [2] – додаток, орієнтований переважно на індивідуальних користувачів, з особливим акцентом на екосистему macOS. Цей інструмент дозволяє зберігати, групувати та шукати фрагменти коду локально, однак позбавлений можливостей для командної взаємодії, а також не використовує механізми штучного інтелекту для покращення досвіду роботи з кодом. SnippetsLab не виконує автоматичної перевірки коду та не пропонує вбудованих інструментів для генерації нових рішень, що значно обмежує його функціональність у порівнянні з потенційними можливостями сучасного програмного середовища.

Іншим прикладом є Code Snippets AI [3] – більш сучасне рішення, яке вже містить елементи використання штучного інтелекту, зокрема для покращення якості коду. Цей інструмент орієнтований на командну роботу, забезпечує керування доступами до фрагментів, але не реалізує повноцінну генерацію нового коду на основі опису задач, що залишає певний розрив між інструментом і повноцінними AI-помічниками. Крім того, можливості перевірки якості коду є лише частково реалізованими і не охоплюють весь спектр практичних потреб командного середовища.

Ще одним популярним продуктом є GitHub Copilot [4], який представляє собою інтеграцію системи штучного інтелекту з середовищем розробки. Цей інструмент здатен генерувати код на основі введеного опису задачі, що суттєво розширює можливості розробника. Проте Copilot не призначений для

систематизації та збереження фрагментів коду, а також не забезпечує керованого спільного доступу до них. Відсутність централізованого сховища та обмеженість у налаштуваннях доступу роблять його менш придатним для побудови командних рішень, де важливо не лише генерувати, але й зберігати, обговорювати та вдосконалювати фрагменти.

Таким чином, усі три аналізовані рішення мають власні переваги, однак не здатні комплексно вирішити проблеми, пов’язані з якісною організацією роботи з фрагментами коду, їх генерацією, перевіркою, зберіганням та пошуком у межах командних чи відкритих середовищ. Складену візуалізацію порівняння конкурентів наведено у таблиці 1.

Таблиця 1 - Порівняння конкурентів (таблиця виконана самостійно)

Аналог	Використання ШІ для генерації програмних проєктів	Корпоративний доступ до фрагментів коду	Відкритий доступ до коду	Перевірка якості коду	Рівень складності UI / UX
SnippetsLab	–	+	+	+	високий
Code Snippets AI	–	+	–	+	помірний
GitHub Copilot	+	+	–	+	помірний

1.2.3 Концепція проєкту

Основною метою проєкту є створення системи, що поєднує кращі сторони існуючих рішень, при цьому додаючи нові інноваційні можливості, які забезпечують комплексну роботу з фрагментами коду. На відміну від конкурентів, програмне забезпечення для зберігання та менеджменту фрагментів коду SNIP&KEEP буде дозволяти не тільки зберігати код, але й аналізувати його якість у момент компіляції, використовуючи нейромережеві моделі. Така перевірка не лише знизить ймовірність помилок, але й сприятиме дотриманню стилістичних та архітектурних вимог.

Додатково, проєкт передбачає потужну функцію генерації проєктів (нових фрагментів коду) на основі введеного запиту, що зробить його аналогом Copilot у контексті роботи з фрагментами, але з додатковими можливостями збереження та пошуку. Інтерфейс користувача буде розроблений з урахуванням UX-досліджень, що дозволить забезпечити інтуїтивність та швидкість взаємодії.

Окремо варто відзначити систему керування доступами, яка дозволить налаштовувати рівні видимості фрагментів – від корпоративного до публічного, що повністю відповідає потребам різних типів команд та поодиноких розробників. Важливою складовою системи є семантичний пошук, заснований на векторному поданні коду, що дозволить знаходити схожі фрагменти навіть тоді, коли синтаксис чи структура відрізняється.

Таким чином, SNIP&KEEP прагне стати повноцінною екосистемою для роботи з кодом, яка зможе враховувати як технічні, так і організаційні потреби користувачів. Завдяки цьому проєкт не лише зможе усунути існуючі обмеження, але й відкриє нові підходи до збереження, повторного використання та вдосконалення фрагментів коду в умовах командної та індивідуальної розробки.

1.2.4 Визначення цільової аудиторії

Розробка програмного продукту SNIP&KEEP ґрунтується не лише на виявленні технологічних прогалин, а й на чіткому розумінні потреб конкретних категорій користувачів. Саме детальне визначення цільової аудиторії дозволяє сформувати функціональні вимоги, інтерфейсні рішення та загальну логіку взаємодії з додатком.

Основною групою користувачів проєкту є розробники програмного забезпечення – як індивідуальні фахівці, так і учасники командної розробки. Для них важливим є збереження робочих фрагментів коду, які часто використовуються у різних проєктах, із можливістю їх подальшого швидкого пошуку, повторного використання та редагування. Така аудиторія цінує зручність, швидкодію та інтуїтивність інтерфейсу, а також наявність функцій автоматичної перевірки якості

коду, яка допомагає уникнути повторюваних помилок і підтримувати чистоту стилю в межах спільного коду.

Іншою важливою групою користувачів є команди розробників у межах невеликих компаній або стартапів, які зацікавлені у централізованому доступі до спільної бази фрагментів коду. У таких умовах особливої ваги набуває можливість швидкого пошуку рішень, повторного використання типових конструкцій та забезпечення єдиного стилю програмування в межах команди. Застосування спільного інструменту для організації знань дозволить підвищити ефективність командної роботи та мінімізувати втрати часу на дублювання зусиль

Окрему нішу становлять розробники-початківці або студенти, які потребують доступу до прикладів реалізації певних алгоритмів, структур даних чи типових завдань [5]. Для них система SNIP&KEEP стане не лише інструментом збереження коду, а й освітньою платформою, яка дозволить шукати фрагменти за ключовими словами або логікою задачі, оцінювати якість та порівнювати альтернативні реалізації. Такий підхід сприятиме розвитку навичок аналізу та критичного мислення у роботі з кодом.

Також варто згадати технічних лідерів команд, архітекторів та менторів, які потребують ефективного інструменту для впровадження стандартів кодування, поширення шаблонів рішень та контролю за якістю фрагментів, що використовуються командою. Завдяки функціям перевірки SNIP&KEEP зможе слугувати механізмом внутрішньої стандартизації та контролю відповідності практик командним вимогам.

Таким чином, цільова аудиторія SNIP&KEEP охоплює широкий спектр фахівців у сфері розробки програмного забезпечення – від новачків до досвідчених експертів, від індивідуальних користувачів до організованих команд. Така широка орієнтація надає проєкту гнучкості та перспективності, дозволяючи адаптувати функціональність під конкретні потреби кожного користувача, водночас зберігаючи цілісність платформи.

1.3 Постановка задачі

Метою кваліфікаційної роботи є створення програмного забезпечення SNIP&KEEP – ефективної системи для організації, збереження та повторного використання фрагментів програмного коду. Сучасні розробники постійно стикаються з необхідністю повторювати одні й ті самі рішення, що призводить до втрати часу, зниження ефективності та порушення принципів якості програмування. Проєкт покликаний вирішити цю проблему шляхом впровадження зручного сервісу, що дозволить зберігати типові рішення, аналізувати їхню якість, швидко здійснювати пошук потрібного коду, а також підтримувати командну роботу та обмін напрацюваннями.

Ідея полягає не лише в тому, щоб забезпечити користувача базовим сховищем для фрагментів, але й запропонувати інтелектуальні можливості пошуку, систематизації та перевірки коду. Завдяки використанню сучасних інструменті, що відкривають доступ до потужних моделей штучного інтелекту, проєкт має на меті автоматизувати перевірку якості програмного коду, допомагати уникати дублювання функціональності та сприяти кращому дотриманню принципів структурованої розробки.

Особливу увагу в межах реалізації системи слід приділити комфорту та логічності взаємодії користувача з інтерфейсом, підтримці командної взаємодії між розробниками, а також можливості масштабування системи відповідно до зростаючих потреб користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Формування вимог

Функціональні задачі, які повинна вирішувати система SNIP&KEEP, випливають із загальної мети та визначених потреб цільової аудиторії. Передусім, система повинна дозволяти зберігати фрагменти програмного коду разом із супровідною інформацією, зокрема мовою програмування, описом, тега́ми та зазначенням контексту використання. Це дозволить організувати базу знань розробника, що структурована за різними критеріями.

Користувач повинен мати можливість створювати персональні колекції коду, редагувати вже додані фрагменти, здійснювати пошук серед наявних записів за ключовими словами. Особливо важливим є впровадження механізму пошуку схожих фрагментів, що дозволить уникати дублювання вже реалізованих рішень. У цьому контексті система повинна інтегрувати алгоритми векторного представлення коду, які здатні зіставляти логічно подібні рішення.

Інтеграція з зовнішніми інструментами, зокрема OpenAI API, дозволятиме системі аналізувати якість коду за визначеними принципами, такими як SOLID та DRY. Крім того, передбачається можливість спільного користування колекціями, що включає надання доступу до фрагментів іншим користувачам в межах однієї компанії.

Таким чином, функціональні задачі системи полягають у створенні повноцінного інструменту для індивідуальної та колективної роботи з фрагментами коду, з акцентом на швидкість, гнучкість і точність виконання операцій з ними.

На додаток до функціональності, система повинна відповідати низці нефункціональних вимог, які забезпечують її стабільну, безпечну та ефективну роботу у реальних умовах експлуатації:

– продуктивність: система має забезпечувати комфортну взаємодію користувача з інтерфейсом навіть у випадках очікування відповіді від

сервера протягом тривалого часу (до 50 секунд). У зв’язку з цим, особливу увагу має бути приділено відображенню станів завантаження, прогресу, повідомлень про очікування та обробки помилок. Важливо, щоб запити до серверної частини, включно з пошуком, фільтрацією та взаємодією з фрагментами коду, не призводили до блокування інтерфейсу, а система залишалася чуйною для користувача.

– надійність: платформа повинна гарантувати збереження даних, забезпечуючи цілісність і відмовостійкість у разі помилок або збоїв;
– масштабованість: архітектура розробки повинна передбачати можливість подальшого розширення – як у частині обсягу даних, так і функціональних можливостей – без необхідності кардинальної зміни базової структури системи.
– безпека: система повинна забезпечувати належний захист персональних даних користувачів та обмежувати доступ до приватних колекцій коду відповідно до ролі користувача. Для реалізації цих вимог передбачається використання механізму аутентифікації та авторизації на основі JSON Web Token (JWT). Система має видавати токен після успішної перевірки облікових даних, що дозволить користувачеві підтверджувати свою автентичність під час кожного запиту. Токени повинні бути підписані, мати обмежений термін дії та передаватися через захищені канали зв’язку HTTPS (Hypertext Transfer Protocol). Очікується, що система також передбачатиме можливість оновлення токена (refresh token), щоб уникнути повторної автентифікації та водночас зберігати безпечний контроль доступу. Такі заходи мають відповідати сучасним вимогам інформаційної безпеки та забезпечувати захист від несанкціонованого доступу;
– зрозумілий інтерфейс: інтерфейс користувача повинен бути логічно організованим, легким для освоєння, забезпечуючи чітке виконання базових дій навіть для недосвідчених користувачів;

– гнучкість доступу: важливою вимогою до системи є її доступність через різні типи пристроїв, включаючи настільні комп’ютери, ноутбуки, планшети та смартфони. Це забезпечить гнучкість у використанні програмного продукту в різноманітних робочих середовищах – як у офісі, так і під час віддаленої роботи. Особливу увагу має бути приділено сумісності з різними браузерами: інтерфейс системи має коректно відображатися та функціонувати у сучасних версіях популярних веб-браузерів, зокрема Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Такий підхід дозволить зробити систему більш універсальною та зручною для максимально широкого кола користувачів.

Клієнтську частину слід реалізувати за допомогою бібліотеки React.JS, яка забезпечує створення динамічного та інтерактивного інтерфейсу користувача. Компонентна архітектура React надасть змогу розбити інтерфейс на незалежні елементи, що спростить його тестування, повторне використання та подальше обслуговування.

Серверну частину треба створити на основі платформи .NET, що забезпечить високу продуктивність, стабільність і безпеку. Серверна частина відповідає за обробку запитів, реалізацію бізнес-логіки, взаємодію з базою даних, а також інтеграцію з зовнішніми сервісами, зокрема з OpenAI API та SMTP (Simple Mail Transfer Protocol)-сервером.

Для зберігання фрагментів коду та пов’язані метадані слід використовувати CosmosDB – розподілена NoSQL база даних, яка забезпечує масштабованість, високу доступність і швидку обробку запитів. Така база є зручною для зберігання неструктурованих і напівструктурованих даних, що особливо актуально для задач, пов’язаних з обробкою коду.

Для реалізації можливостей штучного інтелекту необхідно використати OpenAI API, який інтегрується у серверну частину системи та виконує аналіз фрагментів коду на предмет відповідності принципам якісного програмування,

виявлення повторів і семантичної подібності. Це дозволить користувачам системи отримувати рекомендації щодо збережених рішень.

Для комунікації з користувачами слід використати SMTP-сервер, за допомогою якого буде реалізовано функціональність надсилання службових повідомлень, таких як підтвердження електронної пошти або повідомлення про зміну пароля.

Розробка візуальної частини інтерфейсу треба використати інструмент Figma – сучасну платформу для створення прототипів та дизайн-макетів. Завдяки цьому інструменту буде забезпечено узгодженість інтерфейсу, високу якість візуального оформлення та зручність взаємодії користувача з системою.

Процес розробки слід організувати згідно з принципами Kanban – гнучкої методології управління проєктами, що передбачає поступову реалізацію задач із постійним моніторингом їхнього виконання. Використання Kanban сприяє прозорості командної роботи, адаптації до змін у вимогах і своєчасному виявленню та усуненню перешкод у процесі розробки.

2.2 Аутентифікація та облікові записи

Першочергову роль у системі відіграє модуль аутентифікації, що має дозволяти здійснювати повноцінний процес реєстрації нового користувача. У рамках цього функціоналу передбачається можливість створення облікового запису з підтвердженням електронної адреси за допомогою email-верифікації. Це сприятиме не лише валідації користувачів, а й безпечній організації доступу до персоналізованого функціоналу. Вхід до системи має виконуватися за допомогою логіну й пароля, з подальшою можливістю відновлення доступу у випадку втрати пароля. Усі операції з обліковими записами мають бути реалізовані з дотриманням актуальних стандартів безпеки, включно з перевіркою автентичності запитів та захистом персональних даних.

2.3 Функціональність для неавторизованих користувачів

Навіть не будучи зареєстрованим користувачем, відвідувач сайту має отримувати доступ до базової частини функціоналу. Це включає дозвіл гостям на перегляд загальнодоступних фрагментів коду. Сторінка перегляду фрагментів має включати відображення базових метаданих, таких як назва, мова програмування та теги та опис. Система має дозволяти фільтрувати ці фрагменти за тегами, задля спрощення перегляду великої кількості матеріалів. Крім цього, має бути реалізовано текстовий пошук за назвою, щоб надати змогу швидко знаходити потрібний запис без необхідності поглибленої навігації. Такий підхід дозволить новим користувачам оцінити користь системи ще до створення облікового запису.

2.4 Можливості зареєстрованих користувачів

Зареєстрований користувач має отримувати доступ до всього функціоналу, описаного вище, та додатково – можливість створювати, редагувати й видаляти власні фрагменти коду. У процесі створення користувач повинен мати можливість задавати назву, обирати мову програмування, призначати теги, додавати опис, редагувати код, додавати файли та папки та зберігати цей фрагмент у своєму особистому сховищі. Кожен фрагмент має асоціюватися з конкретним обліковим записом, що надасть змогу користувачеві централізовано керувати своєю базою знань. Крім того, система має дозволяти в будь-який момент повернутися до редагування збережених записів або остаточно видалити ті, що втратили актуальність. Перегляд, фільтрація й пошук мають здійснюватися як серед особистих, так і загальнодоступних фрагментів, із чітким розмежуванням джерела походження.

2.5 Функціональність компаній та командної взаємодії

Окрема категорія функціоналу має бути реалізована для корпоративних облікових записів (компаній). Такі користувачі мають володіти усіма можливостями звичайних облікових записів, а також отримувати доступ до

спільної бази фрагментів компанії. Система має дозволяти переглядати, шукати й фільтрувати кодові записи, які були позначені як корпоративні, або завантажені іншими учасниками цієї компанії. Така модель співпраці значно спростить повторне використання рішень усередині організації.

Ключовим розширенням корпоративного функціоналу має бути можливість створення проєктів на основі наявних фрагментів. У цьому процесі користувачі повинні мати можливість звернутися до штучного інтелекту із запитом, що формує скомбінований кодовий фрагмент на основі наявних шаблонів і метаданих. Згенерований фрагмент має проходити аналіз відповідності принципам DRY і SOLID, що дозволить забезпечити його якість, а також компілюватися у реальному часі. Має бути можливість завантаження кінцевого результату у вигляді архіву .zip, що міститиме увесь код проєкту.

Окрім технічного функціоналу, система має дозволяти учасникам компанії переглядати інформацію про організацію, а адміністраторам – керувати ролями користувачів і контролювати список учасників. Ці інструменти мають бути створені для полегшення внутрішньої координації й забезпечення структурованості командної взаємодії.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування клієнтської частини проєкту

Проєктування клієнтської частини веб-застосунку SNIP&KEEP має ключове значення, оскільки саме вона забезпечує основну взаємодію користувача із системою. Завдяки інтуїтивно зрозумілому та візуально привабливому інтерфейсу користувачі зможуть ефективно створювати, шукати та керувати фрагментами коду. Значну увагу було приділено принципам UI/UX дизайну, що дозволяє гарантувати позитивний досвід користувача (user experience), високий рівень доступності та логічну послідовність у розміщенні елементів інтерфейсу.

Процес проєктування клієнтської частини веб-застосунку розпочався з глибокого аналізу потреб цільової аудиторії, функціональних сценаріїв використання системи та дослідження існуючих рішень у сфері менеджменту кодових фрагментів. Визначення ключових очікувань користувачів – зокрема простоти використання, логічної структури інтерфейсу та зручного доступу до основного функціоналу – стало фундаментом для побудови візуальної та логічної структури клієнтської частини.

На першому етапі було зосереджено увагу на формуванні загальної концепції інтерфейсу, яка б забезпечувала зрозумілу навігацію, послідовність користувацького досвіду та візуальну узгодженість між сторінками. Було визначено набір основних ролей у системі та побудовано сценарії взаємодії для кожної з них.

Щоб визначити основні сценарії взаємодії користувачів із системою, було побудовано UML (Unified Modeling Language) діаграму прецедентів (рис. 1), яка відображає дії доступні для кожного типу користувача: гість, зареєстрований користувач, працівник та адміністратор компанії відповідно.

Рисунок 1 – UML діаграма прецедентів (рисунок виконаний самостійно)

На діаграмі прецедентів зображено ключові сценарії взаємодії користувачів із системою зберігання та менеджменту фрагментів коду. В основі моделі лежить ієрархія користувацьких ролей: гість, користувач, працівник компанії та адміністратор компанії. Кожна роль відображає певний рівень доступу до функціональних можливостей системи, що дозволяє реалізувати гнучке управління доступом та створює чітку логіку взаємодії.

На найнижчому рівні доступу перебуває гість – користувач, який не зареєстрований у системі. Йому доступні базові можливості: перегляд публічних фрагментів коду, фільтрація за тегами та пошук. Крім того, гість може переглядати деталі окремих фрагментів, такі як: повноцінна файлова структура проєкту, опис та іншу допоміжну інформацію. Щоб отримати більше можливостей, гість може зареєструватися або увійти до системи. У процесі входу передбачено додаткові опції, зокрема підтвердження електронної пошти та відновлення пароля – ці функції реалізовані як розширення до базового сценарію входу.

Після реєстрації користувач отримує доступ до персоналізованої взаємодії із системою. Зокрема, йому відкривається можливість створення власних фрагментів коду. Цей процес включає введення назви, та вибір мови програмування для створення початкового фрагменту. Усі створені фрагменти доступні для подальшого редагування або видалення. Окрім особистих фрагментів, користувач може ініціювати створення компанії – окремої організаційної одиниці, що стане доступною для інших учасників.

На роль працівника компанії накладаються додаткові можливості. Окрім функціоналу звичайного користувача, працівник має доступ до перегляду фрагментів коду, створених учасниками його компанії. Він також може створювати проєкти – це функціональність, що об’єднує декілька фрагментів у єдину структуру за допомогою формулювання запиту до штучного інтелекту. Результатом цього процесу є скомбінований проєкт, доступний для перегляду та взаємодії. Система дозволяє перевірити згенерований код на відповідність принципам DRY/SOLID, а також здійснити компіляцію проєкту. За потреби, готовий проєкт можна завантажити у вигляді архіву.

Найвищий рівень доступу має адміністратор компанії. Він успадковує весь функціонал працівника, але додатково отримує змогу керувати компанією: переглядати інформацію про неї, додавати та видаляти учасників, змінювати їх ролі. Кожен із цих сценаріїв має розширення, які забезпечують гнучкість у керуванні та гарантують зручність взаємодії з інтерфейсом.

Таким чином, діаграма відображає комплексну логіку поведінки користувачів у системі, враховуючи всі рівні доступу. Це дозволяє створити чітке уявлення про функціональні вимоги до інтерфейсу користувача та служить основою для подальшого UI/UX-дизайну. Ретельна деталізація сценаріїв гарантує узгодженість між клієнтською і серверною частинами у процесі реалізації програмного забезпечення.

Продовженням проєктування клієнтської частини є візуалізація взаємодії користувачів із системою керування фрагментами коду. UML діаграма компонентів відображає структуру програмної системи на рівні компонентів та їх взаємозв'язків [6]. Розроблену UML діаграма компонентів наведено на рисунку 2.

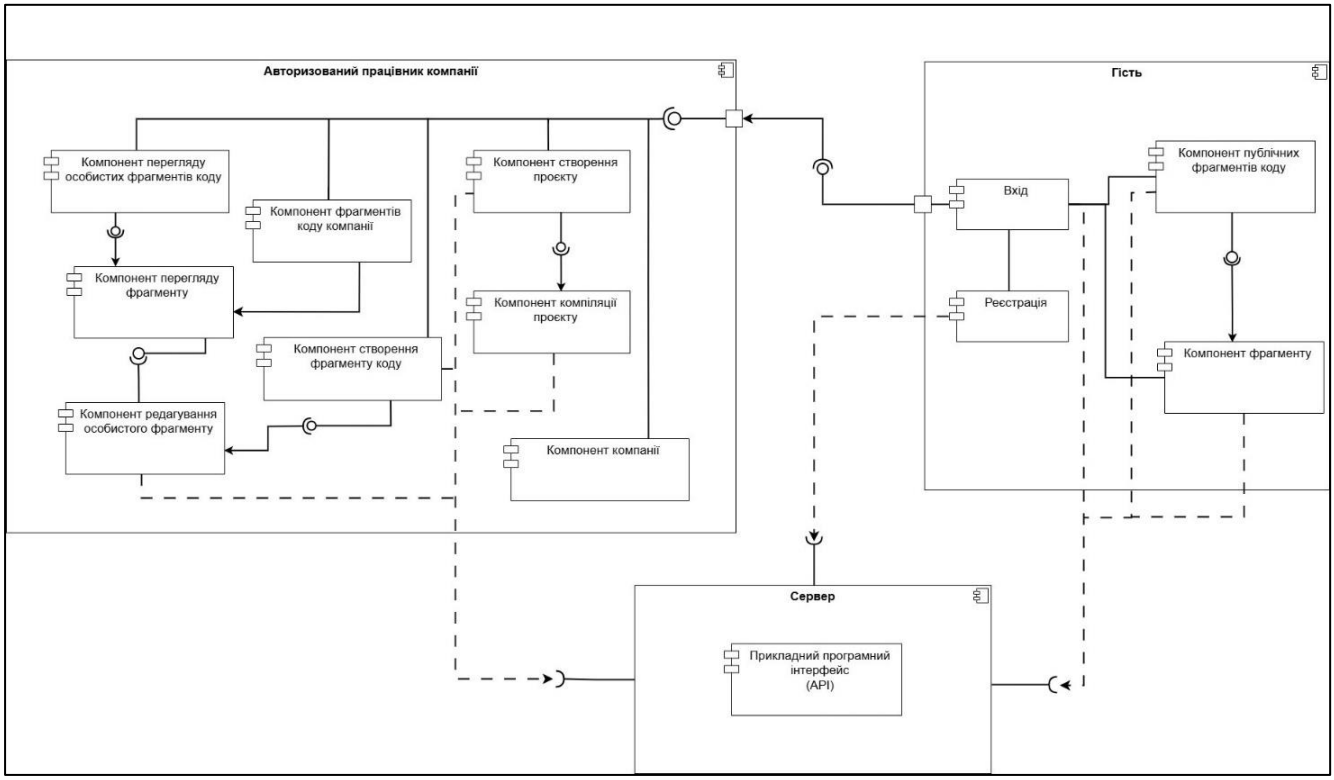

Рисунок 2 – UML діаграма компонентів (рисунок виконаний самостійно)

У розробленій схемі кожен компонент відображає відповідний компонент(або сторінку) реалізованого клієнтського застосунку залежно від рівня доступу – гість або авторизований працівник компанії. Кожна роль має свій набір функцій, що реалізовані у вигляді окремих клієнтських компонентів.

Гість має можливість переглядати публічні фрагменти коду, у разі бажання отримати розширений доступ, система пропонує здійснити реєстрацію або авторизацію, що переводить користувача в інший сценарій роботи.

Авторизований працівник компанії після входу в систему отримує доступ до ширшого функціоналу. Інтерфейс дозволяє йому працювати з особистими фрагментами коду (переглядати, редагувати), а також із фрагментами компанії, комбінувати на їх базі проєкти та ініціювати процес компіляції та аналізу коду.

Усі дії користувача, зокрема запити на перегляд, створення або редагування фрагментів, обробляються через прикладний програмний інтерфейс (API), який є сполучним зв’язком між клієнтською частиною та бізнес логікою.

Таким чином, дана діаграма не лише деталізує компоненти клієнтської частини, а й демонструє повну логіку їх взаємодії у межах системи, забезпечуючи цілісне бачення архітектури веб застосунку.

Щоб забезпечити гнучку, масштабовану та структуровану архітектуру клієнтської частини, було побудовано діаграму пакетів (див. рис. 3), яка відображає логічний поділ функціональності за окремими модулями (папками) та взаємозв’язки між ними. Діаграма демонструє основні пакети застосунку та їхні залежності. Розглянемо їх детальніше.

Головний компонент App є центральною точкою входу до клієнтської частини програмного забезпечення. У цьому компоненті здійснюється підключення основних модулів, зокрема контексту, сторінок, а також інших базових пакетів. Саме з цього компонента починається ініціалізація інтерфейсу користувача.

Модуль Контекст реалізує механізм глобального керування станом за допомогою Context API, що входить до складу бібліотеки React. Це дозволяє централізовано керувати спільними даними (наприклад, інформацією про поточного користувача або тему оформлення), забезпечуючи доступ до них з будь-якого компонента без необхідності передавання через проміжні рівні. Контекст широко застосовується у хуках і компонентах для оптимізації логіки взаємодії.

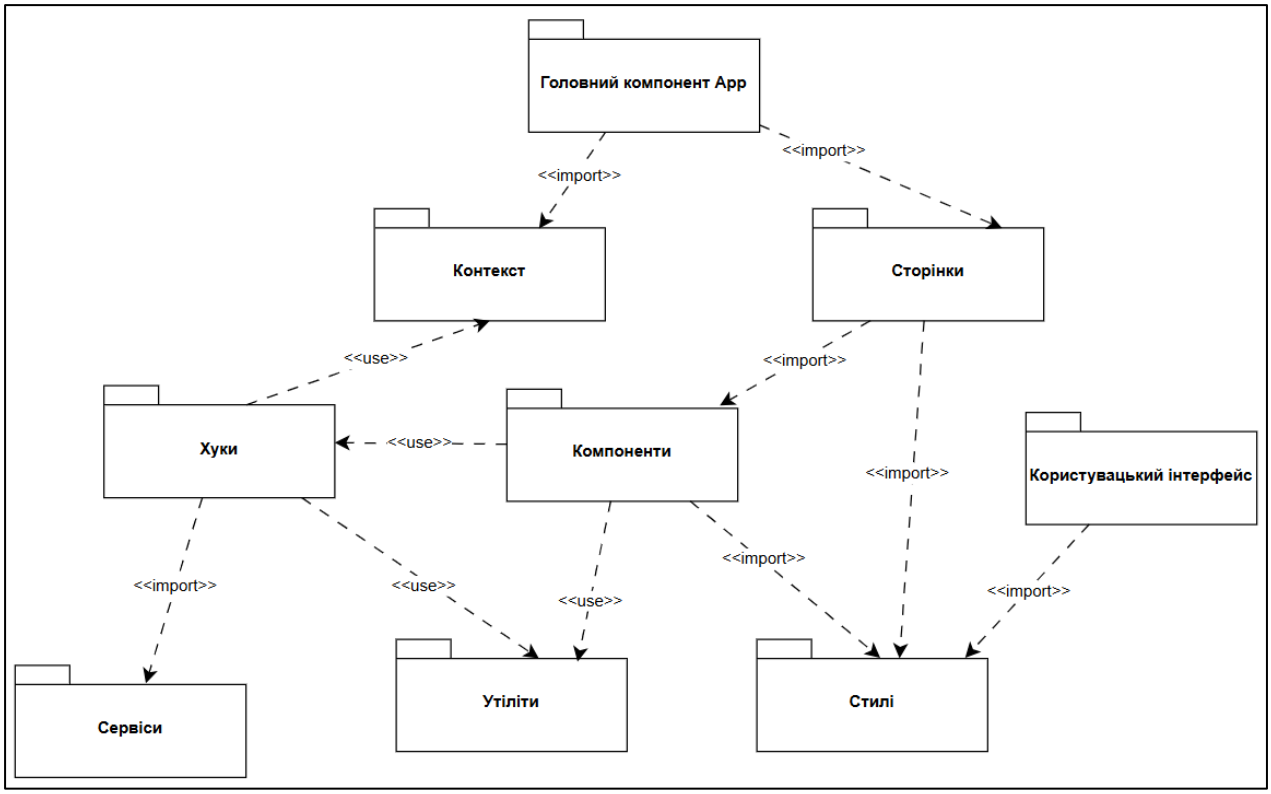

Рисунок 3 – UML діаграма пакетів (рисунок виконаний самостійно)

Модуль Сторінки містить окремі візуальні представлення програмного інтерфейсу, кожне з яких відповідає певному сценарію взаємодії з користувачем (наприклад, сторінки автентифікації, профілю, панелі керування тощо). Кожна сторінка формується із компонентів та UI-елементів, що забезпечує її структурну цілісність. Для оформлення сторінок також використовуються відповідні стилі.

Модуль Компоненти включає функціональні частини інтерфейсу, які відповідають за реалізацію окремих логічних блоків. Вони можуть бути як елементарними (наприклад, кнопки чи поля вводу), так і складеними (наприклад, компонент фрагменту коду, що містить компоненти файлу та папки). Компоненти мають чітко визначену функціональність, а також можуть бути повторно використані в різних частинах застосунку.

Модуль Користувацький інтерфейс містить набір стандартизованих елементів оформлення, що активно використовуються для побудови сторінок застосунку. Серед них: кнопки, модальні вікна, поля введення, форми тощо.

Завдяки повторному використанню та єдиному стилю ці компоненти сприяють забезпеченню послідовного візуального оформлення всього інтерфейсу.

Модуль Хуки включає функції, побудовані на основі механізмів React (наприклад, useState, useEffect) та розширені у вигляді спеціалізованих користувацьких хуків. Вони інкапсулюють повторювану логіку, забезпечуючи зручне використання сервісів, утиліт і контекстів у межах різних компонентів. Застосування хуків підвищує структурованість та гнучкість програмного коду.

Модуль Сервіси відповідає за реалізацію логіки взаємодії з зовнішніми джерелами даних, зокрема API. У ньому розміщуються функції, що здійснюють HTTP-запити, обробляють відповіді сервера, а також керують автентифікаційними токенами. Сервіси не залежать від візуальних компонентів, що дозволяє повторно їх використовувати у будь-якій частині проєкту.

Модуль Утиліти включає допоміжні функції, призначені для обробки даних, перевірки введених значень, форматування інформації, роботи з локальним сховищем тощо. Вони полегшують виконання специфічних завдань і сприяють підвищенню повторної придатності коду та розділенню відповідальностей.

Модуль Стилі містить набір як глобальних, так і модульних стилів, які визначають єдиний візуальний стиль клієнтської частини. Це можуть бути змінні для кольорової палітри, типографіки, відступів, а також специфічні стилі для окремих компонентів. Послідовне використання стилів забезпечує візуальну цілісність і сприяє легшій підтримці зовнішнього вигляду застосунку.

Діаграма чітко розмежовує відповідальності між модулями, що спрощує обслуговування, тестування і масштабування клієнтської частини проєкту.

Архітектура розроблюваної системи має передбачаєти чітке розділення відповідальностей між клієнтською (front-end) та серверною (back-end) частинами, що відповідає принципам сучасної веб-розробки. Спроєктована структура дозволить досягти високої гнучкості, масштабованості та зручності у підтримці та розширенні системи. У сукупності обрана архітектура та технології забезпечать створення надійної, зручної у використанні та масштабованої системи, здатної

ефективно виконувати задачі з управління фрагментами програмного коду як для індивідуальних користувачів, так і для команд.

3.2 Приклади найцікавіших алгоритмів та методів

У процесі проєктування клієнтської частини програмної системи SNEEP&KEEP передбачається реалізація низки алгоритмів, що забезпечуватимуть коректну та зручну взаємодію користувача з інтерфейсом, а також автоматизацію обробки даних.

3.2.1 Рекурсивний алгоритм побудови файлового дерева

Одним із ключових елементів функціоналу інтерфейсу користувача буде візуалізація структури директорій та вкладених кодових фрагментів у вигляді ієрархічного дерева. Для реалізації цієї функції доцільно використати рекурсивний підхід, який дозволяє ефективно обробляти дерево будь-якої глибини.

Вхідні дані матимуть вигляд масиву об’єктів, де кожен об’єкт містить унікальний ідентифікатор елемента, ідентифікатор батьківського елемента та інформацію про тип (файл чи директорія). Алгоритм передбачає побудову дерева наступним чином:

- визначити вузел, який не має батьківських елементів (parentId = null). Це – кореневий елемент дерева;
- для кожного такого елемента рекурсивно знаходити дочірні вузли, використовуючи їх parentId, і повторювати процес;
- формувати дерево у вигляді вкладених об’єктів (чи компонентів – у контексті React).

Такий підхід дозволить динамічно будувати дерево з будь-якою кількістю рівнів вкладеності, що особливо важливо для реалістичного подання великої кількості організованих фрагментів коду.

3.2.2 Розбір JWT токена для отримання інформації про користувача

У межах розробки клієнтської частини системи передбачається використання токенів JWT для реалізації механізмів автентифікації та авторизації користувачів. Одним із ключових етапів є розбір токена безпосередньо на клієнті з метою отримання інформації про поточного користувача.

JWT є компактною формою передачі зашифрованих даних між сторонами у форматі JSON. Він складається з трьох частин, розділених крапками – Header, Payload, Signature. Для клієнта важливо саме значення Payload, оскільки саме в ньому міститься інформація про користувача: його ідентифікатор, email, роль, час створення токена, термін дії тощо.

Алгоритм розбору та перевірки JWT має наступний вигляд:

- зчитати токени доступу та оновлення з локального сховища клієнта;
- якщо хоча б один із токенів відсутній – припинити виконання та повернути null;
- провести декодування токена доступу (Access Token) за допомогою стандартного декодера JWT;
- витягти необхідні атрибути користувача з зарезервованих полів (зокрема, name, id, email, role, exp);
- перевірити час дії токена (поле exp) у порівнянні з поточним UNIX-часом:
  1) якщо токен дійсний – повернути об’єкт користувача;
  2) якщо токен застарілий – ініціювати запит на оновлення токена доступу (Access Token) за допомогою токена оновлення (Refresh Token).
- повторно декодувати оновлений токен та повернути об’єкт користувача.

Важливо зазначити, що під час проєктування системи безпеки враховується той факт, що розбір токена на клієнті не гарантує його валідність – ця перевірка виконується на сервері за допомогою криптографічного підпису. Проте на клієнті це дозволяє миттєво отримати доступ до базової інформації для формування інтерфейсу (наприклад, виведення імені користувача або визначення доступних розділів системи за роллю).

3.3 Створення UI / UX дизайну системи

3.3.1 Загальні принципи проєктування інтерфейсу

Проєктування користувацького інтерфейсу системи базується на сучасних принципах UI/UX дизайну, з урахуванням зручності використання, доступності, логічної структури та естетичної привабливості. Основною метою є створення інтуїтивно зрозумілого, привабливого та функціонального інтерфейсу, який забезпечує ефективну взаємодію користувача із системою.

Було прийнято наступні рішення щодо візуального стилю:

- кольорова схема: інтерфейс має бути виконано в темній кольоровій гамі, що забезпечить зниження навантаження на очі під час тривалого використання [7]. Акцентні кольори мають бути застосовано для кнопок, повідомлень та активних елементів;
- типографіка: має бути використано сучасні шрифти з чіткою ієрархією заголовків, підзаголовків і основного тексту. Всі тексти повинні мати достатній контраст і читабельність;
- іконографія: мають застосовуватись зрозумілі іконки для навігації, дій та інформаційних підказок, що підсилить візуальне сприйняття.

Було прийнято наступні рішення щодо компонування та структури:

- розподіл інтерфейсу: усі екрани (сторінки) мають бути поділені на логічні секції: бокове меню (за наявності), головна область вмісту та навігаційні панелі. Це забезпечить послідовну побудову інтерфейсу на всіх сторінках;
- навігація: має бути забезпечено чітку навігаційну ієрархію з урахуванням різних ролей користувачів (гість, зареєстрований користувач, представник компанії). Основна навігація має бути розташована на видимих і звичних для користувача місцях;
- уніфікація компонентів: повторно використовувані компоненти (кнопки, поля вводу, картки, модальні вікна) мають бути уніфіковані за стилем, розміром та логікою взаємодії.

Було прийнято наступні рішення щодо адаптивності та доступності:

- адаптивний дизайн: інтерфейс має бути зпроєктовано з урахуванням адаптивності для різних розмірів екранів (стаціонарні комп’ютери, ноутбуки, планшети, телефони). Основні елементи мають автоматично перебудовуватись залежно від доступного простору;
- доступність (accessibility): має бути використано контрастні кольори, текстові підказки та інтуїтивні елементи керування, що сприятиме зручному користуванню системою особами з обмеженими можливостями.

Було прийнято наступні рішення щодо послідовності взаємодії:

- очікувана поведінка елементів: усі інтерактивні елементи мають поводяться згідно з очікуваннями користувача (наприклад, натискання кнопки викликає відповідну дію або анімацію завантаження);
- зворотний зв’язок: після дії користувача система має надавати зворотний зв’язок (наприклад, спливаючі повідомлення про успіх або помилку), що забезпечить прозорість процесів.

3.3.2 Інтерфейс неавторизованого користувача (гостя)

Інтерфейс неавторизованого користувача (гостя) є першим рівнем взаємодії з системою. Його мета – надати відвідувачу можливість ознайомитися з доступним функціоналом, переглянути публічний контент та здійснити базову навігацію без обов’язкової авторизації. Дизайн користувацького інтерфейсу було розроблено за допомогою векторного онлайн-сервісу розробки інтерфейсів та прототипування Figma [8].

Гості мають можливість переглядати публічно доступні фрагменти коду у вигляді структурованого списку або у формі карток. Кожен із цих ресурсів містить короткий фрагмент коду, супровідну інформацію, метадані, такі як теги, автор тощо. При натисканні на обраний ресурс, відбувається перехід на його окрему

сторінку, де можна детальніше ознайомитися з його вмістом, прочитати опис та переглянути повний код фрагменту (рис. 4).

Інтерфейс передбачає мінімалістичне меню навігації, у якому представлені лише основні елементи, необхідні для швидкого доступу до головної сторінки, списку ресурсів, а також кнопки для входу та реєстрації. Елементи, що потребують авторизації, недоступні для взаємодії. Якщо гість намагається скористатися функціями, які призначені для зареєстрованих користувачів, система повідомляє про необхідність входу до облікового запису.

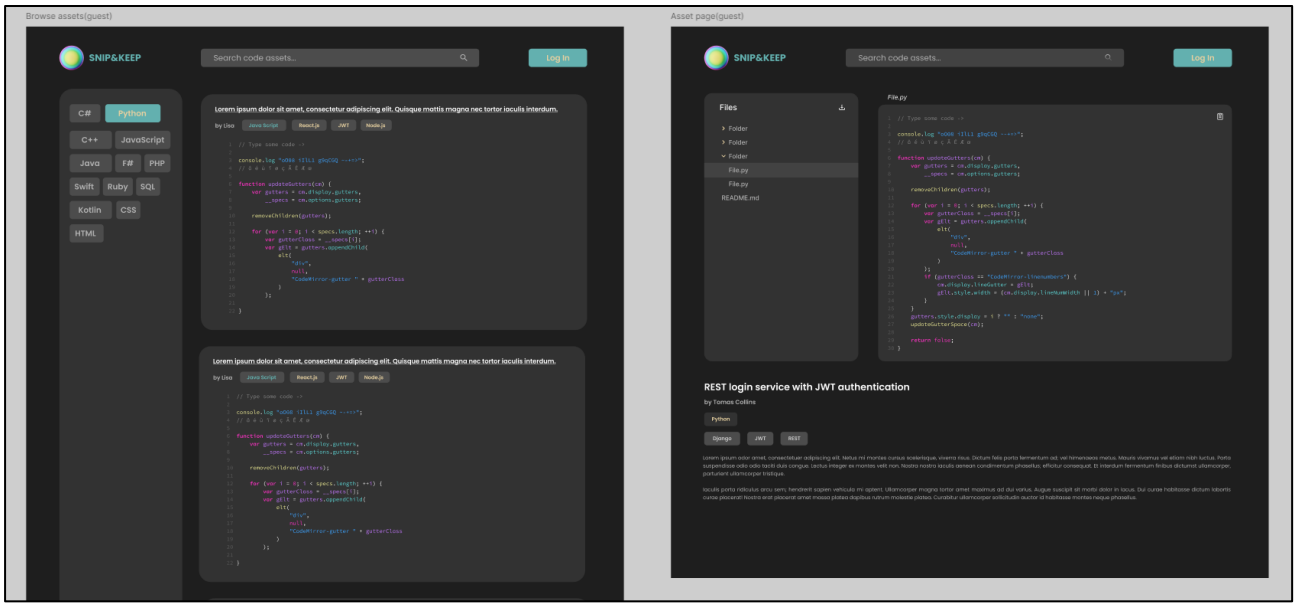

Рисунок 4 – Дизайн інтерфейсу неавторизованого користувача (рисунок виконаний самостійно)

Дизайн цього інтерфейсу орієнтований на інформативність, безпеку та продуктивність. Усі елементи виконано у єдиному стилі, що відповідає загальному візуальному оформленню застосунку. Інтерфейс не дозволяє змінювати або зберігати будь-які дані, а всі дії обмежуються лише читанням доступного вмісту. Це дозволяє уникнути несанкціонованого доступу до приватних частин системи та гарантує стабільність функціонування навіть при високому навантаженні.

Сторінки для входу, реєстрації та зміни пароля були спроектовані з чіткою структурою та наочними підписами, що полегшить користувачам заповнення необхідних даних (рис. 5).

3.3.3 Інтерфейс авторизованого користувача

Інтерфейс авторизованого користувача є другим рівнем взаємодії з системою. Його мета – надати користувачу розширений доступ до функціоналу, що включає у створення та менеджмент особистих кодових фрагментів.

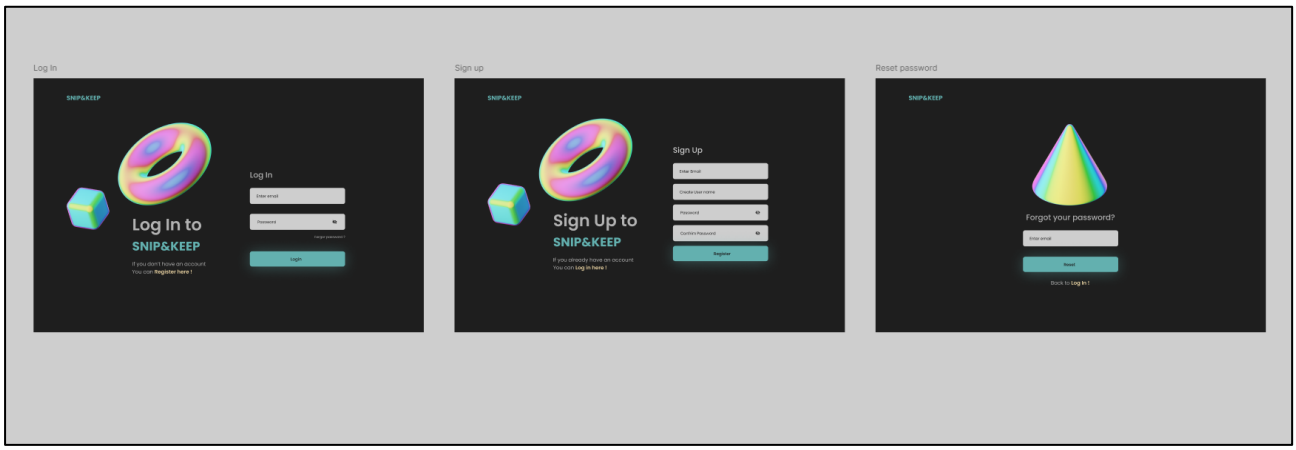

Рисунок 5 – Дизайн інтерфейсу авторизації (рисунок виконаний самостійно)

Для забезпечення покращеного доступу та управління користувацькими правами, дизайн елементів навігації та шапки (header) застосунку передбачає інтеграцію спеціалізованих розширених полів навігації (рис. 6) [9]. Ці поля є захищеними, що означає, що вони доступні лише за умови успішної перевірки статусу та ролі користувача.

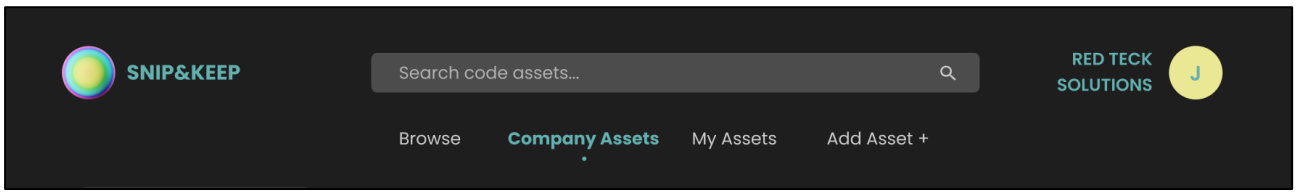

Рисунок 6 – Розширене поле навігації (рисунок виконаний самостійно)

Створення фрагментів коду є однією з ключових можливостей для авторизованого користувача. Дизайн інтерфейсу цієї функції реалізовано через інтуїтивно зрозумілий набір заповнюваних полей, які дозволяють ввести назву

фрагмента, додати опис, вибрати відповідний мовний синтаксис та встановити теги для подальшої категоризації (рис. 7).

Особливу увагу візуальний дизайн приділяє файловій структурі фрагмента та полю для введення коду – воно оформлене у вигляді спеціалізованого редактора з підсвічуванням синтаксису, що сприяє зручності у написанні, редагуванні та читанні коду (рис. 8).

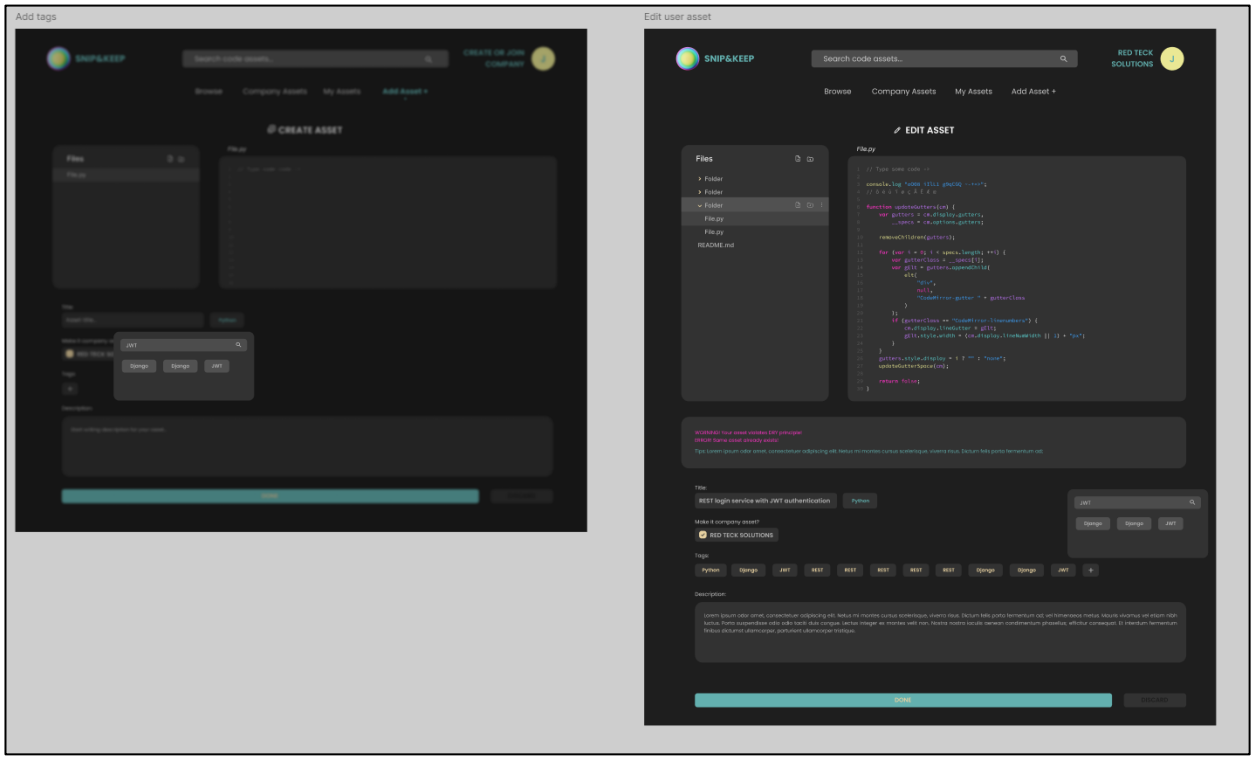

Рисунок 7 – Дизайн додавання та редагування фрагменту коду (рисунок виконаний самостійно)

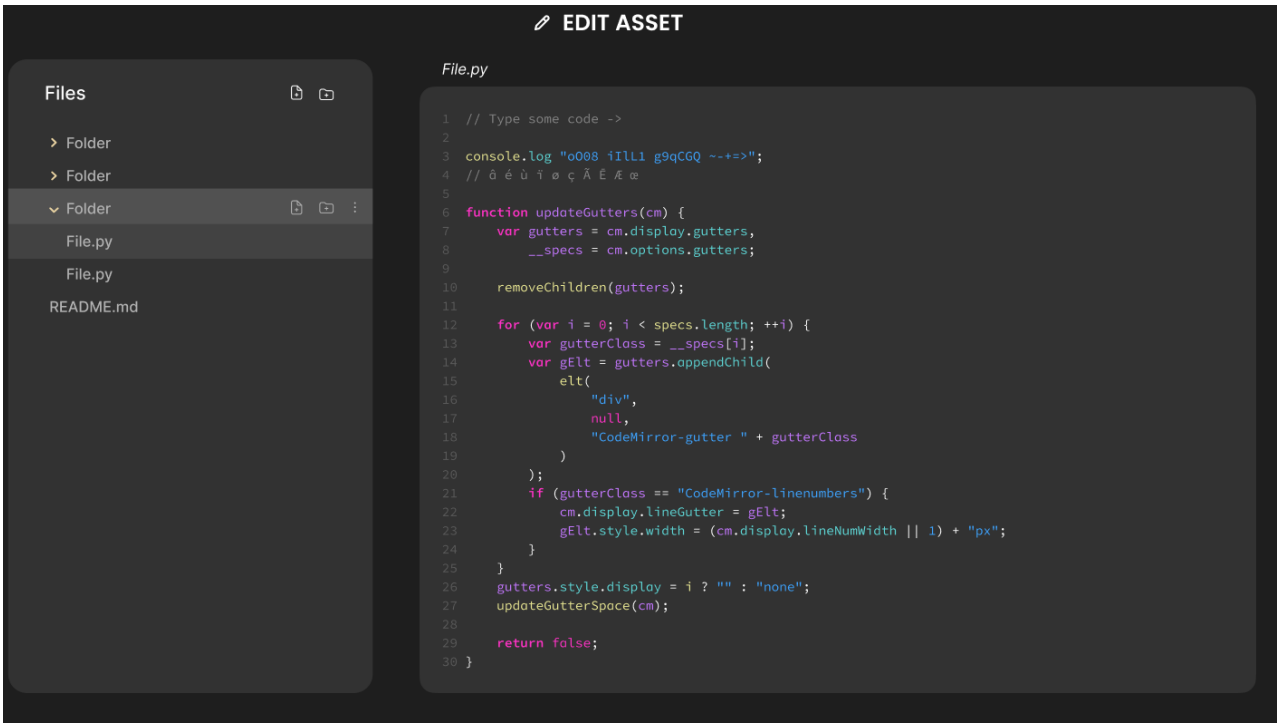

Рисунок 8 – Дизайн структури файлів та поля введення коду (рисунок виконаний самостійно)

3.3.4 Інтерфейс корпоративного користувача

Інтерфейс корпоративного користувача є найрозширенішим рівнем взаємодії з системою. Його основне призначення – надати представникам компаній доступ до спільного корпоративного простору, в якому зберігаються та організовуються фрагменти коду, що належать до певної організації. У межах інтерфейсу корпоративного користувача реалізовано можливість перегляду колекції корпоративних фрагментів, яка відображає усі доступні для компанії кодові фрагменти з відповідними назвами, описами, тегами та мовами програмування. Ці фрагменти можуть бути переглянуті, відфільтровані за тегами або знайдені за допомогою пошуку (рис. 9).

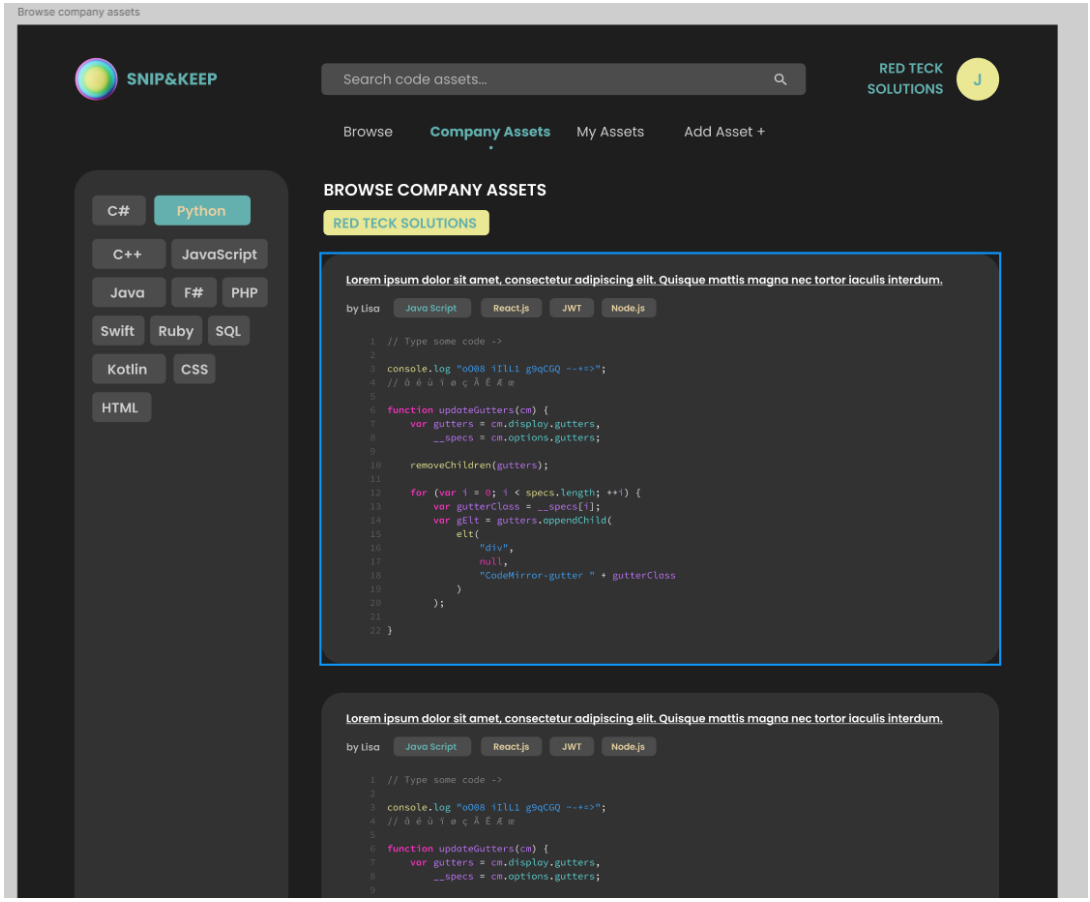

Рисунок 9 – Дизайн інтерфейсу перегляду фрагментів компанії (рисунок виконаний самостійно)

Окрему важливу частину цього інтерфейсу становить функціонал створення нового проєкту на основі наявних фрагментів компанії. Цей процес передбачає інтеграцію зі штучним інтелектом, до якого користувач подає запит у вигляді технічного завдання [10] або текстового опису бажаного функціоналу. На основі цього запиту система комбінує релевантні фрагменти коду та генерує прототип проєкту (перший фрагмент, зображений на рисунку 10). Результатом є цілісна кодова структура, яку користувач може переглянути, зкомпілювати та протестувати безпосередньо в інтерфейсі. Для забезпечення якості згенерованого проєкту спроєктовано дизайн автоматичного аналізу коду на відповідність принципам DRY (Don't Repeat Yourself) та SOLID. Користувач також має змогу завантажити фінальний результат у вигляді архіву (ZIP), що містить повну

структуру проєкту для подальшого використання. Розроблений дизайн інтерфейсу наведено на другому фрагменті рисунку 10.

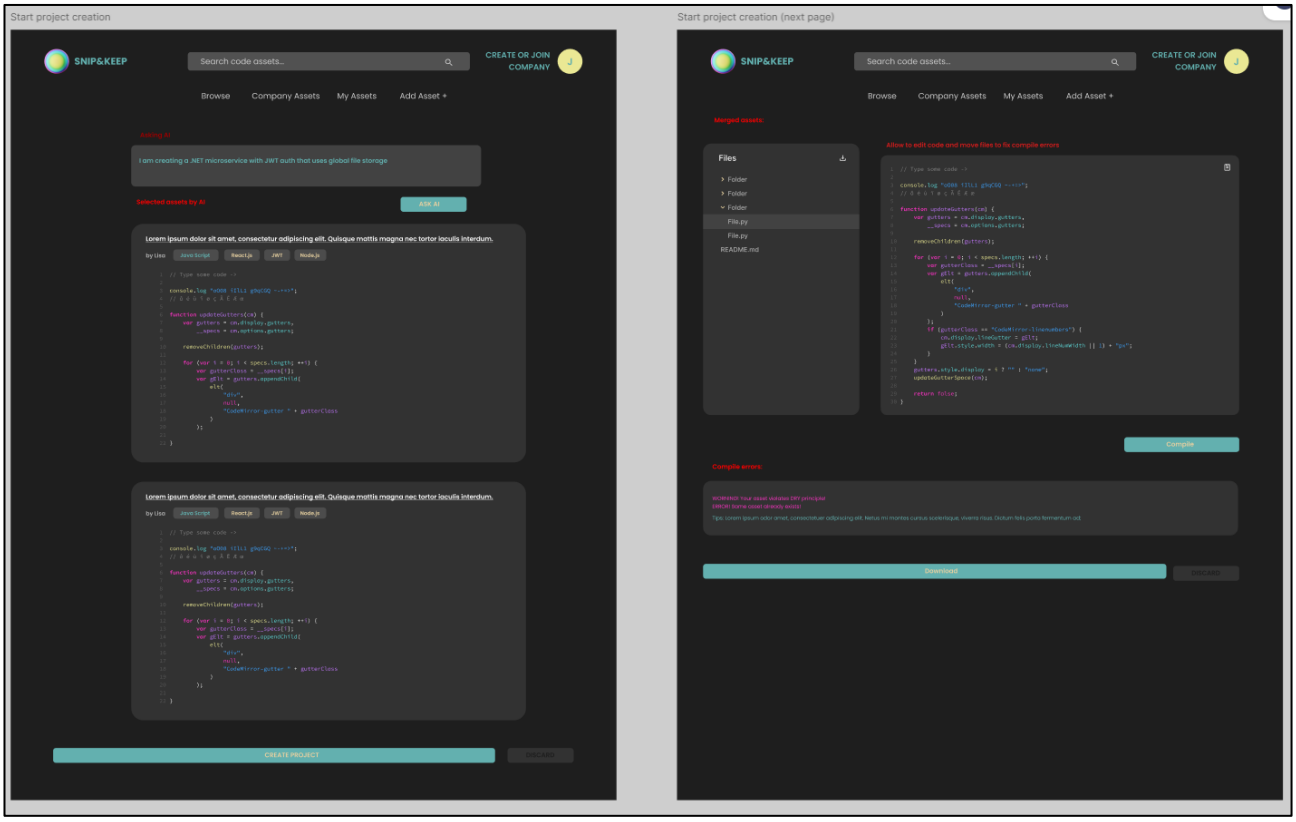

Рисунок 10 – Дизайн інтерфейсу створення проєкту (рисунок виконаний самостійно)

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Використані технології

Під час розробки клієнтської частини системи було прийнято рішення використовувати сучасний набір технологій, який поєднує в собі гнучкість, масштабованість, високу швидкодію та підтримку індустріальних стандартів розробки. До переліку використаних технологій входять наступні інструменти та бібліотеки:

- React.js – JavaScript-бібліотека, що була обрана як основа клієнтського застосунку завдяки своїй компонентній архітектурі та широкій екосистемі. Вона дозволила ефективно розділяти інтерфейс на логічно ізольовані елементи, які можуть бути повторно використані. Реактивна модель оновлення стану посприяла високій продуктивності та покращеній взаємодії з користувачем;
- React Query – бібліотека для управління серверним станом. З її допомогою реалізовано зручну взаємодію з API: автоматичне кешування, повторні запити, оновлення даних у фоновому режимі, обробка помилок та спрощене керування статусами завантаження. Цей інструмент дозволив зменшити обсяг імперативного коду та підвищити надійність взаємодії з сервером;
- React Hook Form – легка та ефективна бібліотека для створення форм із валідацією. Забезпечила мінімальну кількість перевантажень, простоту валідації та високу продуктивність. ЇЇ було активно використано в реалізації форм авторизації та реєстрації;
- React Hot Toast – інструмент для реалізації сповіщень у застосунку. Завдяки цьому рішенню користувачі миттєво отримують зворотний зв’язок про результати своїх дій, що суттєво покращує UX і підвищує прозорість взаємодії з інтерфейсом;
- Monaco Editor for React – редактор коду, який є офіційним web-аналогом редактора Visual Studio Code. Він інтегрований до інтерфейсу системи для

роботи з фрагментами коду. Забезпечує підсвічування синтаксису, перевірку помилок та інші інструменти, необхідні для комфортної роботи з кодом безпосередньо у браузері;

– React Responsive – бібліотека, що надає інструменти для адаптації інтерфейсу під різні розміри екранів. Використовується для створення мобільної версії застосунку та забезпечення належного відображення на планшетах і смартфонах;
– Radix UI – набір низькорівневих, доступних та адаптивних компонентів інтерфейсу. Бібліотека забезпечує відповідність стандартам доступності (accessibility) та дозволяє створювати функціональні, проте гнучкі з точки зору дизайну, компоненти;
– JWT Decode – утиліта для декодування JSON Web Token. Вона використовується для зчитування даних про користувача (ID (Identification), роль тощо) після аутентифікації без необхідності додаткових запитів до сервера. Це пришвидшує обробку даних і покращує UX.
– Styled Components – бібліотека для CSS-in-JS, що дозволяє описувати стилі безпосередньо в компоненті. Вона забезпечує ізоляцію стилів, підтримку тем, динамічне оновлення стилів залежно від стану компонента, а також покращує читабельність та підтримуваність коду;
– Testing Library, Jest-DOM та Vitest – набір інструментів для написання модульних тестів. Testing Library орієнтована на тестування з точки зору користувача (user-centric approach), Vitest – як сучасна, сумісна з Vite тестова утиліта, забезпечує швидке виконання тестів. Їх застосування дозволяє перевіряти стабільність і коректність логіки застосунку;
– ESLint – інструмент статичного аналізу коду. Він забезпечує дотримання єдиного стилю написання, виявлення потенційних помилок на ранніх етапах розробки та сприяє підтримуваності проєкту.

Усі ці технології були ретельно обрані з урахуванням специфіки проєкту, його масштабів, цільової аудиторії та технічних вимог. Їхнє поєднання дозволило створити гнучкий, адаптивний та продуктивний користувацький інтерфейс, який може масштабуватися відповідно до майбутніх потреб системи.

4.2 Реалізація авторизації та реєстрації користувача

Система підтримує повноцінну авторизацію та реєстрацію користувачів із використанням JWT (JSON Web Token) (див. дод. А), що дозволяє реалізувати безпечне та масштабоване управління доступом. Даний функціонал реалізовано з урахуванням сучасних практик клієнтської розробки, зокрема, з використанням бібліотек React Query, jwt-decode, React Hook Form та стандартних механізмів браузера.

Після реєстрації або авторизації користувач отримує пару токенів:

– Access Token – короткотривалий токен (зазвичай з часом дії 15 хвилин), який використовується для авторизації запитів;
– Refresh Token – довготривалий токен, за допомогою якого можливо отримати новий Access Token без повторного входу користувача.

У випадку, коли час дії accessToken минув, викликається функція refreshAccessToken(). Вона надсилає запит до сервера, передаючи обидва токени:

```
const response = await fetch(`${BASE_URL}/tokens/refresh`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ accessToken, refreshToken }),
});
```

Якщо відповідь від сервера успішна, оновлені токени знову зберігаються в localStorage, і користувач залишається в системі без необхідності входити заново. Функція має захист від одночасного запуску кількох запитів через змінну refreshPromise, що гарантує, що один користувачський запит не призведе до декількох одночасних запитів на оновлення. Було впроваджено допоміжну функцію для визначення, чи є автентифікований користувач. Вона декодує токен,

перевіряє його дійсність і за потреби оновлює accessToken. Для роботи з JWT використовується бібліотека jwt-decode, яка дозволяє зчитувати зашифровану у токені інформацію про користувача без відправки запитів до сервера. У реалізованій функції decodeToken відбувається зчитування типових для .NET API claims.

```
      const decodedData = jwtDecode(accessToken);
      const cleanedData = {
        name:
decodedData["http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"],
        id:
decodedData["http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameiden
tifier"],
        email:
decodedData["http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailadd
ress"],
        role:
decodedData["http://schemas.microsoft.com/ws/2008/06/identity/claims/role"]
,
        exp: decodedData.exp,
      };
```

Ці дані використовуються для побудови інтерфейсу (наприклад, відображення імені або ролі користувача) та обмеження доступу до певного функціоналу.

Розроблений користувацький інтерфейс реєстрації та авторизації наведено на рисунку 11.

4.3 Побудова головної сторінки та навігації

Головна сторінка розробленого вебінтерфейсу виконує функцію централізованого перегляду фрагментів програмного коду (рис. 12). Головна панель навігації містить можливості для переходу між основними розділами: «Browse», «My Assets», «Add Asset», «Company Assets» і «Start Project». Також доступна панель пошуку, що дозволяє знаходити фрагменти за ключовими словами. Бокове меню з категоріями реалізує фільтрацію фрагментів за тематикою, такою як Networking, Machine Learning, Database, Encryption тощо.

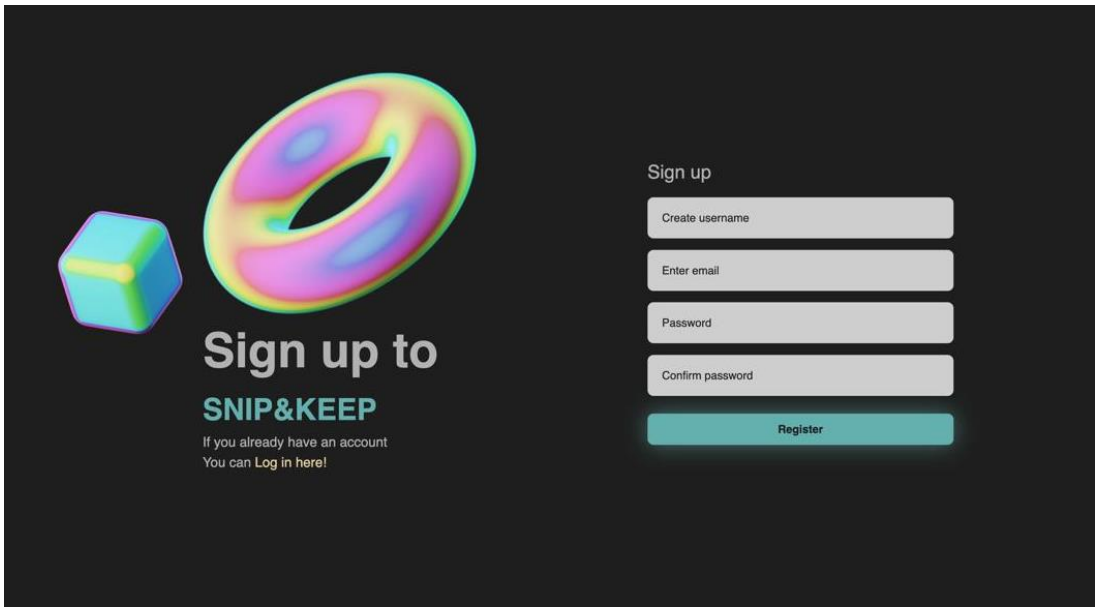

Рисунок 11 – Розроблений користувацький інтерфейс реєстрації (рисунок виконаний самостійно)

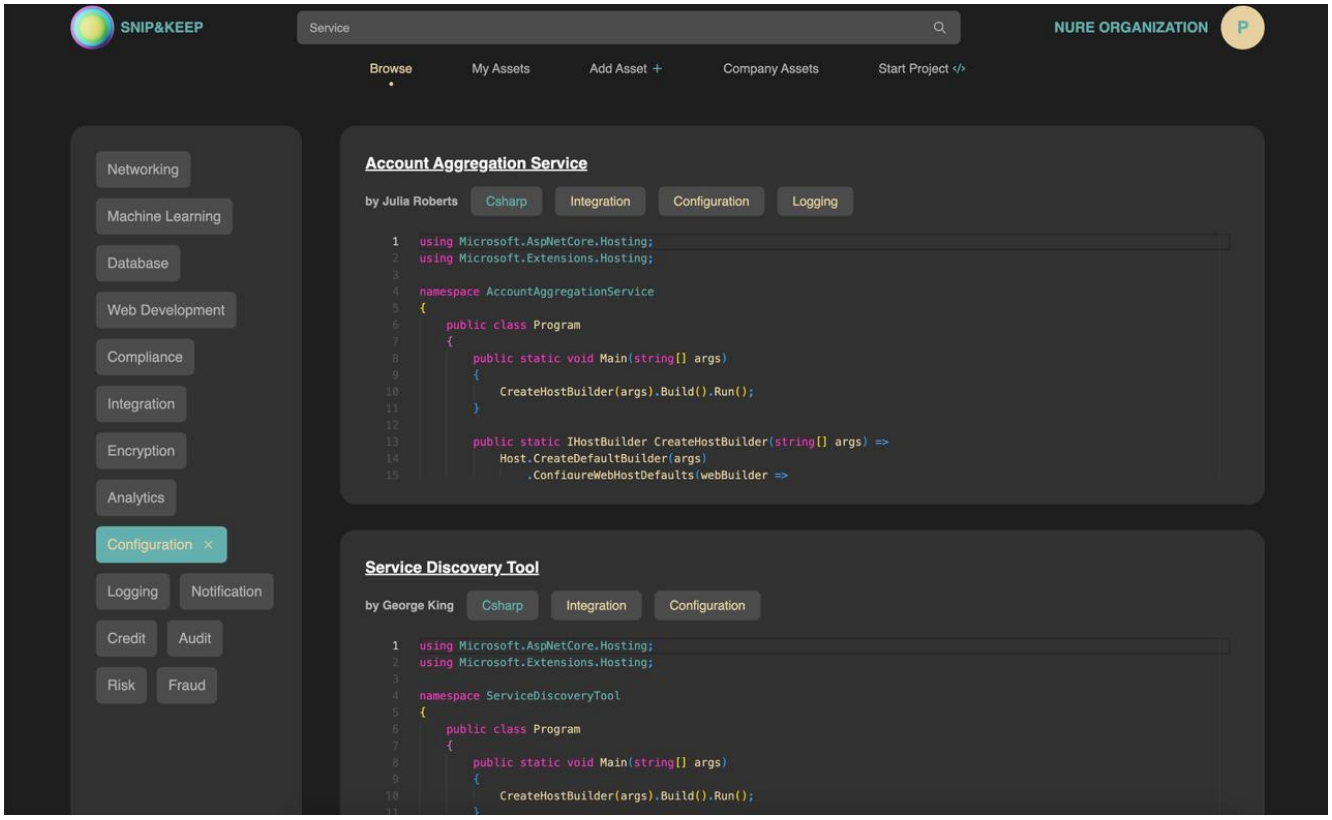

Рисунок 12 – Розроблений користувацький інтерфейс перегляду вільно доступних фрагментів (рисунок виконаний самостійно)

Кожна категорія є інтерактивною кнопкою для швидкого доступу до відповідного типу кодових фрагментів. Центральна частина сторінки містить

перелік кодових фрагментів з назвою, автором, мовою програмування (наприклад, C#), мітками (теги) та підсвіченим синтаксисом коду.Це забезпечує зручний візуальний перегляд та швидке ознайомлення зі змістом. Користувацький інтерфейс для перегляду особистих фрагментів та фрагментів компанії має такий саме виглял, але є приватним та доступний лише користувачам з відповідними ролями.

4.4 Робота з фрагментами коду

У вебзастосунку реалізовано можливість керування фрагментами коду, при цьому користувач має змогу взаємодіяти лише з тими фрагментами, які були створені ним особисто. Після створення фрагмента користувач може вносити до нього зміни, зокрема редагувати його назву (рис. 13). Це дозволяє надати кожному фрагменту більш осмислену і впізнавану назву, що значно полегшує подальший пошук і організацію матеріалів.

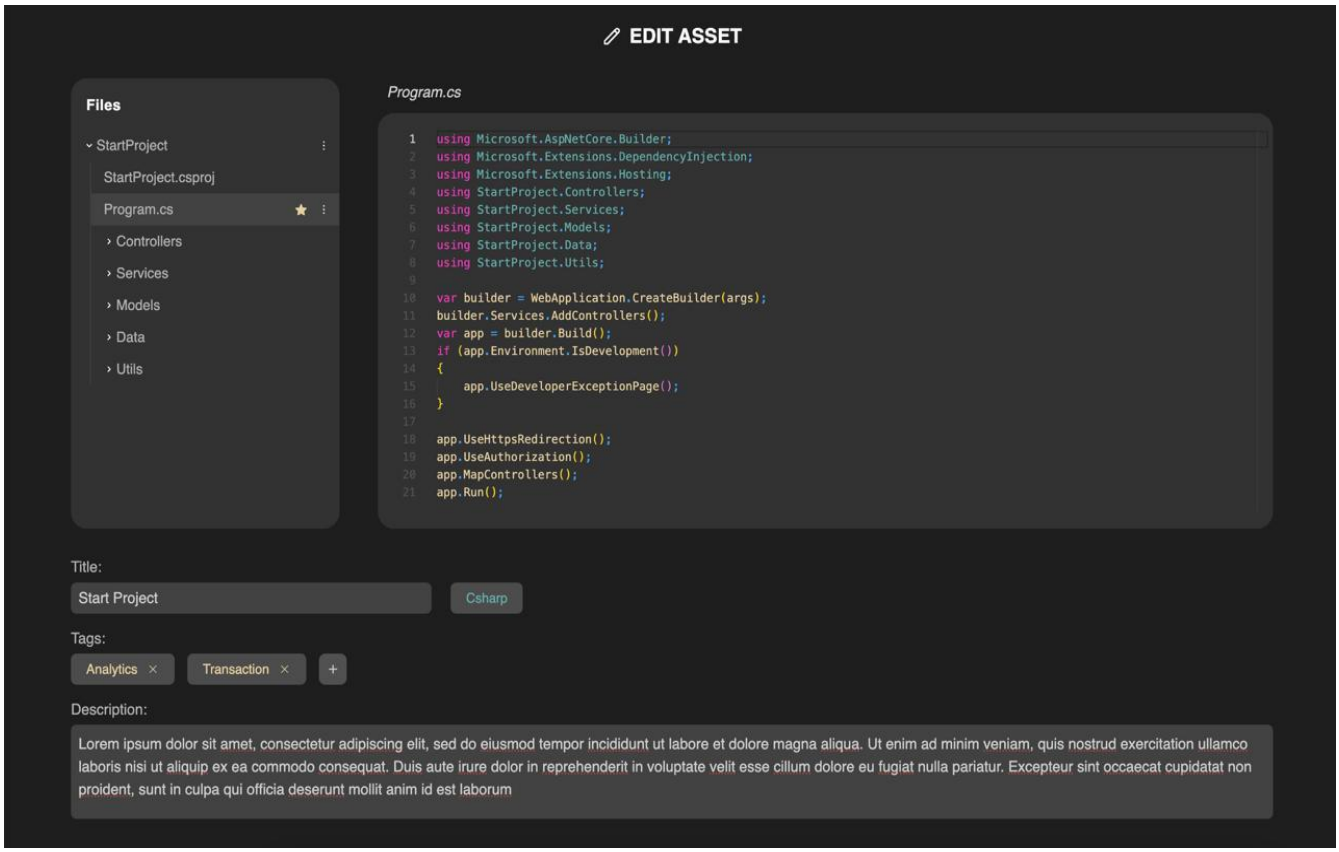

Рисунок 13 – Розроблений користувацький інтерфейс редагування фрагменту (рисунок виконаний самостійно)

Також у фрагмент можна додати короткий опис, який роз'яснює його призначення або особливості реалізації. Опис виконує роль додаткового орієнтира і дозволяє швидко зрозуміти, для чого призначено конкретний блок коду, не відкриваючи його повністю.

Мова програмування, якою написано код у фрагменті, визначається автоматично. Це відбувається на основі аналізу основного файлу в межах фрагмента. Завдяки цьому користувачу не потрібно обирати мову вручну, що робить процес додавання нових фрагментів швидшим і зручнішим.

Окрім цього, користувач має можливість додавати до фрагментів тематичні теги, які допомагають структурувати код за категоріями (рис. 14). Наприклад, можна позначити фрагмент тегом "Web Development", якщо він стосується веброзробки, або "Notification", якщо в ньому реалізовано логіку сповіщень. Це спрощує процес навігації в особистій бібліотеці фрагментів, особливо коли їх кількість зростає.

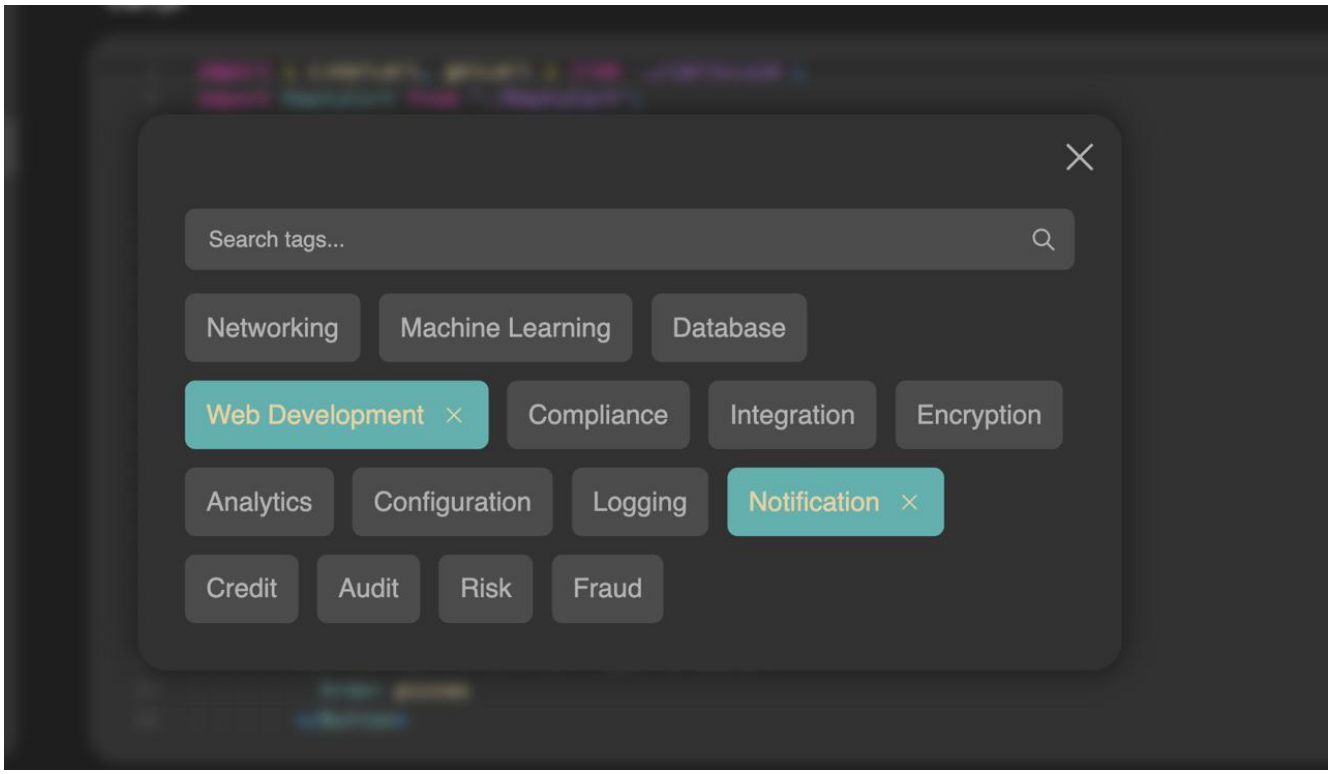

Рисунок 14 – Розроблений користувацький інтерфейс додавання тегів (рисунок виконаний самостійно)

У межах одного фрагмента може бути декілька файлів, але лише один з них вважається основним. Користувач має змогу позначити цей файл спеціальним символом – зірочкою. Саме вміст основного файлу відображається в загальному списку фрагментів (як було зазначено на рисунку12) як представницький, тобто саме його бачать користувачі в особистій бібліотеці або на головній сторінці застосунку.

4.5 Побудова файлового дерева

У процесі додавання нових файлів і папок в інтерфейсі застосунку використовується рекурсивна логіка, яка забезпечує динамічне побудування структури каталогу у вигляді дерева. Користувач може взаємодіяти з кожною папкою окремо, відкриваючи її для перегляду вмісту або додаючи до неї нові елементи (рис. 15).

Цей процес починається з кореневої папки (rootFolder), яка передається компоненту FolderTree (див. дод. Б). Усередині цього компоненту перевіряється, чи є дана папка відкритою, використовуючи глобальний стан з контексту. Якщо папка відкрита, її вміст відображається у вигляді списку вкладених елементів – файлів та інших папок.

```
function Folders({ rootFolder, readOnly = true }) {
  const isMobile = useMediaQuery({ query: "(max-width: 768px)" });
  if (isMobile) {
    return (
      <EditedFolderProvider>
        <EditedFileProvider>
          <Modal>
            <Modal.Open opens="folders">
              <FilterButton variation="icon">
                <HiOutlineFolderOpen /> Files
              </FilterButton>
            </Modal.Open>
            <Modal.Window name="folders">
              <OpenFoldersProvider>
                <FoldersContainer>
                  <Title>Files</Title>
                  <FolderTree folder={rootFolder} readOnly={readOnly}
/>
                </FoldersContainer>
              </OpenFoldersProvider>
```

```
            </Modal.Window>
          </Modal>
        </EditedFileProvider>
      </EditedFolderProvider>
    );
  }
  return (
    <EditedFolderProvider>
      <EditedFileProvider>
        <OpenFoldersProvider>
          <FoldersContainer>
            <Title>Files</Title>
            <FolderTree folder={rootFolder} readOnly={readOnly} />
          </FoldersContainer>
        </OpenFoldersProvider>
      </EditedFileProvider>
    </EditedFolderProvider>
  );
}
```

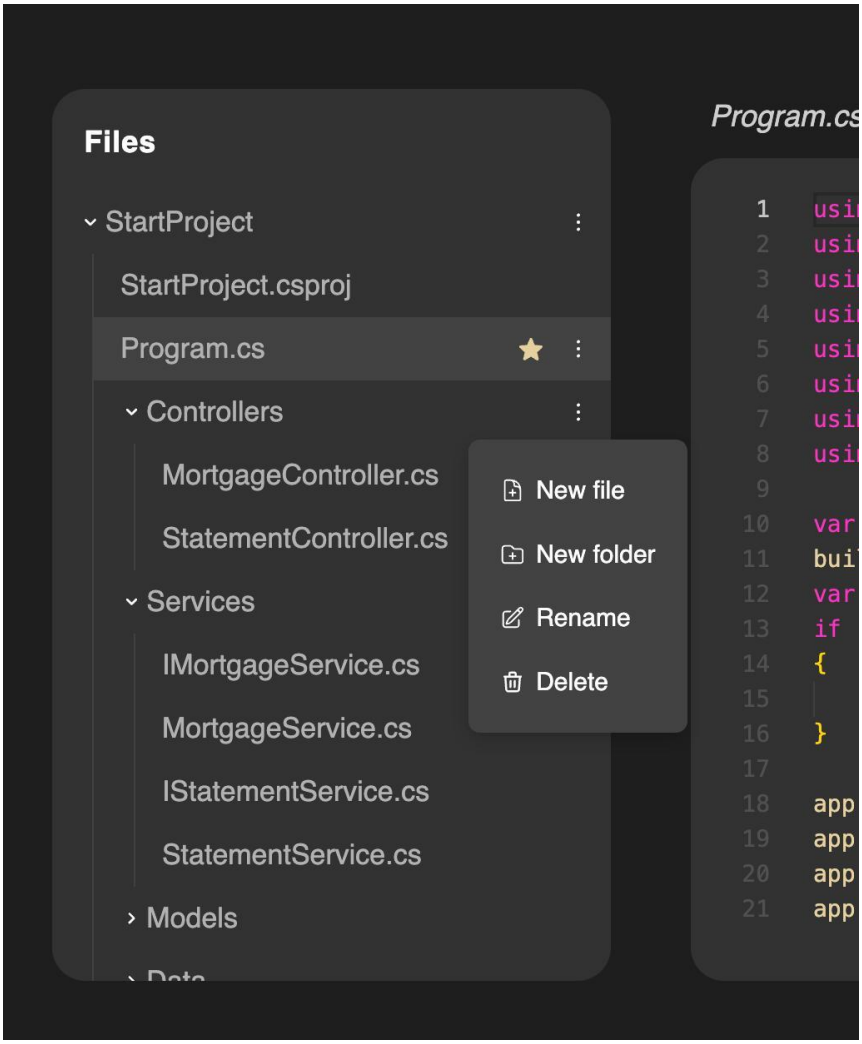

Рисунок 15 – Розроблений користувацький інтерфейс організації файлів (рисунок виконаний самостійно)

Кожна вкладена папка, в свою чергу, також є об’єктом типу folder, і тому для її відображення рекурсивно знову викликається компонент FolderTree. Завдяки такому підходу дерево файлів може мати будь-яку глибину, а процес візуалізації завжди буде правильним незалежно від рівня вкладеності. Якщо користувач створює нову папку або файл, відповідна функція (handleCreateFolder чи handleCreateFile) передає parentId, що відповідає ID тієї папки, всередині якої виконується створення. Таким чином, новий елемент автоматично додається у відповідне місце в дереві, і при успішному створенні його ID тимчасово зберігається в локальному стані, щоб ініціювати редагування назви щойно створеного елементу.

Варто підкреслити, що створення та відображення нових елементів повністю інтегроване у рекурсивну логіку дерева. Компонент FolderTree не лише будує вміст, а й обробляє додавання нових елементів, делегуючи функції зворотного виклику (onFolderCreated та onFileCreated) дочірнім компонентам. Це дозволяє кожному рівню вкладеності працювати автономно, зберігаючи єдину логіку взаємодії. Завдяки рекурсивній побудові кожен новий елемент одразу інтегрується у відповідну гілку дерева, а зміни в структурі відображаються у реальному часі без необхідності перезавантаження всього дерева з нуля. Такий підхід забезпечує ефективність, масштабованість та зручність у роботі з великою кількістю файлів і папок, а також дозволяє користувачеві легко створювати складні структури проєктів.

4.6 Створення та перегляд проєктів корпоративного користувача

Функціонал створення проєктів для корпоративного користувача реалізовано з урахуванням потреб повторного використання раніше створених рішень та автоматизації процесу на основі штучного інтелекту.

Процес створення нового проєкту починається з введення текстового запиту, в якому користувач описує мету або вимоги до майбутнього проєкту (рис.16). На

основі цього опису система аналізує запит, використовуючи механізми штучного інтелекту, та здійснює пошук відповідних реалізованих фрагментів з внутрішньої бази даних компанії.

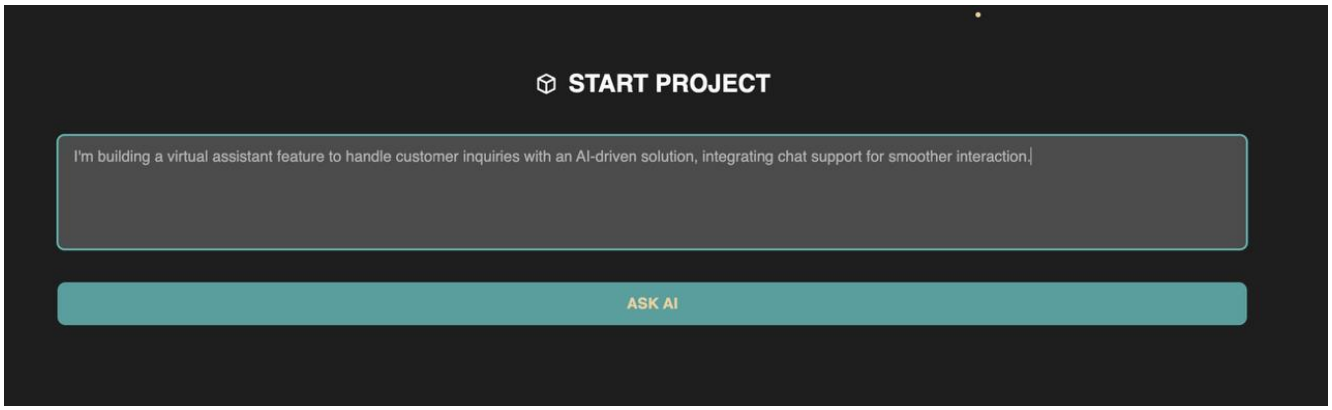

Рисунок 16 – Розроблений користувацький інтерфейс запиту (рисунок виконаний самостійно)

Зі знайдених фрагментів система формує пропозицію комбінації, яка найкраще відповідає заданому запиту (рис. 17).

Згенерований проєкт містить кілька обраних фрагментів коду, що об’єднані в єдину структуру. Кожен з фрагментів зберігає можливість взаємодії, аналогічно до індивідуального перегляду, зокрема: перегляд, редагування, переміщення між директоріями та видалення.

Окрім цього, в межах проєкту реалізовано можливість компіляції коду (рис.18). Після запуску компіляції користувач отримує результат у вигляді успішності виконання або помилок, а також додаткові рекомендації щодо покращення якості коду. Такі рекомендації ґрунтуються на принципах чистого коду, включаючи DRY та SOLID. Це дозволяє не лише оперативно виявляти технічні помилки, але й підтримувати високий рівень підтримуваності та масштабованості проєктів.

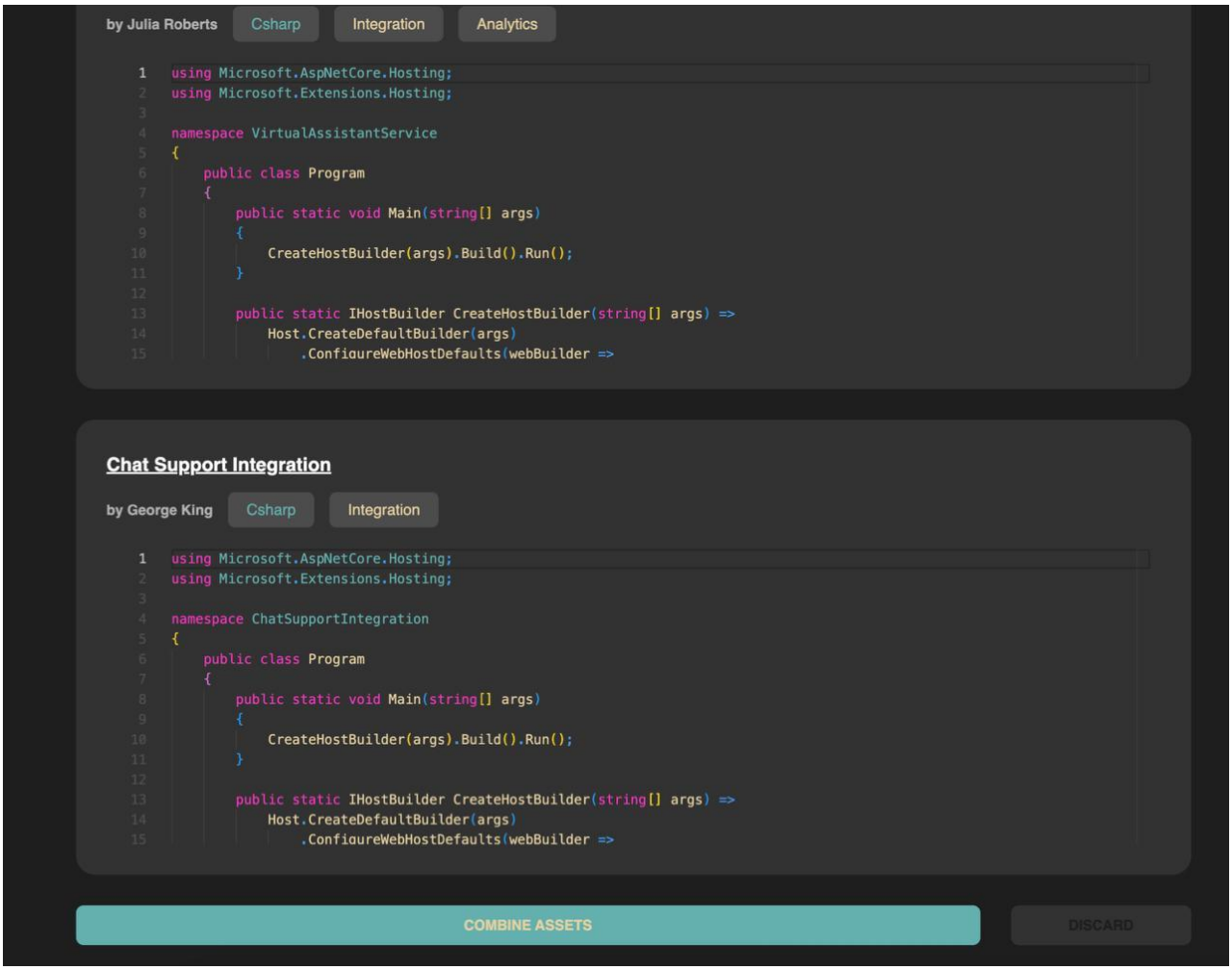

Рисунок 17 – Розроблений користувацький інтерфейс комбінації фрагментів (рисунок виконаний самостійно)

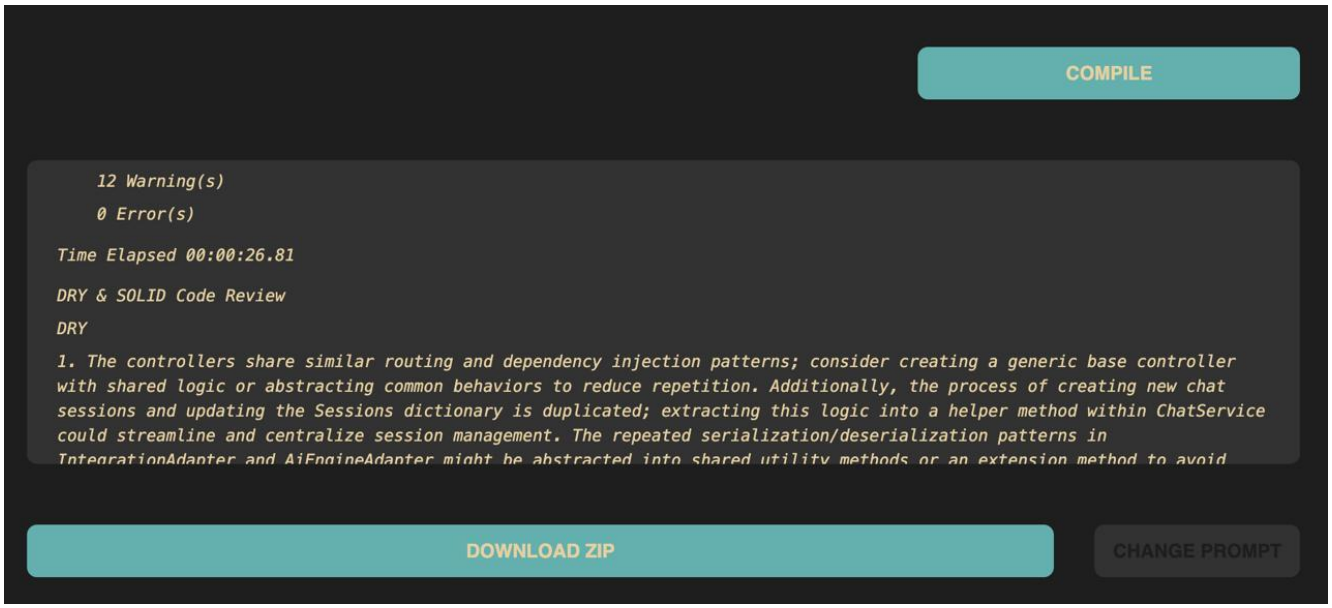

Рисунок 18 – Розроблений користувацький інтерфейс компіляції проєкту (рисунок виконаний самостійно)

Окрім цього, в межах проєкту реалізовано можливість компіляції коду (рис.18). Після запуску компіляції користувач отримує результат у вигляді успішності виконання або помилок, а також додаткові рекомендації щодо покращення якості коду.

4.7 Керування інформацією про компанію та її учасників

Однією з важливих функцій системи є сторінка компанії, яка містить базову інформацію про організацію, а також дозволяє керувати учасниками, що входять до її складу (рис. 19). На цій сторінці відображається назва компанії, супровідна інформація, а також перелік користувачів, які приєднані до неї. Для кожного учасника вказується його ім’я, електронна пошта та поточна роль у межах компанії. Перегляд цієї інформації доступний усім користувачам, які мають доступ до компанії, проте додаткові можливості відкриваються лише для користувачів із роллю адміністратора.

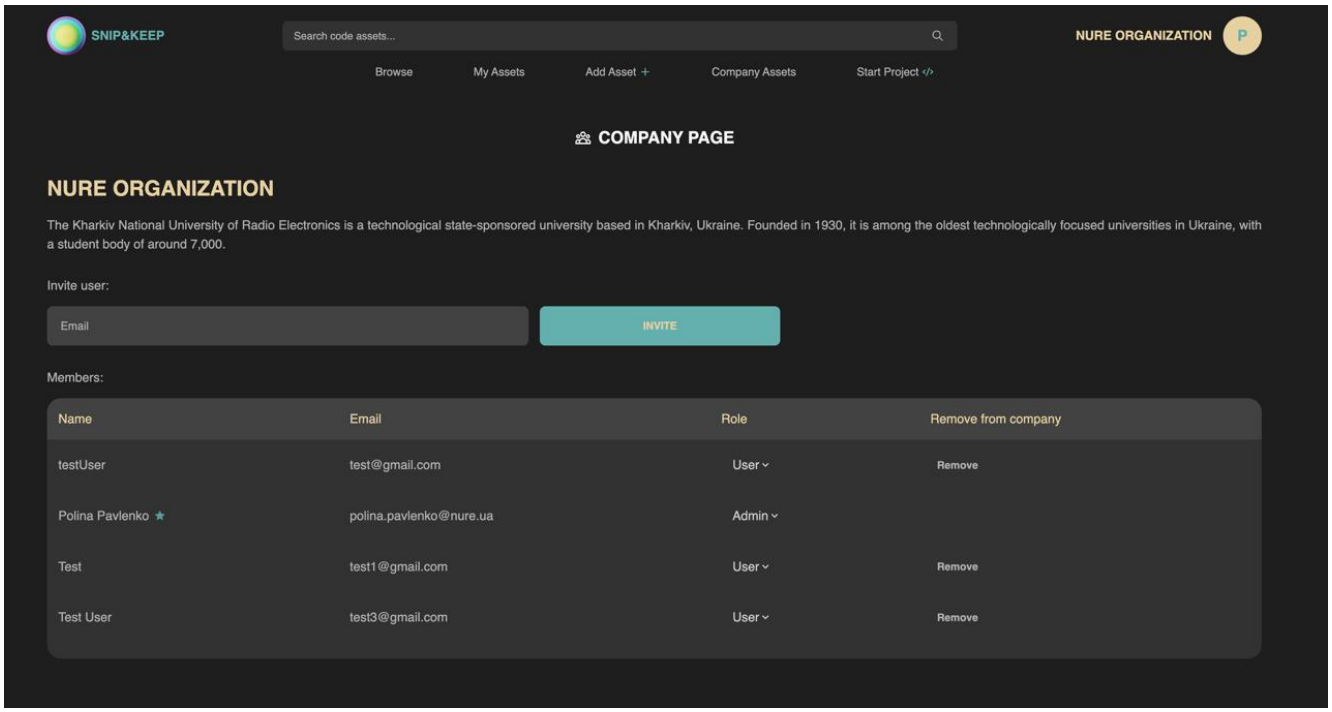

Рисунок 19 – Розроблений користувацький інтерфейс компанії (рисунок виконаний самостійно)

Адміністратор компанії може додавати нових користувачів на основі їх електронної пошти. У разі, якщо користувач уже зареєстрований у системі, він приєднується до компанії. Якщо ж обліковий запис відсутній, система попереджає про неможливість додавання. Крім того, адміністратор має можливість змінювати ролі вже наявних учасників, що дозволяє гнучко керувати повноваженнями в команді. У межах цієї ж сторінки передбачено функцію видалення користувачів з компанії.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Модульне тестування

Модульне тестування є одним із базових підходів до перевірки коректності роботи програмного забезпечення. Воно передбачає тестування окремих одиниць логіки програми – функцій, компонентів або модулів – ізольовано від інших частин системи. Такий підхід дозволяє виявити помилки на ранньому етапі розробки, забезпечити стабільність коду під час рефакторингу та підвищити впевненість у правильності реалізованої логіки.

У процесі розробки вебзастосунку було реалізовано модульні тести для основних функціональних компонентів інтерфейсу, а також для допоміжних функцій, таких як валідація, обробка введення користувача, логіка створення проєктів тощо. Для тестування використовувалась бібліотека Vitest, яка є сучасним інструментом для модульного тестування в середовищі Vite. Вона забезпечує високу швидкість виконання тестів, має зручний API та добре інтегрується з бібліотекою Testing Library, що використовується для тестування компонентів інтерфейсу користувача.

Було створено модульні тести для форми додавання нового фрагменту коду, що перевіряли правильність валідації полів, коректність виклику функції надсилання даних, а також наявність повідомлень про помилки у випадку, якщо користувач залишив обов’язкові поля порожніми. Один із тестів перевіряв, що після заповнення форми та натискання кнопки «Створити» надсилається запит до API та форма очищується. Інший тест імітував процес завантаження та перевіряв, чи блокується можливість введення нових даних під час процесу обробки запиту сервером.

```
describe("AddAssetForm Loading State", () => {
  it("disables form elements when isPending is true", () => {
    render(<AddAssetForm />);

    // Check if inputs are disabled
    expect(screen.getByPlaceholderText(/provide a
name/i)).toBeDisabled();
```

```
      // Check if button is disabled
      const button = screen.getByRole("button");
      expect(button).toBeDisabled();
    });
});
```

Тестування компонентів також здійснювалося ізольовано, з використанням фіктивних функцій для зовнішніх запитів, щоб уникнути залежності від сервера. Завдяки цьому вдалось досягти стабільного середовища тестування, що дозволяє зосередитись саме на поведінці компонента в різних умовах. Результат проведення модульних тестів над компонентом програми, що відповідає за створення нового фрагмента коду наведено на рисунку 20, повний код наведено у додатку В.

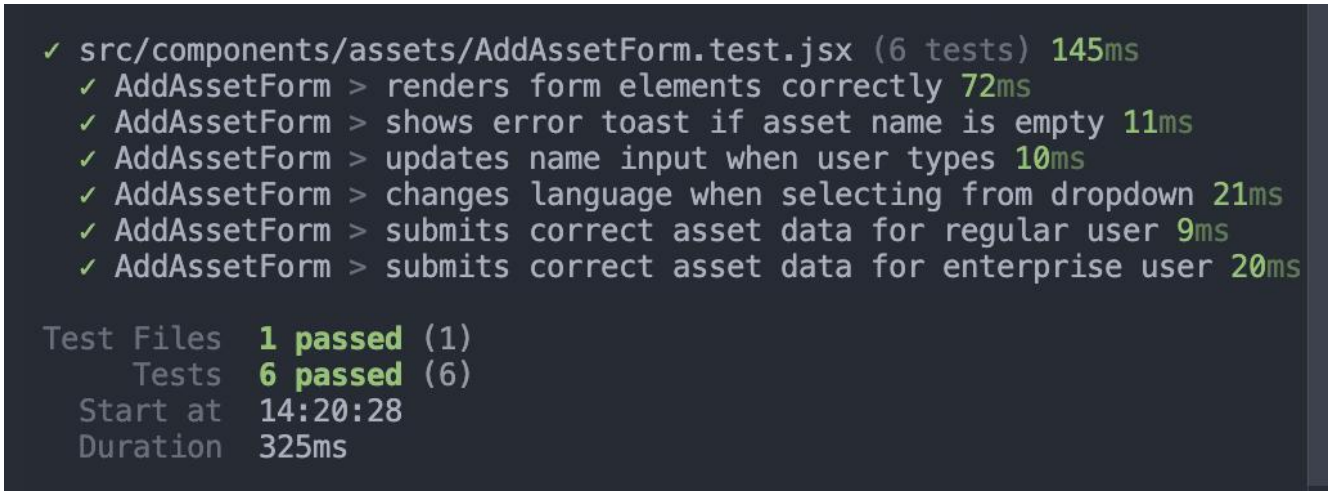

Рисунок 20 – Результат виконання модульних тестів (рисунок виконаний самостійно)

5.2 Перевірка негативних сценаріїв

Особливу увагу було приділено негативним тест кейсам. Вони моделюють ситуації, коли користувачі вводять некоректні або неповні дані, що дозволяє виявити, як система поводиться у разі помилки (див. рис. 21). Було протестовано різні ситуації: введення некоректної електронної пошти, спроба збереження без заповнених полів, некоректне звернення до API тощо.

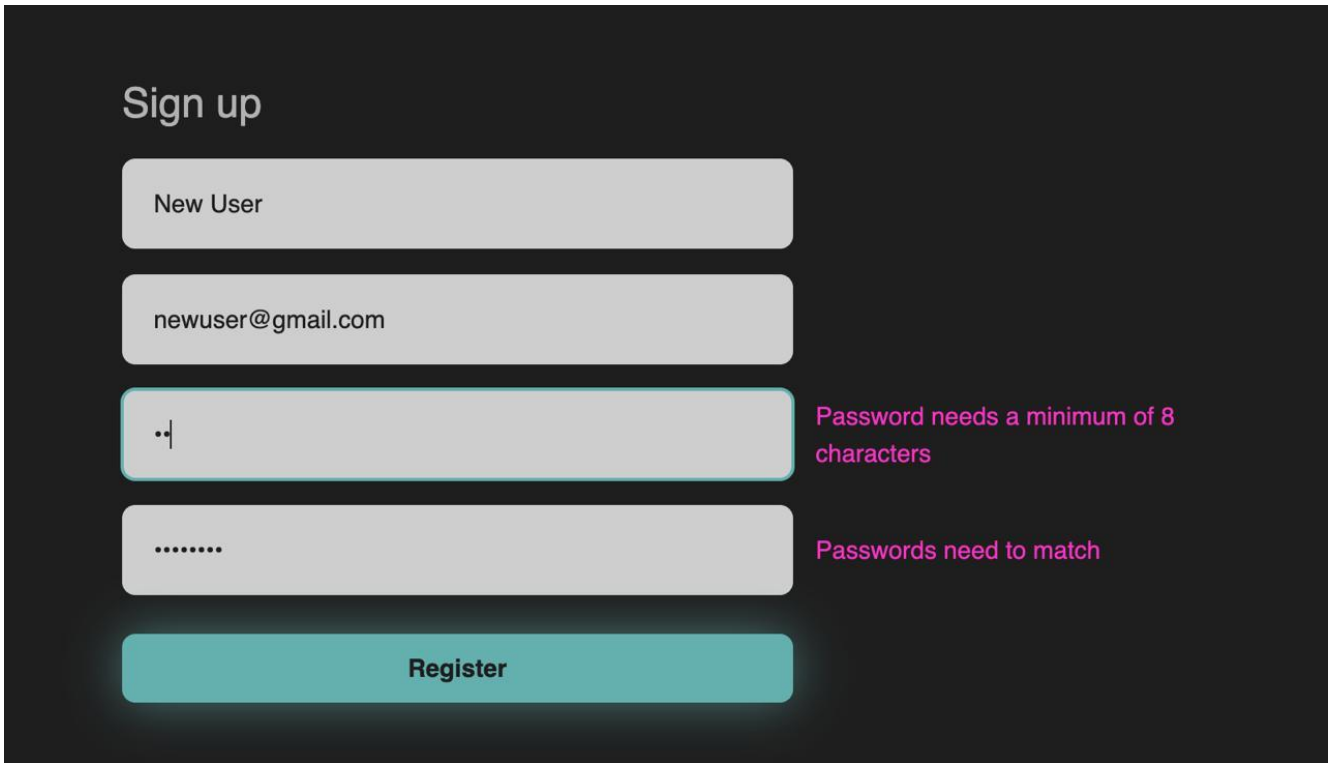

Рисунок 21 – Результат відповіді системи на некоректні дані (рисунок виконаний самостійно)

Наприклад при спробі користувача створити новий файл із розширення, що не підтримується у додатку користувачеві повертається відповідне попереджувальне повідомлення (рис. 22).

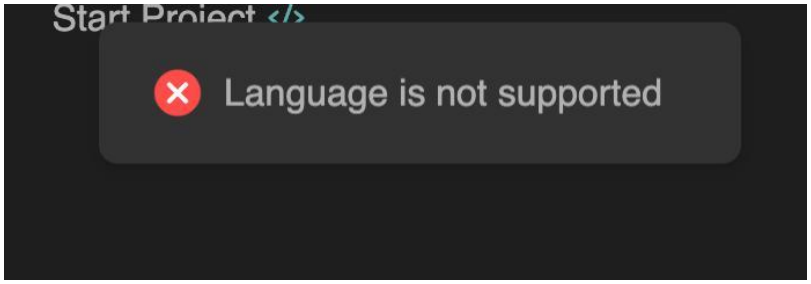

Рисунок 22 – Попереджувальне повідомлення (рисунок виконаний самостійно)

Також було враховано сценарії, коли шукану користувачем інформацію не знайдено, у такому випадку відображається відповідний результат проведення операції пошуку (див. рис. 23).

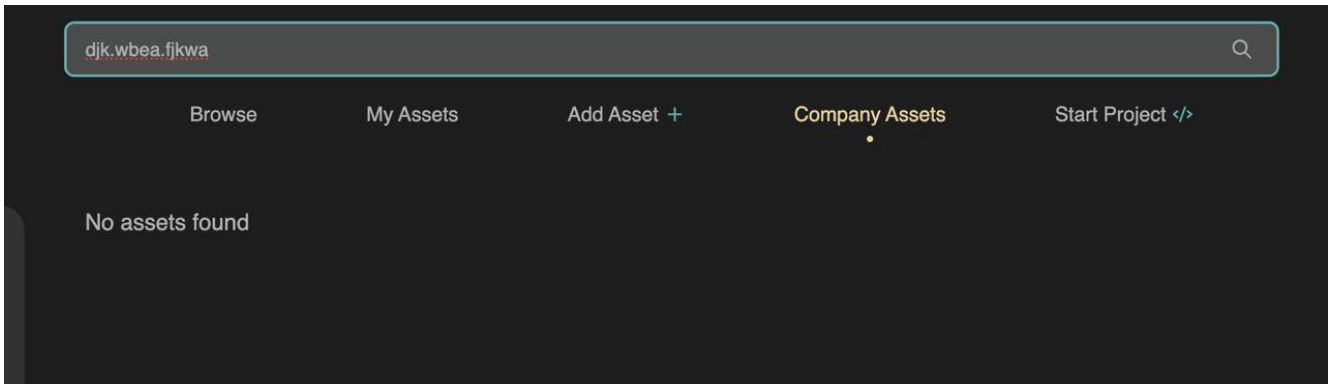

Рисунок 23 – Негативний результат пошуку (рисунок виконаний самостійно)

У всіх випадках система обробляє помилки без падіння, відображаючи відповідні повідомлення. Наприклад, у випадку помилки на сервері при створенні проєкту користувач бачить повідомлення: "Не вдалося створити проєкт. Спробуйте пізніше".

5.3 Тестування адаптивності інтерфейсу

Тестування адаптивності виконувалося за допомогою вбудованих інструментів розробника у браузері Google Chrome та BrowserStack – хмарної платформи для веб та мобільного тестування застосунків у різних браузерах, операційних системах та пристроях. Зокрема, перевірялася коректність відображення основних сторінок застосунку на популярних типах екранів: мобільних пристроях, планшетах та десктопах.

Було протестовано поведінку елементів навігації, форм, таблиць і модальних вікон. Наприклад, при зменшенні ширини вікна застосунок коректно змінює структуру відображення, ховаючи або адаптуючи бічне меню, змінюючи розмір шрифтів та кнопок (рис. 24). Бічні та навігаційні меню на планшетах та телефонах приховуються та працюють як модальні вікна (рис. 24).

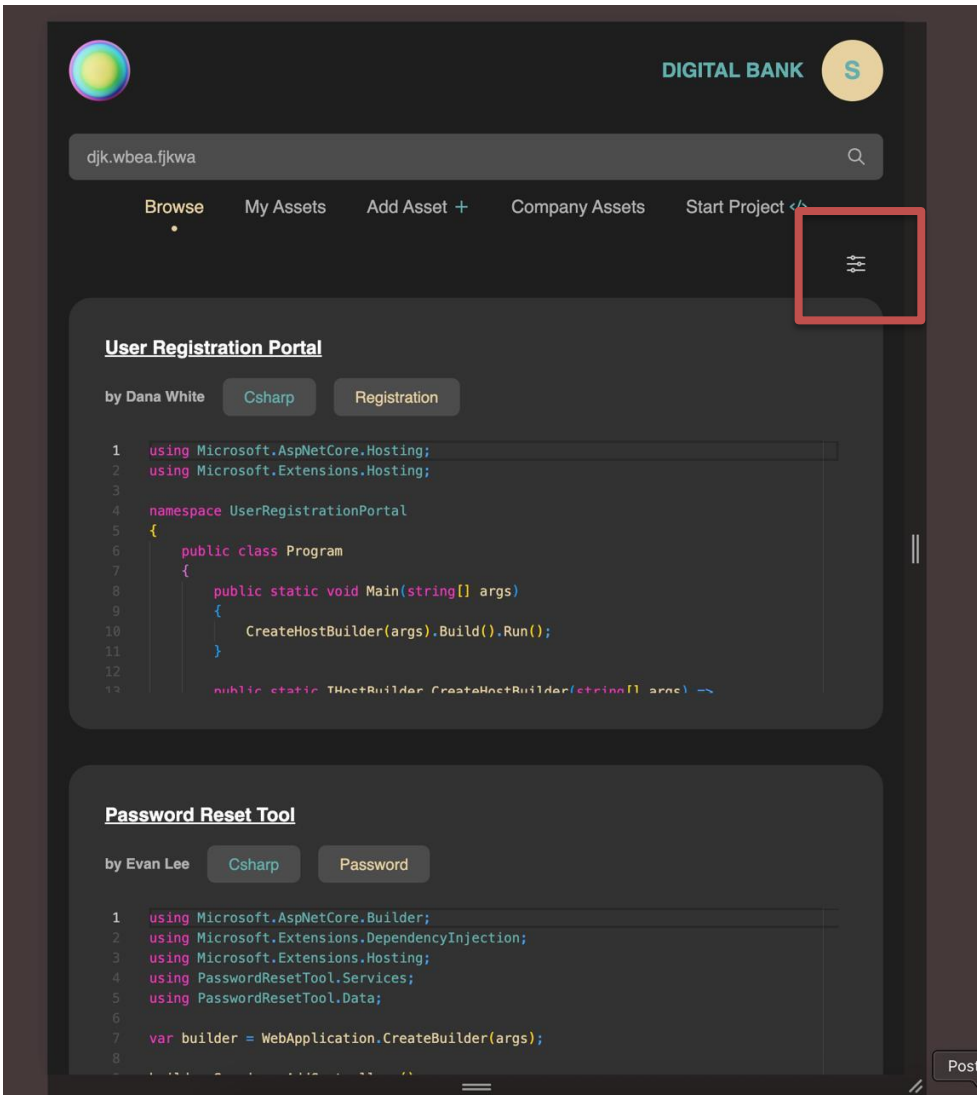

Рисунок 24 – Відображення застосунку на планшеті (рисунок виконаний самостійно)

У ході тестування за допомогою BrowserStack було перевірено відображення інтерфейсу на таких пристроях та конфігураціях:

а) мобільні телефони:

1) Samsung Galaxy S23 (Android 13);
2) Samsung Galaxy S22 (Android 12);
3) iPhone 14 (iOS 16);
4) iPhone 13 Mini (iOS 15);
5) iPhone 8 (iOS 11).

б) планшети:

1) iPad 9-го покоління (iOS 15);
2) Samsung Galaxy Tab S8 (Android 12).

в) браузери:

1) Windows 11: Chrome 135, Firefox 139 Beta, Edge 137 Beta;
2) macOS Monterey: Safari 15.6;
3) macOS Ventura: Safari 16.5.

Застосунок успішно пройшов тестування на всіх зазначених конфігураціях. Інтерфейс коректно адаптується до різних розмірів екранів, шрифт і компоненти змінюються відповідно до типу пристрою, а основний функціонал зберігає повну працездатність. Особливу увагу було приділено мобільній версії (див. рис. 25), зокрема правильному розташуванню форм, меню навігації та модальних вікон.

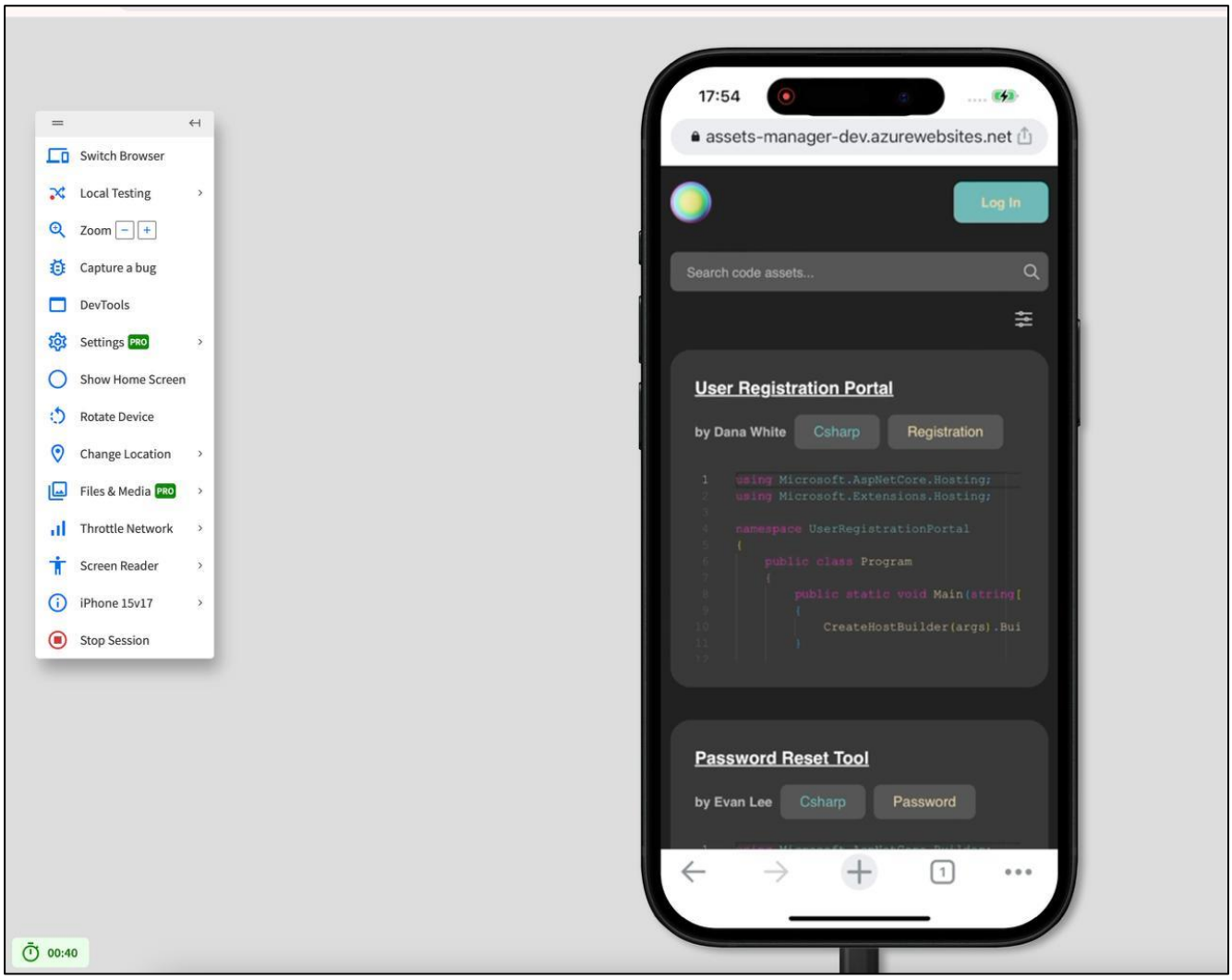

Рисунок 25 – Тестування застосунку за допомогою BrowserStack (рисунок виконаний самостійно)

Також проводилася ручна перевірка на декількох реальних пристроях з різними розмірами екранів, що дозволило переконатися у коректній роботі адаптивної верстки у реальному використанні (див. рис. 26) .

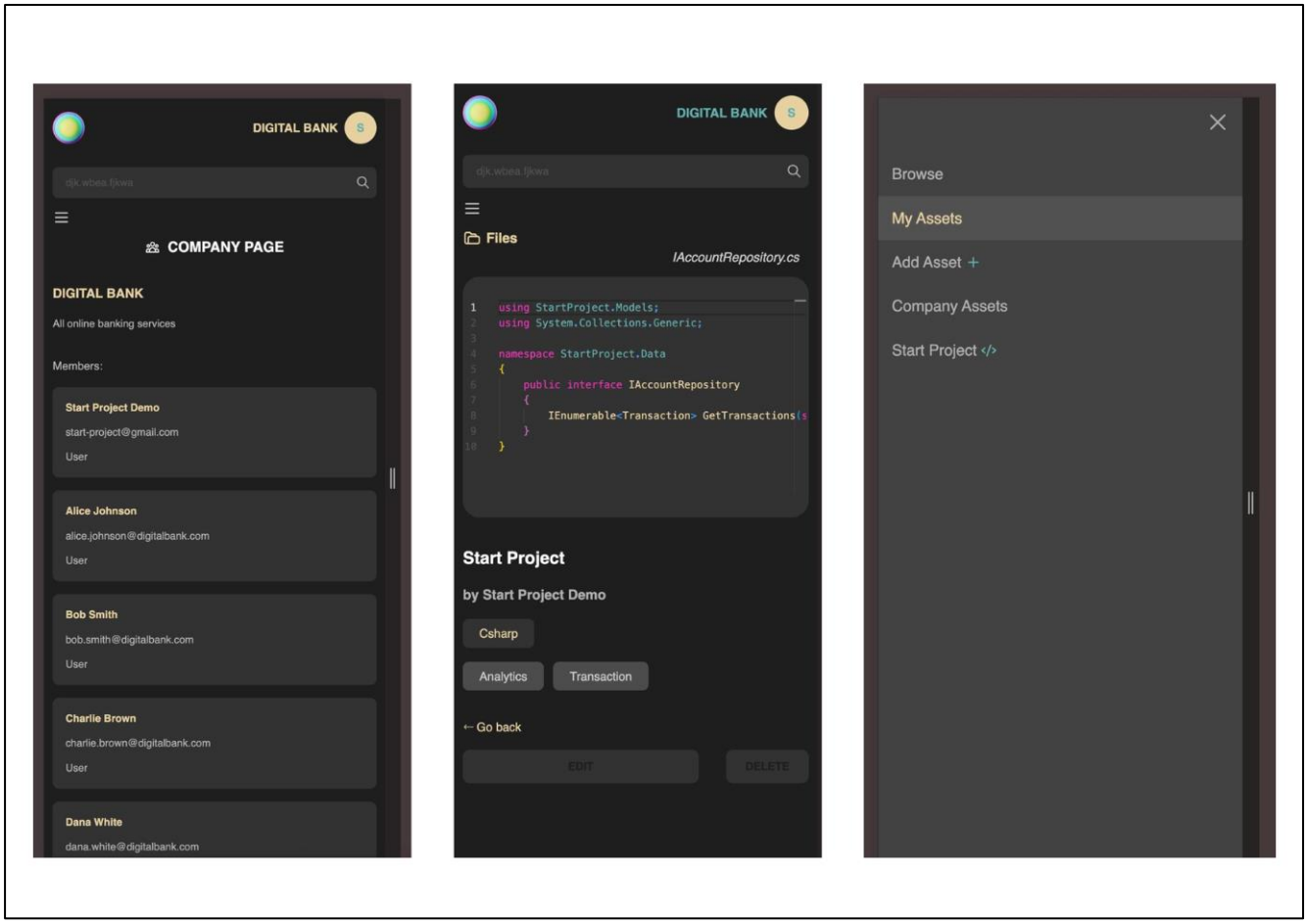

Рисунок 26 – Відображення застосунку на телефоні (рисунок виконаний самостійно)

У результаті проведеного тестування вдалося виявити та усунути низку помилок, пов’язаних як з некоректною поведінкою окремих компонентів, так і з відображенням інтерфейсу на різних пристроях. Систему було протестовано як на позитивні, так і на негативні сценарії використання, що дозволило досягти високого рівня стабільності, передбачуваності та зручності для кінцевого користувача.

Тестування стало важливим етапом у розробці програмного забезпечення, який забезпечив його якість та надійність.

ВИСНОВКИ

Під час написання кваліфікаційної бакалаврської роботи було проведено дослідження проблем управління кодовими базами та визначено основні труднощі, з якими стикаються розробники. На основі цього аналізу була розроблена клієнтська частина програмної системи для зберігання та управління фрагментами програмного коду, що сприяє підвищенню ефективності роботи команд розробників.

Розробка включала створення концептуальної моделі системи, що дозволило чітко визначити функціональні вимоги та потреби користувачів. Було спроектовано інтерфейс клієнтської частини та реалізовано її з використанням сучасних технологій програмування для забезпечення швидкого доступу до кодових фрагментів, їх аналізу та повторного використання.

У результаті роботи була спроектована клієнтська частина системи, яка дозволяє розробникам легко додавати, редагувати та шукати фрагменти коду, а також отримувати рекомендації щодо покращення його якості. Ця частина системи забезпечує ефективне управління кодом як індивідуально, так і в рамках компанії, сприяє командній роботі та покращує процес розробки нових проєктів.

Результатом кваліфікаційної роботи є спроектована клієнтська частина системи для управління кодовими фрагментами SNEEP&KEEP, яка підвищує ефективність командної роботи, знижує ризик помилок і скорочує час на розробку нових програмних продуктів. Ця система є корисним інструментом для незалежних розробників та компаній, що працюють над великими проєктами та мають значні обсяги повторюваного коду.

Апробація отриманих результатів відбулася у вигляді підготовки тез доповіді для конференції Інформаційні системи та технології (ICT-2024) (див. дод. Г), а також участі у науковій виставці XXIX Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті» (див. дод. Д), що засвідчило актуальність і практичну цінність проведеного дослідження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Zhang, Tanghaoran, et al. "How do Developers Adapt Code Snippets to Their Contexts? An Empirical Study of Context-Based Code Snippet Adaptations." IEEE Transactions on Software Engineering (2024). URL: https://ieeexplore.ieee.org/abstract/document/10510659/metrics#metrics (дата звернення 01.03.25)

2. About SnippetsLab. URL: https://www.renfei.org/snippets-lab/ (дата звернення 10.03.25)

3. About Code Snippets AI. URL: https://codesnippets.ai/about (дата звернення 10.03.25)

4. About Github Copilot. URL: https://github.com/features/copilot (дата звернення 11.03.25)

5. Димо, О. Б.. Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом. Diss..—Миколаїв, 2007. – 226с, 2012. URL: https://rep.nuos.edu.ua/server/api/core/bitstreams/d984524d-c38b-4673-9fd0-3b5a59ff2858/content (дата звернення 16.03.25)

6. Повне розуміння діаграми компонентів UML за допомогою легкого методу. URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/ (дата звернення 15.04.25)

7. Темна тема чи світла: що краще? Висновки на основі наукових публікацій URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/ (дата звернення 21.04.25)

8. Figma Documentation. URL: https://help.figma.com/hc/en-us/categories/360002042553-Figma-Design (дата звернення 22.04.25)

9. Why is a navigation menu important? URL: https://www.one.com/en/websitebuilder/navigation-menu#:~:text=A%20navigation%20bar%20allows%20you,right%20of%20the%20navigation%20menu. (дата звернення 22.04.25)

10. What Is Prompt Engineering? Definition, Elements, Techniques, Applications, and Benefits. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-prompt-engineering/ (дата звернення 22.04.25)

11. Shchoholiev S., Pavlenko P., Shirokopetleva M., Kobziev V. AI-powered Software Project Generation from Natural Language Input Using Pre-stored Code Assets in Vector Databases // Інформаційні системи та технології: матеріали 13-ї Міжнародної науково-технічної конференції (ICT-2024), 26–28 листопада 2024 р., Харків, Україна. – Харків, 2024. – Ч. 2. – С. 85–86. URL: https://ist-conf-nure.com.ua/IST-2024_part-2.pdf

12. Посилання на GitHub, де розташовані всі електронні матеріали до кваліфікаційної роботи. URL: https://github.com/NurePavlenkoPolina4/2025-B-PI-PZPI-21-4-Pavlenko-P-O.git