НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

_____комп'ютерних наук_____

(назва кафедри)

_____ _Голуб Б. Л.__

(підпис) (ПІБ)

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення для автоматизації створення HTML-розмітки інтерфейсів веб-сторінок за допомогою нейронних мереж»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми

___доцент к.т.н._____ _____ ___Вайганг Г.О.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____доцент к.т.н._____ _____ _____Боярінова Ю.Є.__

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Пркопчук А.Ю.__

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)
___к.т.н., доцент_____Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Прокопчук Андрій Юрійович

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення для автоматизації створення HTML-розмітки інтерфейсів веб-сторінок за допомогою нейронних мереж» затверджена наказом ректора НУБіП України №570 С від 08.04.2025

2. Термін подання завершеної роботи на кафедру____2025.05.25_____
(рік, місяць, число)

3. Вихідні дані до роботи:

Створення HTML-сторінок, застосування нейронних мереж

4. Перелік питань, що розглядаються:

1. Аналіз проблемної області
2. Моделювання предметної області
3. Проєктування програмної системи
4. Впровадження та експлуатація системи

Дата видачі завдання 08.04.2025.

Керівник бакалаврської кваліфікаційної роботи _____/Боярінова Ю.Є./
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/Прокопчук А.Ю. /
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI (Artificial Intelligence) – штучний інтелект, область комп’ютерних наук, що займається створенням інтелектуальних систем, здатних до навчання, аналізу та прийняття рішень.

API (Application Programming Interface) – інтерфейс програмування додатків, набір методів для взаємодії з іншими сервісами або програмами, наприклад, запити до нейромережі.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова, що використовується для оформлення зовнішнього вигляду веб-сторінок, визначення кольорів, шрифтів, відступів та адаптивності.

Firestore – хмарна NoSQL база даних від Google (частина Firebase), яка забезпечує реальночасове збереження і синхронізацію даних між клієнтом і сервером.

Firebase – хмарна платформа від Google для розробників, яка надає інструменти для аутентифікації, збереження даних, аналітики та інших сервісів.

HTML (Hypertext Markup Language) – мова розмітки гіпертексту, що використовується для створення структури та контенту веб-сторінок.

HTTP (Hypertext Transfer Protocol) – протокол прикладного рівня для передачі даних між веббраузером і сервером.

JS (JavaScript) – динамічна мова програмування, яка використовується для створення інтерактивної логіки на вебсторінках.

JSON (JavaScript Object Notation) – текстовий формат зберігання та обміну даними, який часто використовується для комунікації між клієнтом і сервером.

SPA (Single Page Application) – односторінковий застосунок, який динамічно оновлює контент на сторінці без повного перезавантаження.

UI (User Interface) – інтерфейс користувача, сукупність елементів на екрані, з якими взаємодіє користувач (кнопки, поля, повідомлення).

UX (User Experience) – користувацький досвід, якість взаємодії користувача з вебзастосунком у плані зручності, інтуїтивності та задоволення

від використання.

IDE (Integrated Development Environment) – інтегроване середовище розробки, в якому програміст пише код, виконує тестування, переглядає помилки (наприклад, Visual Studio Code).

DOM (Document Object Model) – об’єктна модель документа, яка представляє HTML-сторінку як дерево елементів, що може змінюватися скриптами.

CRUD (Create, Read, Update, Delete) – базові операції для роботи з даними: створення, читання, оновлення та видалення.

UX Flow – схема послідовної взаємодії користувача з системою, яка показує шлях користувача через інтерфейс.

GPT (Generative Pre-trained Transformer) – архітектура нейронної мережі, яка використовується для генерації тексту, в тому числі HTML-коду, на основі запитів.

CLI (Command Line Interface) – інтерфейс командного рядка, спосіб взаємодії з операційною системою або середовищем розробки через текстові команди.

ВСТУП

У дипломній роботі на тему «Програмне забезпечення для автоматизації створення HTML-розмітки інтерфейсів веб-сторінок за допомогою нейронних мереж» було виконано розробку вебзастосунку, який дозволяє користувачам автоматично генерувати HTML-код інтерфейсів за допомогою штучного інтелекту через простий і зручний інтерфейс у браузері.

Актуальність завдання. У сучасному світі створення вебінтерфейсів стало щоденною задачею для багатьох розробників, дизайнерів та фахівців без технічного бекграунду. Зі зростанням попиту на швидку розробку вебсторінок зростає і потреба в інструментах, які автоматизують рутинні процеси. Застосування нейронних мереж у генерації HTML-коду дозволяє значно спростити й прискорити створення користувацьких інтерфейсів навіть тим, хто не володіє мовами розмітки. Це робить тему роботи актуальною як з практичної, так і з науково-дослідної точки зору.

Мета розробки.

Метою роботи є створення вебдодатку, який дозволяє користувачу вводити текстовий запит і отримувати згенеровану HTML-розмітку за допомогою нейронної мережі. Результати генерації можна переглянути, скопіювати або зберегти до особистого кабінету. Додаток також забезпечує можливість реєстрації, входу, збереження запитів користувачем та керування особистим архівом згенерованих кодів.

Основні завдання, реалізовані в роботі:

- аналіз предметної області та визначення вимог до системи;
- розробка архітектури вебзастосунку;
- реалізація сторінок входу, реєстрації, генератора, кабінету;
- підключення нейромережевого API для генерації HTML-коду;
- забезпечення авторизації та зберігання даних у Firebase;
- створення адаптивного інтерфейсу для всіх типів пристроїв;
- тестування та перевірка стабільності роботи системи.
-

Методи та технології.

Для реалізації проєкту було використано такі вебтехнології:

- HTML / CSS / JavaScript — базові мови розмітки та стилізації;
- Firebase — для авторизації користувачів та збереження згенерованих даних у Firestore;
- AI API (GPT) — для генерації HTML-коду за текстовим запитом;
- DOM-маніпуляції та валідація форм — для обробки введення та виводу;
- GitHub / VS Code — для розробки та тестування.

Апробація програмного додатку.

Розроблений вебсайт був протестований на різних пристроях і браузерах, що підтвердило його стабільність, адаптивність та функціональність. Сервіс продемонстрував високу точність генерації HTML-коду, зручність використання інтерфейсу та гнучкість у збереженні результатів.

Структура роботи.

Дипломна робота складається з п’яти основних розділів:

Аналіз предметної області — визначено проблему, цілі, задачі, та огляд аналогічних рішень;

Моделювання предметної області — представлено об’єктне та функціональне моделювання, побудовано UML-діаграми та ER-модель;

Розробка інформаційного та програмного забезпечення — описано архітектуру, логіку реалізації сторінок, модулів, а також обґрунтовано вибір технологій;

Впровадження та експлуатація системи — розглянуто технічні вимоги, тестування, інсталяцію, перевірку функціоналу;

Висновки — підсумовано виконану роботу та ефективність системи. Дипломна робота містить понад __ сторінок, діаграми, графічні ілюстрації, а також приклади роботи реалізованого вебзастосунку.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

В епоху цифрових технологій веб-розробка займає ключову позицію у створенні онлайн-сервісів, презентацій, електронної комерції, медіа та інформаційних платформ. Проте процес створення веб-інтерфейсів вимагає технічних знань, насамперед у мовах розмітки та стилізації — HTML та CSS. Це ускладнює доступ до цієї сфери для початківців або нетехнічних спеціалістів.

У зв’язку з цим актуальним стає застосування інструментів автоматизації, що дозволяють згенерувати готову HTML-розмітку лише на основі текстового запиту. Використання нейронних мереж та штучного інтелекту значно розширює можливості таких платформ, роблячи їх інтелектуальними асистентами у створенні UI-компонентів.

Особливу цінність мають онлайн-рішення, що не потребують встановлення, реєстрації на сторонніх сервісах або складного навчання. Користувач повинен мати змогу зайти на сайт, пройти реєстрацію, ввести запит — і отримати функціональний HTML-фрагмент.

1.2 Мета та завдання дослідження

Мета роботи — створення повнофункціонального вебзастосунку, що дозволяє користувачу у зручному онлайн-інтерфейсі вводити запити до генератора HTML-коду, зберігати та керувати результатами в особистому кабінеті.

Основні завдання:

- розробити механізм реєстрації та входу в систему;
- реалізувати генератор HTML-коду на основі текстових запитів;
- забезпечити можливість збереження згенерованого коду до особистого кабінету;

- створити інтерфейс кабінету з можливістю перегляду, копіювання та видалення результатів;
- реалізувати функцію безпечного виходу з акаунту; забезпечити безперервну роботу сайту у публічному інтернеті (наразі як прототип, у перспективі — з доменом та SSL).

1.3 Аналіз існуючих рішень

На ринку вебтехнологій існує кілька продуктів, які частково вирішують подібні завдання(див. Таблиця 1.1):

ChatGPT, Gemini, Claude (AI-асистенти)

Вони дозволяють згенерувати HTML-фрагменти за текстовим запитом, проте: не мають структури проєкту; не надають особистого кабінету користувача; не зберігають історію запитів та результатів.

CodePen, JSFiddle

Онлайн-редактори з підтримкою HTML/CSS/JS, але: не підтримують генерацію коду через AI; потребують знань кодування; не мають персоналізації під користувача.

GitHub Copilot

AI-помічник для IDE, орієнтований на досвідчених розробників. Недоступний у вигляді окремого вебсайту для загального користувача без технічних знань.

Таблиця 1.1

Характеристики існуючих рішень

Характеристика	**ChatGPT / Gemini / Claude**	**CodePen / JSFiddle**	**GitHub Copilot**
Генерація HTML за текстом	+	-	+

Наявність UI-інтерфейсу	-	✓	-
Збереження історії генерацій	-	✓ (тільки вручну)	-
Персональний кабінет користувача	-	-	-
Простота для початківців	✓	-	-
Придатність без знання коду	✓	-	-
Наявність вебверсії	✓	✓	-
Публічний доступ без IDE	✓	✓	-
Можливість інтеграції з базою даних	-	-	✓ (у межах IDE)

1.4 Унікальність запропонованої системи

Відмінністю створеної системи є поєднання простоти для користувача та автоматизації з використанням штучного інтелекту. Користувач бачить лише інтерфейс із полем запиту та кнопками дій:

- генерація коду;
- збереження коду;

- перехід до особистого кабінету;
- видалення або копіювання результатів;
- вихід з акаунту.

І все це реалізовано у вигляді сайту, доступного онлайн, без необхідності завантаження ПЗ.

Основна унікальність розробленого застосунку полягає у поєднанні трьох ключових аспектів:

- простота інтерфейсу та UX для користувача без технічних навичок;
- застосування нейронної мережі для генерації HTML-коду з текстового опису;
- збереження результатів у персональному кабінеті з можливістю подальшого управління.

Це дозволяє говорити про гібридну модель взаємодії: з одного боку, користувач взаємодіє зі звичним інтерфейсом вебзастосунку, а з іншого — отримує доступ до інтелектуального ядра, яке виконує складні обчислення на сервері. Така комбінація зазвичай недоступна в загальнодоступних онлайн-сервісах безкоштовного класу.

Основні особливості, що відрізняють запропоновану систему від аналогів:

1. Фокус на HTML-генерацію за допомогою штучного інтелекту. Більшість популярних систем або взагалі не мають функціоналу генерації за текстовим запитом, або ж надають її у форматі «допоміжного інструменту». У нашій системі ця функція — основа функціональності: користувач одразу бачить поле для запиту, натискає кнопку — і отримує HTML-код. Усі інші функції побудовані навколо цієї логіки.
2. Повноцінний вебінтерфейс із системою аутентифікації. У той час як більшість AI-сервісів працюють у режимі сесійної взаємодії без авторизації (або з тимчасовою), наша система реалізує персоналізований кабінет користувача. Це відкриває можливість зберігати згенеровані фрагменти, керувати ними, переглядати історію, видаляти або копіювати.
3. Простота користування навіть для людей без технічного бекграунду.

Усі дії користувача обмежуються лише кількома інтуїтивно зрозумілими кнопками: введення запиту; генерація результату;збереження коду;перехід до кабінету;видалення або копіювання фрагменту;вихід із системи.

4. Інтерфейс не перевантажений зайвими елементами, що робить його інтуїтивно зрозумілим навіть для новачків.

5. Використання Firebase як безпечної та масштабованої платформи для бекенду.

Інтеграція з Firebase дозволяє уникнути проблеми підтримки серверної логіки — усе зберігається у хмарі, забезпечується авторизація, захист доступу до даних за UID, і надалі можлива масштабована підтримка через Firebase Hosting. Це значно знижує поріг входу для розгортання проєкту у продакшн.

6. Можливість використання як SaaS-рішення у майбутньому.

Враховуючи реалізацію у вигляді вебзастосунку з авторизацією та базою даних, систему можна без змін у логіці використовувати як готову платформу для впровадження моделі SaaS (Software as a Service). Це створює перспективи монетизації, інтеграції з преміум-функціоналом (наприклад, CSS-генерація, експортування в ZIP, перегляд мобільної верстки тощо).

Система, запропонована у цій дипломній роботі, демонструє унікальний баланс між складною технічною реалізацією і простотою для кінцевого користувача. На відміну від аналогів, вона не лише генерує HTML-код, але й дозволяє зручно керувати результатами у персональному кабінеті, забезпечує безпеку через Firebase та має потенціал до масштабування як повноцінна онлайн-платформа. Таким чином, вона вирішує реальну потребу ринку — надати простий, але потужний інструмент для автоматизованого створення інтерфейсної розмітки вебсторінок.

1.5 Функціональна структура системи

Функціональна структура системи — це набір основних підсистем, які забезпечують коректну роботу вебзастосунку, що дозволяє користувачам генерувати HTML-код на основі текстового запиту, а також зберігати та

управляти результатами у своєму особистому кабінеті. Для забезпечення високої доступності, стабільності та масштабованості всі функціональні блоки розділені на логічні компоненти, які відповідають за окремі напрямки взаємодії користувача з системою.

1. Користувацька автентифікація (реєстрація / вхід)

Цей функціональний блок відповідає за створення нового облікового запису користувача, перевірку даних при вході та збереження сеансу. Реалізовано з використанням Firebase Authentication, який надає:

- створення облікового запису за допомогою email та паролю;
- збереження пароля у хешованому вигляді;
- перевірку наявності облікового запису при авторизації;
- автоматичне відновлення сеансу після перезавантаження сторінки;
- захист від несанкціонованого доступу до персональних даних інших користувачів.

Ця система дозволяє гарантувати безпеку та індивідуалізацію кожної сесії.

2. Генератор HTML-коду на основі текстового запиту

Центральна функція вебзастосунку — обробка текстового запиту користувача (prompt) та генерація HTML-коду. Алгоритм взаємодії побудований наступним чином:

- користувач уводить запит у відповідне поле;
- після натискання кнопки «Згенерувати» запит надсилається на сервер (або до зовнішнього API з підтримкою генерації, наприклад OpenAI API);
- результат у вигляді HTML-розмітки виводиться в окремому вікні на сторінці;
- користувач має змогу перевірити код, скопіювати його, зберегти або очистити поля.

Ця функція дозволяє не лише автоматизувати рутинну розмітку, а й адаптувати генерацію під індивідуальні запити користувача.

3. Збереження результатів у Firestore

Після успішної генерації HTML-коду користувач має можливість зберегти результат у власному обліковому записі. Система виконує:

- прив’язку збереженого фрагменту до userId поточного користувача;
- фіксацію часу збереження (timestamp);
- зберігання як тексту запиту, так і згенерованого HTML-коду;
- можливість доступу до результатів лише цим користувачем (через правила доступу Firestore).

Таким чином, база даних постійно оновлюється індивідуально для кожного користувача, і жоден інший користувач не має доступу до чужих результатів.

4. Особистий кабінет (User Dashboard)

Користувачі мають доступ до особистого кабінету після авторизації. У ньому реалізовано такі функції:

- відображення списку збережених HTML-фрагментів, включаючи дату та назву запиту;
- копіювання коду в буфер обміну натисканням на відповідну іконку;
- видалення непотрібного фрагменту за допомогою кнопки видалення;
- динамічне оновлення списку без перезавантаження сторінки (через Firestore SDK).

Кабінет дозволяє користувачам швидко повертатися до попередніх результатів та використовувати їх повторно або редагувати за потреби.

5. Інтерфейс керування сесією (Log Out)

Для завершення сесії користувач має кнопку **«Log Out»**, яка виконує:

- видалення токену автентифікації;
- завершення поточної сесії;
- перенаправлення користувача на головну або сторінку входу.

Ця функція гарантує безпечне завершення роботи та недопущення сторонніх осіб до облікового запису користувача в разі використання спільного пристрою.

6. Інтерфейс користувача (UI/UX)

Уся взаємодія реалізована у вигляді SPA-додатку (Single Page Application), що забезпечує швидкість і плавність переходів між екранами. Особливості UI:

- простота та інтуїтивність: кожен елемент має чітке призначення;
- адаптивність: сайт відображається коректно як на ПК, так і на мобільних пристроях;
- мінімалізм у стилістиці: використано Tailwind CSS для чистого й сучасного вигляду;
- швидке реагування на дії користувача.

Функціональна структура системи чітко розділена на компоненти, кожен з яких відповідає за окрему частину користувацького досвіду: від входу в систему до збереження результатів і виходу. Такий підхід дозволяє забезпечити гнучкість, масштабованість та безпеку, що робить запропоновану систему не просто демонстраційним проєктом, а потенційно готовим рішенням для комерційного використання або розширення у майбутньому. Чітка структура дозволяє легко підтримувати систему, впроваджувати оновлення та відстежувати помилки на будь-якому етапі взаємодії користувача з платформою

1.6 Загальна схема взаємодії

Загальна схема взаємодії користувача з вебзастосунком охоплює повний цикл роботи з системою — від входу до завершення сесії. Вона демонструє логіку переходів між етапами та роль кожного функціонального блоку у забезпеченні стабільного й інтуїтивного користувацького досвіду. Кожен крок реалізовано через взаємодію фронтенду з базою даних і сервісом автентифікації.

1. Вхід до системи / реєстрація нового користувача

Користувач, який уперше заходить на сайт, має можливість створити новий обліковий запис через форму реєстрації. Вона складається з двох основних полів: email і пароль. Після введення даних відбувається звернення до Firebase Authentication, який:

- перевіряє правильність форми email;
- шифрує та зберігає пароль;
- створює унікальний UID для кожного користувача;
- підтверджує успішну реєстрацію або повертає повідомлення про помилку (наприклад, вже зареєстрований email).

Після реєстрації або авторизації користувач переходить до головної сторінки застосунку, де доступний генератор HTML.

2. Введення текстового запиту

Центральним елементом є текстове поле (textarea), у яке користувач вводить природномовний запит — наприклад, “зроби адаптивну картку товару з кнопкою купити” або “створи простий футер з трьома колонками”.

Цей запит передається функції генерації, яка звертається до штучного інтелекту через API (наприклад, OpenAI). Запит обробляється в режимі реального часу, а результат — фрагмент HTML-коду — повертається на сторінку у вигляді окремого текстового блоку для перегляду.

3. Отримання та обробка результату

Згенерований код з’являється під текстовим запитом. Користувачу доступні наступні опції:

- Зберегти результат — код та пов’язаний із ним запит надсилаються до Firebase Firestore, де зберігаються під UID користувача;

- Очистити поле — видаляє вміст як запиту, так і результату, повертаючи інтерфейс у початковий стан;

- Повторна генерація — дозволяє скоригувати текст запиту й отримати оновлену версію HTML-коду.

Усі операції виконуються без перезавантаження сторінки, завдяки використанню JavaScript та реактивного підходу до рендерингу.

4. Робота з особистим кабінетом

У правому верхньому куті або через відповідне меню користувач має доступ до особистого кабінету (dashboard). Тут відображаються всі збережені ним результати:

- кожна генерація представлена у вигляді карточки з кодом;

- Видалити — повністю прибирає запис із бази даних (після підтвердження).

Кабінет автоматично синхронізується з Firestore, тому зміни відображаються одразу, без додаткових запитів.

5. Вихід з системи

Після завершення роботи користувач натискає кнопку “Log out”, яка:

- викликає функцію завершення сесії у Firebase;

- очищує локальне збереження токену;

- перенаправляє користувача на сторінку входу.

Це гарантує завершення автентифікаційної сесії та безпеку персональних даних.

Загальна схема взаємодії користувача із застосунком побудована навколо принципів простоти, логічності та безпеки. Всі етапи — від введення запиту до збереження результатів — чітко розділені й реалізовані через зручний та мінімалістичний інтерфейс. Наявність особистого кабінету, швидке копіювання

та можливість видалення старих фрагментів створюють зручне середовище для щоденного використання сервісу. Така схема робить систему доступною як для новачків, так і для досвідчених користувачів.

1.7 Інфраструктура розгортання системи

Розроблений вебзастосунок функціонує як клієнт-серверна система, що реалізована у форматі односторінкового застосунку (SPA) та повністю працює через веб-браузер. Для забезпечення надійної та масштабованої роботи без використання традиційного бекенду було обрано архітектуру типу Serverless, що базується на платформі Firebase. Це дозволяє досягти гнучкості, швидкого розгортання і високої продуктивності без необхідності налаштовувати власні сервери.

Архітектура системи

Загальна інфраструктура включає три ключові компоненти:

- Фронтенд (інтерфейс користувача);
- Сервіси Firebase (авторизація, база даних, хостинг);
- Штучний інтелект (API-запити до моделі генерації HTML).

1. Фронтенд (Next.js, HTML, CSS, JS)

Фронтенд написаний з використанням Next.js — сучасного фреймворку на базі React, що дозволяє створювати швидкі, адаптивні та SEO-оптимізовані SPA-додатки. Його переваги:

- серверний рендеринг (за потреби);
- покращена продуктивність;
- автоматична маршрутизація;
- підтримка статичної генерації сторінок;
- можливість гнучкого управління станом через React.

Інтерфейс користувача побудований за допомогою HTML та CSS (зокрема, з використанням утиліт Tailwind CSS), що дозволяє швидко створювати стилізовані компоненти, адаптовані під мобільні та десктопні пристрої.

2. Firebase як серверна інфраструктура

Усі функції, які традиційно виконує сервер, делеговано до сервісів Firebase:

- Firebase Authentication — забезпечує реєстрацію, вхід, зберігання паролів, підтримку сесій та безпечний вихід. Реалізовано авторизацію на основі електронної пошти та паролю.
- Cloud Firestore — хмарна база даних у реальному часі, яка зберігає згенеровані HTML-фрагменти. Кожен фрагмент асоціюється з конкретним користувачем за допомогою UID, що гарантує ізоляцію даних.
- Firebase Hosting (у подальшій реалізації) — дозволяє опублікувати застосунок у публічному доступі через HTTPS, автоматично надаючи SSL-сертифікат, кешування та швидкий глобальний CDN. Це забезпечить:високу швидкість завантаження; доступність застосунку 24/7; автоматичну доставку нових версій сайту.

3. Підключення до AI через API

Окрема частина інфраструктури — взаємодія з нейронною мережею, яка генерує HTML-код на основі запиту. Компонент працює наступним чином:

- користувач вводить текстовий запит;
- запит надсилається через HTTP до стороннього API (наприклад, OpenAI або аналогічного);
- у відповідь надходить HTML-код;
- код виводиться на екран і за бажанням зберігається в Firestore.

Таким чином, застосунок не зберігає й не обробляє модель локально, а лише взаємодіє з нею як із зовнішнім сервісом.

Інфраструктура вебзастосунку побудована на сучасних хмарних рішеннях, які забезпечують надійну, масштабовану та економічно ефективну основу для роботи онлайн-сервісу. Завдяки використанню Firebase як бекенд-платформи та Next.js для побудови фронтенду вдалося реалізувати зручний, безпечний і швидкодіючий застосунок без складної серверної інфраструктури.

Такий підхід дозволяє фокусуватись на функціональності та інтерфейсі, не витрачаючи ресурси на підтримку інфраструктури

1.8 Безпека та персональні дані

Питання безпеки користувацьких даних є пріоритетним. Система гарантує:

Шифрування паролів — збереження паролів у хешованому вигляді через Firebase Authentication;

Захист доступу — дані в Firestore ізольовані, і доступ до них мають лише авторизовані користувачі з відповідним UID;

Реалізація безпечного виходу — через кнопку Log Out, що очищує сесію та припиняє доступ до особистих даних;

Використання HTTPS — для захищеної передачі даних між користувачем та сервером (на продакшн-хостингу).

1.9 Технічне обґрунтування вибору інструментів

Усі технології, використані у проекті, відповідають принципам відкритості, швидкодії та простоти інтеграції.

Таблиця 1.2

Порівняння технологій для розробки

Компонент	**Технологія**	**Причина вибору**
Генерація HTML	JavaScript + AI API	Просте підключення, швидкий результат
Інтерфейс	HTML + CSS	Стандарти браузера
Авторизація	Firebase Auth	Просте API, безпека, інтеграція з UI
Зберігання даних	Firestore	Гнучка NoSQL модель, масштабованість
Хостинг	Firebase Hosting	SSL, швидкість, CDN

1.10 Логіка та структура взаємодії користувача із системою

Взаємодія користувача з вебзастосунком побудована за принципом максимальної простоти, інтуїтивності та логічної послідовності дій. Основною метою інтерфейсу є надання користувачеві можливості легко виконувати ключові дії — від генерації HTML-коду до його збереження, перегляду, копіювання чи видалення — без необхідності вивчення інструкцій чи технічних знань.

Структура користувацького потоку

Кожен користувач проходить наступний типовий шлях у системі:

1. Реєстрація / Вхід
   - Користувач потрапляє на екран авторизації.
   - Має можливість зареєструватися, вказавши email і пароль.
   - Після успішного входу відкривається доступ до основного функціоналу.
   - Авторизація відбувається через Firebase Authentication, що забезпечує захищену обробку облікових даних.
2. Сторінка генерації HTML

Інтерфейс містить: поле для введення текстового запиту; кнопку "Згенерувати", яка відправляє запит до AI API; область для відображення згенерованого коду; кнопку "Очистити", яка обнуляє обидва поля; кнопку "Зберегти", яка записує результат у Firestore.

3. Робота з генераціями

- Якщо результат збережено, він буде доступний у розділі Особистий кабінет.
- Кабінет показує список усіх збережених генерацій, прив'язаних до UID користувача.
- Біля кожної генерації розміщено:
- Кнопку видалити — остаточно видаляє об’єкт з бази.

4. Завершення роботи

- Кнопка "Log out" завершує сесію і повертає користувача до екрана входу.
- При цьому локальні дані очищуються, а доступ до Firestore припиняється.

Технічні аспекти взаємодії

- Кожна дія користувача супроводжується асинхронною обробкою через JavaScript.
- Запити до AI здійснюються через HTTP-запит до зовнішнього API.
- Доступ до Firestore налаштований на основі UID користувача: кожен бачить лише власні записи.
- Сторінка кабінету динамічно оновлюється при видаленні, без перезавантаження.
- Стан користувача контролюється у режимі реального часу — система знає, хто авторизований, і відображає відповідний інтерфейс.

Особливості UX / UI

- Всі кнопки мають підказки (tooltip), індикатори завантаження та статуси.
- Застосовано адаптивну верстку, яка коректно відображається на смартфонах, планшетах і десктопах.
- Візуальні елементи мінімалістичні, з фокусом на основну дію — запит і генерацію.

Розроблена логіка взаємодії між користувачем і системою базується на простій та прозорій структурі, яка не потребує додаткових знань у програмуванні. Кожен користувач, незалежно від технічного рівня, може пройти повний цикл роботи: від входу до отримання і збереження HTML-фрагменту. Використання Firebase як бази даних та системи автентифікації дозволило забезпечити високий рівень безпеки і стабільності. Таким чином, інтерфейс і логіка поведінки системи повністю відповідають вимогам зручності, швидкодії та функціональної завершеності.

1.11 Висновки до розділу 1

У першому розділі дипломної роботи було проведено системний аналіз предметної області, що охоплює створення вебзастосунку для генерації HTML-коду на основі текстових запитів з використанням технологій штучного інтелекту. Розглянуто актуальність теми в контексті автоматизації веб-розробки та підвищення доступності інструментів для створення інтерфейсів користувача без потреби в знаннях програмування.

Сформульовано мету дослідження — розробка інтуїтивно зрозумілого онлайн-сервісу, що дозволяє здійснювати генерацію, збереження та управління HTML-фрагментами. Окреслено основні завдання, серед яких: побудова реєстрації, генератора, особистого кабінету та функціоналу керування результатами. Здійснено порівняльний аналіз аналогів, таких як ChatGPT, CodePen та GitHub Copilot, що дозволило виявити їх обмеження та обґрунтувати необхідність створення нового рішення.

Особливу увагу приділено унікальності запропонованої системи, яка поєднує простоту, доступність і інтелектуальні можливості. Описано функціональну структуру та логіку взаємодії користувача з системою, що ґрунтується на принципах зручності та швидкодії. Пояснено, як відбувається генерація, збереження, перегляд та видалення результатів. Проаналізовано технічне середовище і стосовні технології: HTML/CSS, JavaScript, Firebase, а також Firebase Firestore та Authentication як основні інструменти для реалізації backend-функцій.

Наведено інфраструктурну модель вебзастосунку — сайт доступний у браузері, без необхідності встановлення, що підвищує зручність для кінцевого користувача. Особливу увагу приділено питанням безпеки: шифрування даних, ізоляція доступу до персональної інформації, обмеження прав доступу.

Таким чином, цей розділ заклав теоретичне та концептуальне підґрунтя для розробки практичного вебзастосунку, чітко окреслив функціональні вимоги, обґрунтував вибір інструментів та описав користувацькі сценарії. Ці результати є основою для подальшого моделювання та реалізації системи у наступних

розділах дипломної роботи

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

Моделювання — це фундаментальний етап розробки будь-якої програмної системи, що дозволяє на ранніх етапах формалізувати логіку роботи сервісу, визначити його структуру, функціональні блоки, взаємодії між користувачем та компонентами, і забезпечити майбутню масштабованість.

В межах даної дипломної роботи, що присвячена розробці вебсайту для генерації HTML-коду за допомогою текстових запитів, моделювання використовувалося за нступними напрямками.

Формалізація функціоналу: визначити повний список дій, які користувач може виконувати у системі — від реєстрації до генерації коду та управління збереженими результатами;

Проєктування структури даних: сформувати моделі користувачів та результатів генерації, описати їх атрибути та зв’язки;

Планування логіки взаємодії: описати, у якому порядку користувач буде переходити від однієї дії до іншої, за допомогою діаграми активностей;

Проєктування моделі зберігання даних: розробити логічну схему бази даних, у даному випадку — Firebase Firestore, та оптимізувати доступ до інформації з урахуванням авторизації;

Забезпечення масштабованості: створити таку структуру системи, яка в подальшому дозволить безболісно додавати новий функціонал без кардинального переписування логіки.

Моделювання в цій роботі було реалізоване за допомогою стандартів UML (Unified Modeling Language), які є міжнародно визнаними для графічного опису об’єктно-орієнтованих систем. Вибір UML обумовлений його універсальністю, високим рівнем деталізації та зрозумілою семантикою.

Зокрема, у дипломному проєкті було застосовано такі види діаграм:

- Use Case Diagram (Діаграма прецедентів) — для опису основних сценаріїв взаємодії користувача з системою;

- Class Diagram (Діаграма класів) — для опису об’єктної структури даних;
- Activity Diagram (Діаграма активностей) — для візуалізації логіки переходу між станами в межах сесії користувача;
- ER-діаграма — адаптована для Firebase Firestore, яка відображає структуру колекцій та документів у NoSQL-базі.

Завдяки використанню моделювання як окремого етапу, було досягнуто наступних результатів: зменшено кількість помилок у проєктуванні; виявлено надлишкові або дубльовані дії; спрощено реалізацію логіки на етапі програмування; підвищено узгодженість між фронтендом і бекендом; створено наочну документацію, яку можна використовувати як технічне завдання для розробників або презентацію для інвесторів/експертів.

Таким чином, моделювання виступає не просто як формальність, а як дієвий інструмент планування, який значно підвищує ефективність розробки та полегшує подальшу підтримку проєкту.

2.2 Моделювання користувацької взаємодії (діаграма прецедентів)

Діаграма прецедентів демонструє взаємодію користувача з функціональними можливостями вебзастосунку, який дозволяє генерувати HTML-код за допомогою текстових запитів. На цій діаграмі зображено умовного користувача (актор), який має доступ до певного набору дій у системі. Серед них — реєстрація нового облікового запису, вхід до особистого кабінету, введення запиту для генерації коду, перегляд отриманого результату, збереження HTML-розмітки, перегляд історії збережених генерацій, копіювання результату у буфер обміну, видалення непотрібних записів і завершення сесії через вихід з системи.

Ця діаграма відображає всі основні сценарії, які користувач може реалізувати під час взаємодії з сайтом. Вона допомагає зрозуміти, як саме реалізована взаємодія між користувачем і функціоналом застосунку, та дозволяє

переконатися, що всі ключові можливості враховані при розробці. Таким чином, діаграма прецедентів — це концентроване уявлення про поведінку користувача в системі, яка автоматизує створення HTML-коду.

Ця діаграма відображає основні сценарії взаємодії користувача з вебсайтом.

В центрі — користувач, який виконує дії:

Основні сценарії (прецеденти):

Реєстрація нового облікового запису

Користувач створює новий акаунт шляхом введення email та пароля. Дані зберігаються у Firebase Authentication, що забезпечує безпечну ідентифікацію користувача. Цей сценарій є першим етапом взаємодії з системою.

Авторизація (вхід до системи)

Якщо обліковий запис уже існує, користувач може увійти до системи, ввівши свої облікові дані. У разі успішної авторизації — відкривається доступ до функціоналу генерації HTML і особистого кабінету.

Збереження результату у Firestore

Користувач має можливість зберегти отриманий результат до своєї персональної бази даних, яка реалізована у Firestore. Збереження виконується лише за бажанням користувача.

Перегляд історії генерацій у особистому кабінеті

Користувач може відкрити кабінет, у якому відображається список збережених HTML-фрагментів. Дані фільтруються автоматично за UID користувача, забезпечуючи приватність.

Видалення запису

Якщо певний код уже не потрібен користувачу, його можна легко видалити зі сховища, не зачіпаючи інші записи.

Завершення сесії (вихід із акаунту)

В системі реалізовано безпечний вихід, який очищує сесію та припиняє доступ до персоналізованих даних до наступного входу.

Призначення діаграми прецедентів:

Цей тип моделі дозволяє: зрозуміти, який мінімальний і повний набір функцій повинен бути реалізований; визначити межі системи (тобто де закінчуються її обов’язки і починається зовнішній вплив); розмежувати ролі та права доступу, що є важливою складовою у забезпеченні інформаційної безпеки; забезпечити спільне бачення функціоналу серед розробників, тестувальників, дизайнерів і замовника.

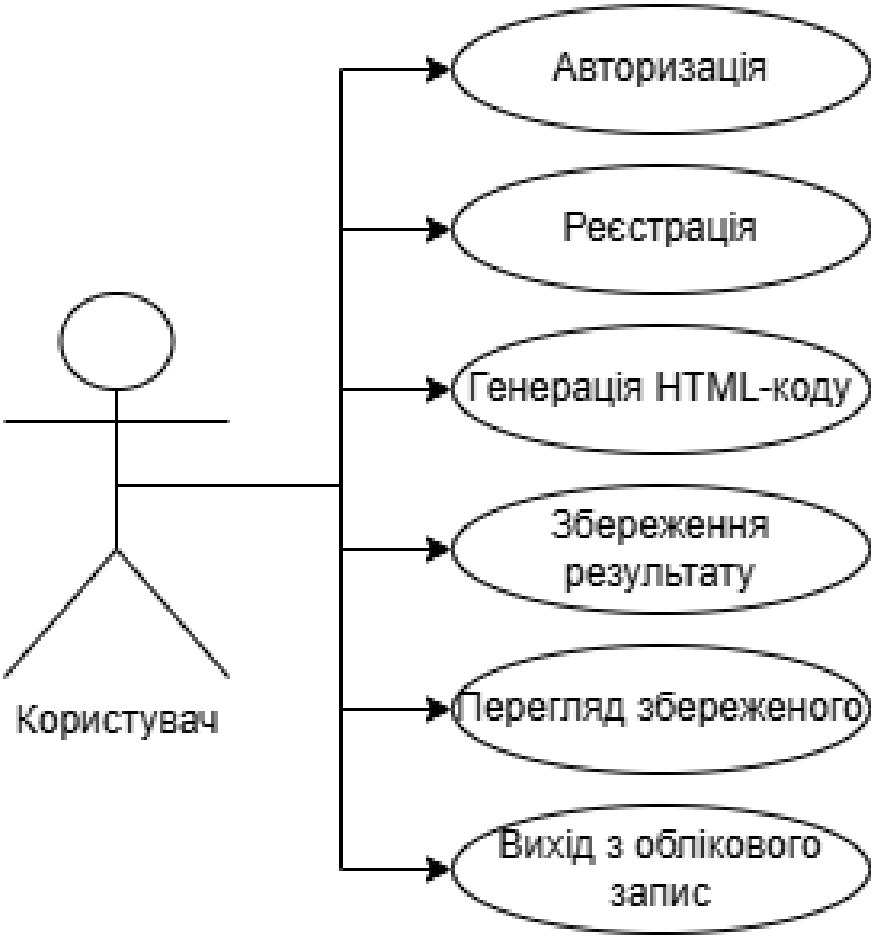

Рис 1. Use Case Діаграма

Діаграма показує чітку ізольовану взаємодію з основними блоками системи, що відповідає реальним діям користувача на сайті.

2.3 Об’єктно-орієнтоване моделювання (діаграма класів)

Діаграма класів у дипломному проєкті демонструє логічну структуру основних об'єктів вебсайту і їхні зв’язки (Рис.2). У системі виділено два ключові

класи:

User: містить унікальний ідентифікатор (userId), email та пароль (в хешованому вигляді). Він також зберігає зв'язок з усіма сніпетами, які користувач створив.

Snippet: кожен об’єкт цієї сутності містить текстовий запит (prompt), HTML-код (result), дату створення (createdAt) і посилання на користувача, який його згенерував (userId).

Між класами існує відношення "один до багатьох": один користувач може мати декілька сніпетів, однак кожен сніпет належить лише одному користувачу. Ця модель відповідає архітектурі бази даних Firebase, яка використовується у проекті, та чітко описує, як дані зберігаються, обробляються і виводяться.

Для розуміння структури даних у системі використовується діаграма класів, яка демонструє сутності, що беруть участь у функціонуванні генератора HTML, та зв'язки між ними. У нашій системі це:

User — основний об’єкт, що містить ідентифікацію користувача (ID, email);

Prompt — текстовий запит користувача;

GeneratedCode — результат, який повертає AI (HTML-текст);

SavedResult — збережена версія коду, яка зберігається у Firestore (прив’язана до UID);

AuthSession — об’єкт поточної сесії (Firebase Auth).

Зв'язки:

Один User → багато SavedResult;

Кожен SavedResult → має GeneratedCode + дату створення;

Prompt не зберігається окремо, але може логуватись при необхідності.

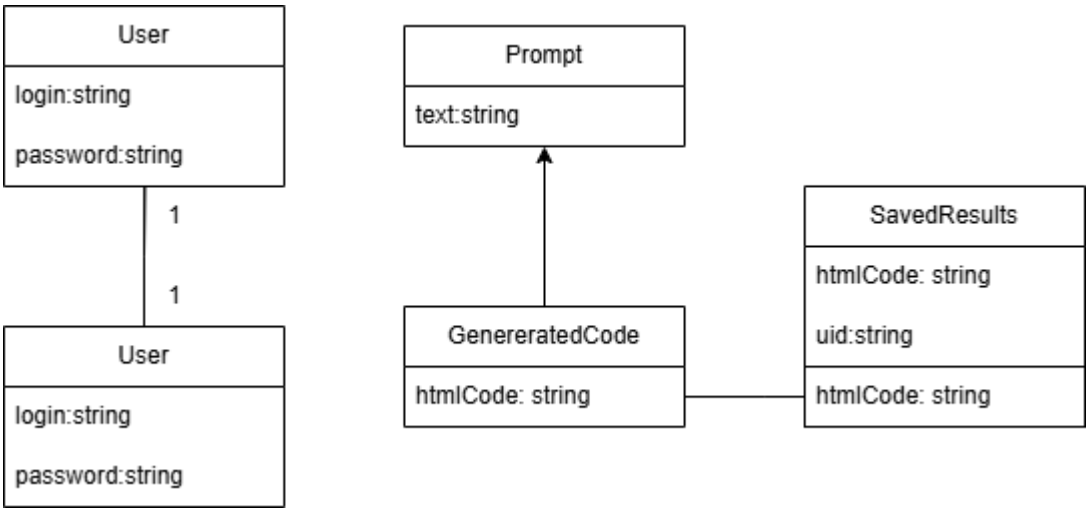

Рис 2. Діаграма класів

2.4 Діаграма активностей (activity diagram)

Діаграма активностей представляє покрокову логіку використання сайту(Рис.3). Вона починається з першої дії користувача — відкриття сторінки реєстрації або входу. Далі, у разі успішної авторизації, користувач переходить до головної сторінки генератора.

На цій сторінці він вводить текстовий запит у поле для генерації, після чого система надсилає запит до вбудованого модуля AI, що повертає HTML-код. Користувач бачить результат і може або скопіювати його, або зберегти до свого кабінету. Якщо натиснута кнопка "Очистити", поля запиту та результату обнуляються.

Користувач також може перейти у свій кабінет, де побачить історію збережених результатів. Там передбачено дві основні дії: копіювати код у буфер обміну або видалити збережене. Після завершення роботи користувач натискає кнопку "Вийти", що завершує його сеанс.

Ця діаграма дозволяє побачити логіку роботи застосунку як безперервний потік дій: від авторизації до генерації, збереження та керування результатами.

Ця діаграма описує послідовність дій користувача у типовому сеансі:

1. Користувач заходить на сайт.
2. Якщо не авторизований — реєструється або входить.

Рис 3. Діаграма активностей

3. Переходить на головну сторінку генератора.
4. **Вводить запит → натискає кнопку генерації.**
5. Отримує HTML-код у полі відповіді.
6. За потреби — очищує поля або зберігає результат.
7. Переходить у кабінет.

8. Переглядає список збережених фрагментів.
9. Може скопіювати чи видалити фрагмент.
10. Завершує сесію — виходить з акаунту.

Кожна активність реалізована у вигляді окремого компонента або дії у фронтенді, пов’язаної з Firebase (запит, відповідь, запис, видалення)(див. рис.3)

Докладніше про етапи активності.

1. Відкриття сайту

Користувач відкриває вебсайт у браузері. На цьому етапі система перевіряє, чи автентифікований користувач. Якщо ні — система автоматично перенаправляє його на сторінку реєстрації або входу.

2. Реєстрація або вхід

Якщо користувач новий — він проходить реєстрацію, вводячи email і пароль. Ці дані зберігаються у Firebase Authentication.

Якщо користувач вже має обліковий запис — виконує вхід, проходячи перевірку даних.

У разі помилки автентифікації (неправильні дані) — система видає повідомлення про помилку.

3. Перехід до генератора

Після успішної автентифікації користувач потрапляє на головну сторінку генерації коду. Інтерфейс включає: поле для введення текстового запиту (Prompt); кнопку «Згенерувати»; поле для виводу HTML-коду.

4. Генерація HTML-коду

Користувач вводить текстовий запит (наприклад: "створи форму з полем для email та кнопкою").

Запит надсилається до AI-модуля.

AI формує відповідь у вигляді HTML-розмітки.

HTML-код виводиться у полі результату.

У разі технічної помилки — система відображає повідомлення з поясненням.

5. Подальші дії з результатом

Після отримання HTML-коду користувач має кілька варіантів:

- зберегти — результат записується у Firebase Firestore із прив’язкою до UID користувача;
- очистити — очищуються обидва поля: запиту та відповіді;

6. Перехід до кабінету

Користувач може натиснути кнопку «Кабінет» або відповідний елемент інтерфейсу, після чого потрапляє на сторінку перегляду збережених результатів.

У кабінеті реалізовано:

- виведення списку збережених HTML-кодів;
- копіювання будь-якого з них в буфер обміну;
- видалення вибраного результату з Firestore.

Дані завжди фільтруються за UID, тому користувач бачить тільки свої збереження.

7. Вихід із системи

При натисканні кнопки «Log Out»:

- закривається сесія у Firebase;
- користувач повертається на сторінку входу;
- доступ до кабінету та генератора блокується до повторної авторизації.

Особливості реалізації діаграми активностей

Дана діаграма дозволяє візуально структурувати:

- всі логічні гілки користувацького потоку (авторизований чи ні, помилка чи успішна генерація);
- циклічні дії — наприклад, генерація коду кілька разів поспіль без перезавантаження сторінки;
- умовні розгалуження — збереження чи видалення коду;
- кінцеву точку — вихід із системи.

Кожна активність пов’язана або з клієнтською логікою (JavaScript, DOM), або з хмарними службами Firebase (Auth, Firestore). Це забезпечує чітку

синхронізацію між інтерфейсом та backend'ом без необхідності в окремому сервері.

Діаграма активностей(див.рис.3) показує повний життєвий цикл взаємодії користувача із системою: від моменту входу до завершення сесії. Такий підхід дозволяє:

- спростити логіку розробки;
- виявити слабкі місця у взаємодії;
- покращити UX шляхом оптимізації кожного кроку.

Це також допомагає у тестуванні — усі гілки сценаріїв можуть бути перевірені на відповідність очікуваній поведінці.

2.5 Модель даних у Firestore (ER-діаграма)

Для зберігання даних у вебзастосунку використовується Firebase Firestore — хмарна документоорієнтована NoSQL база даних. Вона дозволяє створювати масштабовану, динамічну та безсерверну структуру збереження інформації без необхідності ручного адміністрування сервера чи складних SQL-запитів.

Firestore використовує модель колекцій і документів, де кожен документ — це гнучка структура, подібна до JSON, яка може містити вкладені об’єкти, мітки часу, текстові дані тощо.

Основні колекції у проєкті

У межах даного вебзастосунку реалізовано три ключові одиниці зберігання інформації:

1. Колекція users

Ця колекція зберігає інформацію про всіх зареєстрованих користувачів.

Типова структура документа**:**

```
{
  "uid": "U1aB7F9dPz...",         // Унікальний ідентифікатор користувача
  "email": "user@example.com"     // Email, з яким користувач зареєструвався
}
```

uid генерується автоматично Firebase Authentication після реєстрації і є

основним ключем доступу до персональних даних користувача.

2. Об’єкт session (автентифікаційна сесія)

Це не окрема колекція у Firestore, а сесія, що зберігається локально в браузері після автентифікації. Вона дозволяє отримати поточні дані про авторизованого користувача без звернення до бази.

Основна інформація, що зберігається у сесії:

```
{
  "userId": "U1aB7F9dPz..."   // UID з Firebase Auth
}
```

Сесія автоматично створюється Firebase і діє до моменту виходу користувача. На її основі система визначає:

- чи авторизований користувач;
- кому належать які збережені HTML-коди;
- які дії дозволено виконувати.

3. Колекція generatedHtml

Ця колекція є основною для зберігання згенерованого HTML-коду, який створюється за текстовим запитом користувача.

Типова структура документа:

```
{
  "uid": "U1aB7F9dPz...",                  // UID користувача, який створив код
  "htmlCode": "<div class='...'>...</div>", // Згенерований HTML-код
  "createdAt": "2025-05-26T14:22:00Z"       // Час збереження результату
}
```

Кожен запис асоціюється з користувачем через uid, що дозволяє зберігати, фільтрувати й виводити лише особисті результати конкретного користувача у його кабінеті.

Візуальне представлення ER-структури(див.рис.4)

Попри те, що Firestore — не реляційна БД, можна описати її логічну модель

за принципами ER-моделювання:

USERS (uid, email)

|---< creates >--- GENERATEDHTML (uid, htmlCode, createdAt)

SESSION (userId) → визначає поточного користувача

Таблиця 2.1

Призначення полів

Поле	Призначення
uid	Унікальний ідентифікатор користувача (пов'язує дані між колекціями)
email	Контактна інформація користувача, відображається в інтерфейсі
htmlCode	Результат генерації, який користувач отримує за текстовим запитом
createdAt	Дата та час створення фрагмента HTML
userId	Значення з session (локальне), що підтверджує авторизацію

Безпека та масштабованість

- Безпека: доступ до generatedHtml дозволено лише за UID, що відповідає поточному користувачеві;
- Масштабованість: система може обробляти тисячі генерацій, оскільки Firestore оптимізована для роботи з великими наборами документів;
- Ізоляція даних: кожен користувач бачить лише свої HTML-фрагменти, що гарантує приватність.

Модель зберігання даних у Firestore є простою, масштабованою і ідеально підходить для такого типу застосунків. Вона дозволяє зручно зберігати:

- облікові дані користувача;

- результати генерації HTML;
- забезпечує авторизацію та фільтрацію даних через UID.

Використання Firebase у якості бекенду дає змогу розробнику повністю зосередитись на інтерфейсі та функціоналі, а не на налаштуванні серверної інфраструктури чи захисті бази.

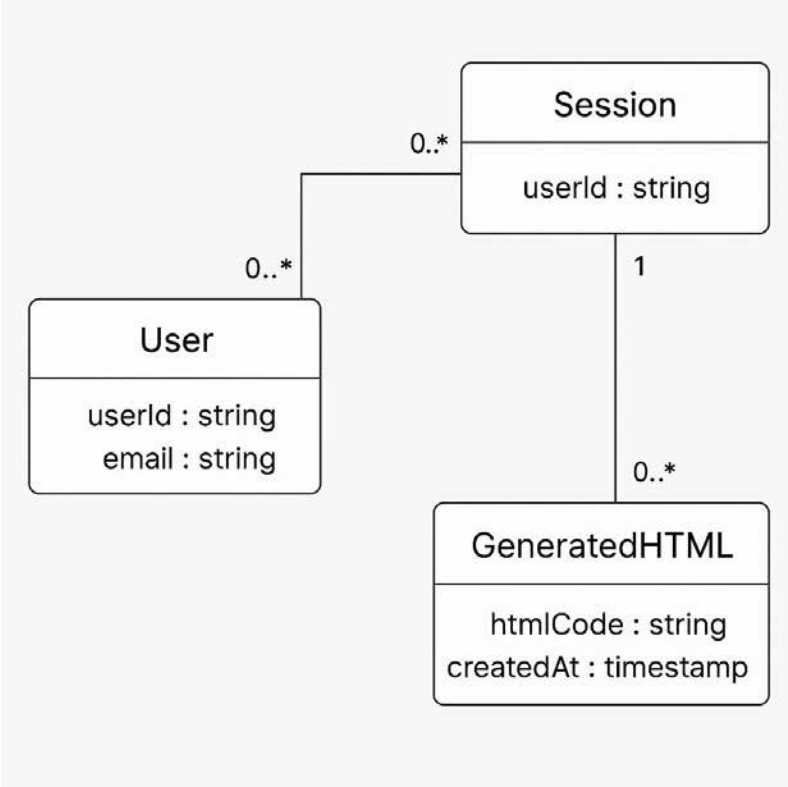

Рис 4. ER-діаграма

2.6 Висновки до розділу 2

У другому розділі дипломної роботи проведено глибоке моделювання предметної області, що дозволило чітко структурувати логіку та архітектуру майбутнього вебзастосунку. Завдяки використанню UML-діаграм вдалося сформувати повне уявлення про функціональну поведінку системи, структуру даних, об’єкти та їхню взаємодію.

Діаграма прецедентів показала всі ключові сценарії використання — від автентифікації до генерації коду та роботи з особистим кабінетом. Вона дозволила окреслити межі відповідальності користувача та системи. Діаграма класів продемонструвала внутрішню структуру системи, включаючи сутності

User, GeneratedHtml, Session та їхні атрибути і зв’язки. Це стало основою для формування моделі даних у Firebase Firestore.

Діаграма активностей візуалізувала типовий сеанс користувача: від входу до збереження результатів та завершення роботи. Вона дозволила наочно побачити весь потік дій у системі. А ER-діаграма описала логічну модель зберігання даних у Firestore, забезпечуючи розуміння структури бази з погляду доступу, ізоляції та безпеки.

Завдяки застосуванню сучасних підходів до моделювання, вдалося:

- систематизувати всі функціональні компоненти сайту;
- виявити й усунути дублювання або надлишковість логіки;
- створити зручну основу для подальшої реалізації архітектури;
- закласти підґрунтя для масштабування системи в майбутньому.

Таким чином, моделювання предметної області стало надійною відправною точкою для реалізації стабільного, безпечного та функціонального вебзастосунку, здатного генерувати HTML-код на основі запитів користувача та зберігати результати в персоналізованому середовищі.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура системи

Розроблена система є повноцінним вебдодатком із авторизацією, генерацією HTML-коду та кабінетом користувача, який працює в онлайні. Архітектура застосунку побудована за принципами розподіленої взаємодії між фронтендом, backend-сервісами та хмарним сховищем.

Основні блоки:

Фронтенд (HTML/CSS/JS) — взаємодія з користувачем, візуалізація інтерфейсу, збір запитів, відображення результатів, доступ до Firebase;

AI-модуль — модуль генерації HTML за запитом (використовується API зовнішнього ШІ, наприклад OpenAI);

Firebase Authentication — безпечна авторизація користувачів за email і паролем;

Firestore — хмарне сховище результатів, яке прив’язане до унікального ідентифікатора користувача (uid);

Хостинг (опціонально Firebase Hosting) — у майбутньому використовується для розміщення програми онлайн.

Типова архітектурна схема виглядає так:

КопіюватиРедагувати

```
Користувач → Фронтенд (браузер)
        → Firebase Auth (вхід)
        → AI API (запит/відповідь)
        → Firestore (читання/запис)
```

3.2 Інтерфейс користувача

Сайт містить кілька ключових сторінок, кожна з яких відповідає за окремий блок функціональності:

Сторінка реєстрації / входу

- Поля введення логіну (email) та паролю;
- Обробка авторизації через Firebase;
- Перенаправлення після входу на генератор.

Головна сторінка генератора

- Поле для введення текстового запиту;
- Кнопка “Згенерувати” — надсилає запит до AI;
- Блок із відповіддю — згенерований HTML;
- Кнопка “Зберегти” — запис у Firestore;
- Кнопка “Очистити” — очищення обох полів.

Кабінет користувача

- Виводиться список збережених HTML-фрагментів;
- Біля кожного фрагмента — кнопка “Видалити”;
- По кліку — фрагмент видаляється з бази;
- HTML-код можна просто виділити та скопіювати.

Вихід із системи

- Кнопка “Log out” завершує сесію та повертає на сторінку входу.

Усі сторінки мають **адаптивний дизайн**, реалізований через CSS Flex/Grid, що дозволяє працювати з сайтом як на ПК, так і на смартфонах.

3.3 Firebase як основа логіки авторизації та зберігання

Firebase Authentication:

Створення користувача (createUserWithEmailAndPassword);

Вхід користувача (signInWithEmailAndPassword);

Зберігання сесії (Token/UID);

Вихід (signOut()).

Firestore:

Колекція savedResults, де кожен документ зберігає:

uid: власник,

htmlCode: вміст,

timestamp: дата додавання.

При завантаженні сторінки “кабінет” відбувається фільтрація:

where("uid", "==", currentUser.uid)

3.4 Програмна реалізація функціоналу генератора

1. Введення запиту:

```
<textarea id="prompt"></textarea>
```

2. Генерація HTML:

```
fetch("https://api.openai.com/v1/chat/completions", {
  method: "POST",
  headers: {
    Authorization: "Bearer YOUR_API_KEY",
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    model: "gpt-3.5-turbo",
    messages: [{ role: "user", content: prompt }],
  }),
})
.then(res => res.json())
.then(data => {
  document.getElementById("result").innerText =
data.choices[0].message.content;
});
```

3. Збереження до Firestore:

```
import { getFirestore, collection, addDoc } from "firebase/firestore";

await addDoc(collection(db, "savedResults"), {
```

```
  uid: currentUser.uid,
  htmlCode: result,
  timestamp: new Date()
});
```

3.6 Алгоритм дій користувача

Алгоритм взаємодії користувача з вебзастосунком є послідовністю логічних дій, які користувач виконує під час роботи із системою. Ця логіка відображає типовий життєвий цикл одного сеансу: від входу в систему до генерації, перегляду або збереження результату, і завершення роботи.

Крок 1: Відкриття сайту

Користувач відкриває вебсторінку в браузері. Здійснюється перевірка, чи авторизований користувач:

- якщо ні, автоматично перенаправляється на сторінку входу;
- якщо так (сесія активна), переходить до головної сторінки генератора.

Цей етап реалізовано за допомогою Firebase Auth Listener, який відслідковує зміну статусу користувача (onAuthStateChanged).

Крок 2: Реєстрація або Вхід

На екрані з’являється форма з двома полями:

- Email;
- Пароль.

Користувач обирає дію:

- Реєстрація - створюється новий запис користувача у Firebase Auth;
- Вхід - перевіряється існуючий користувач, при успіху — сесія зберігається.

У разі успішної авторизації — переходить на сторінку генератора.

Крок 3: Введення текстового запиту

На головній сторінці генератора користувач бачить область для введення запиту.

Крок 4: Натискання кнопки “Згенерувати”

Після натискання:

- Здійснюється виклик функції fetch() з параметрами до API (наприклад, OpenAI);
- Система чекає на відповідь — HTML-код;
- Код відображається в окремому вікні результату.

Цей процес — центральний функціонал застосунку. Він реалізує основну ідею "AI на запит".

Крок 5: Подальші дії з результатом

Користувач має кілька варіантів:

- Очистити — поля очищуються (textarea.value = '', output.innerText = '');
- Зберегти — результат зберігається у Firestore:
  - uid: поточний користувач;
  - htmlCode: згенерований результат;
  - timestamp: дата й час.

Кнопка “Зберегти” активна лише, якщо є відповідь.

Крок 6: Перехід до Кабінету

У верхній частині сайту або окремою кнопкою можна перейти на сторінку “Мої збереження”. Там система:

- Виконує запит у Firestore;
- Отримує всі документи з колекції savedResults, де uid === currentUser.uid;
- Відображає список фрагментів із датою створення.

Крок 7: Дії у Кабінеті

Для кожного збереженого результату доступно:

- Копіювання — просте виділення тексту (без додаткової кнопки);
- Видалення — після підтвердження елемент видаляється з Firestore.

Інтерфейс автоматично оновлюється після видалення.

Крок 8: Завершення сеансу

На всіх сторінках присутня кнопка “Log out”. Вона:

- Викликає signOut() з Firebase;
- Завершує сесію користувача;
- Перенаправляє назад на сторінку входу.

Це дозволяє захистити персональні дані при використанні на публічних пристроях.

3.6 Вибір стеку технологій

Вибір стеку технологій — це один із найважливіших етапів при створенні будь-якого програмного продукту (сайту, мобільного застосунку, системи автоматизації тощо). Правильний вибір стеку безпосередньо впливає на: швидкість розробки, вартість проєкту, вродуктивність і масштабованість, везпеку, доступність розробників, підтримку в майбутньому (див. табл. 3.1)

Таблиця 3.1

Обґрунтування вибору стеку технологій

Компонент	**Технологія**	**Призначення**
UI	HTML + CSS	Основна структура і стилі
Логіка	JavaScript	Обробка взаємодії, виклики API
ШІ-модуль	OpenAI API	Генерація HTML по запиту користувача
Авторизація	Firebase Auth	Безпечна реєстрація/вхід
Зберігання результатів	Firebase Firestore	Зберігання згенерованого коду користувача
Хостинг	Firebase Hosting	Публікація сайту в інтернеті

Середовище	Веб-браузер	Кросплатформенний доступ до всіх функцій

3.7 Висновки до розділу 3

У цьому розділі було здійснено детальний розгляд процесу проєктування, архітектури та програмної реалізації вебзастосунку для генерації HTML-коду на основі текстових запитів. Було сформовано технічну структуру системи, яка охоплює всі ключові елементи — від інтерфейсу користувача до взаємодії з базою даних та штучним інтелектом.

Було обґрунтовано вибір архітектурного підходу з використанням фронтенду на HTML/CSS/JS, AI-модуля для генерації коду, Firebase Authentication для авторизації та Firestore як хмарного сховища збережених результатів. Архітектура побудована за принципом розподіленої логіки та повністю відповідає вимогам сучасних SPA-додатків.

Описано всі інтерфейсні компоненти системи: сторінки входу, генератора, кабінету користувача та виходу із системи. Детально розглянуто, як реалізована інтеграція з Firebase для зберігання HTML-коду, а також механізм авторизації й фільтрації даних за користувачем.

Алгоритм дій користувача було описано у вигляді блок-схеми, що дозволяє чітко зрозуміти всі кроки взаємодії з сайтом — від входу до генерації та керування збереженими результатами.

У результаті, на основі описаного функціоналу, створено логічно завершену, зручну та функціональну систему, що дозволяє автоматизувати процес створення HTML-розмітки без потреби у технічних знаннях з боку користувача.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Особливості розгортання вебзастосунку

У поточній реалізації вебсайт вже повністю функціональний, підтримує авторизацію, генерацію HTML та управління збереженими результатами. Він розгортається як односторінковий застосунок, що працює у браузері без потреби в локальному сервері. У дипломній роботі розглянуто застосунок як публічно доступний онлайн-сайт, на який можуть заходити користувачі з будь-якої точки світу.

4.1.1 Діаграма розгортання (Deployment Diagram)

Ця діаграма відображає, як компоненти взаємодіють у розподіленій системі:

Користувач (браузер):

- Відкриває сторінку
- Авторизується (Firebase Auth)
- Працює з інтерфейсом генератора

Фронтенд (HTML/JS/CSS):

- Виконується у браузері користувача
- Має доступ до API Firebase та AI

AI API (наприклад OpenAI):

- Отримує запит
- Віддає HTML-код

Firebase Auth:

- Зберігає облікові записи
- Керує сесіями

Firebase Firestore:

- Зберігає згенеровані коди (прив’язані до UID

→ `Діаграма буде представлена в наступному повідомленні візуально.`

4.2 Технічні вимоги

4.2.1 До клієнтського пристрою (браузер)

- **ОС**: Windows, macOS, Linux, Android, iOS
- **Браузер**: Chrome, Firefox, Edge, Safari
- **RAM**: 2 ГБ мінімум (4 ГБ рекомендовано)
- **Процесор**: Будь-який сучасний з 2+ ядрами
- **Інтернет**: мінімум 1 Мбіт/с

4.2.2 До сервера (хмарного середовища)

- **Firebase Hosting** (опціонально)
- **Firestore** як база даних
- **Firebase Auth** як менеджер користувачів
- **OpenAI API** або інший генератор

4.3 Інсталяція та запуск (логіка)

Інсталяційний процес для користувача не потрібен — він просто заходить на сайт через браузер. Але для розробника інструкція виглядає так:

1. Клонування репозиторію:

   git clone https://github.com/your-username/html-generator

   cd html-generator

2. Встановлення залежностей:

   npm install

3. Налаштування .env файлу:

   VITE_OPENAI_KEY=your_api_key

   VITE_FIREBASE_API_KEY=your_firebase_key

4. Запуск локально:

npm run dev

5. Для публічного сайту:

firebase deploy

4.4 Тестування функціональності

Усі ключові функції було протестовано вручну та частково автоматично (Jest).

Таблиця 4.1

Основні сценарії

Тестова дія	**Очікуваний результат**
Реєстрація користувача	Успішний запис у Firebase
Вхід з правильними даними	Перехід на генератор
Генерація HTML	Поява результату в полі відповіді
Збереження результату	Додавання до Firestore з UID
Відображення в кабінеті	Перелік усіх збережених результатів поточного користувача
Видалення результату	Видалення з Firestore
Log out	Вихід з сесії, повернення на логін

4.5 Оцінка інтерфейсу та якості програмного продукту

Авторизація (логін/реєстрація)

Першим екраном, який бачить користувач, є сторінка входу або реєстрації. Вона дозволяє зареєструватися за допомогою електронної пошти та пароля, або увійти, якщо обліковий запис уже створено. Форма має простий інтерфейс, поля логіну/пароля, кнопки для переходу між входом та реєстрацією.

Рис 5. Сторінка входу/реєстрації кристувача

Головна сторінка генерації HTML

Після авторизації користувач потрапляє на головну функціональну сторінку. Тут доступні:

Поле запиту — куди вводиться текст (наприклад: "створи форму з 2 полями вводу і кнопкою").

Кнопка "Згенерувати" — надсилає запит до AI.

Поле результату — нижче виводиться готовий HTML-код.

Кнопка "Зберегти" — додає результат до Firestore.

Кнопка "Очистити" — повністю очищує обидва поля.

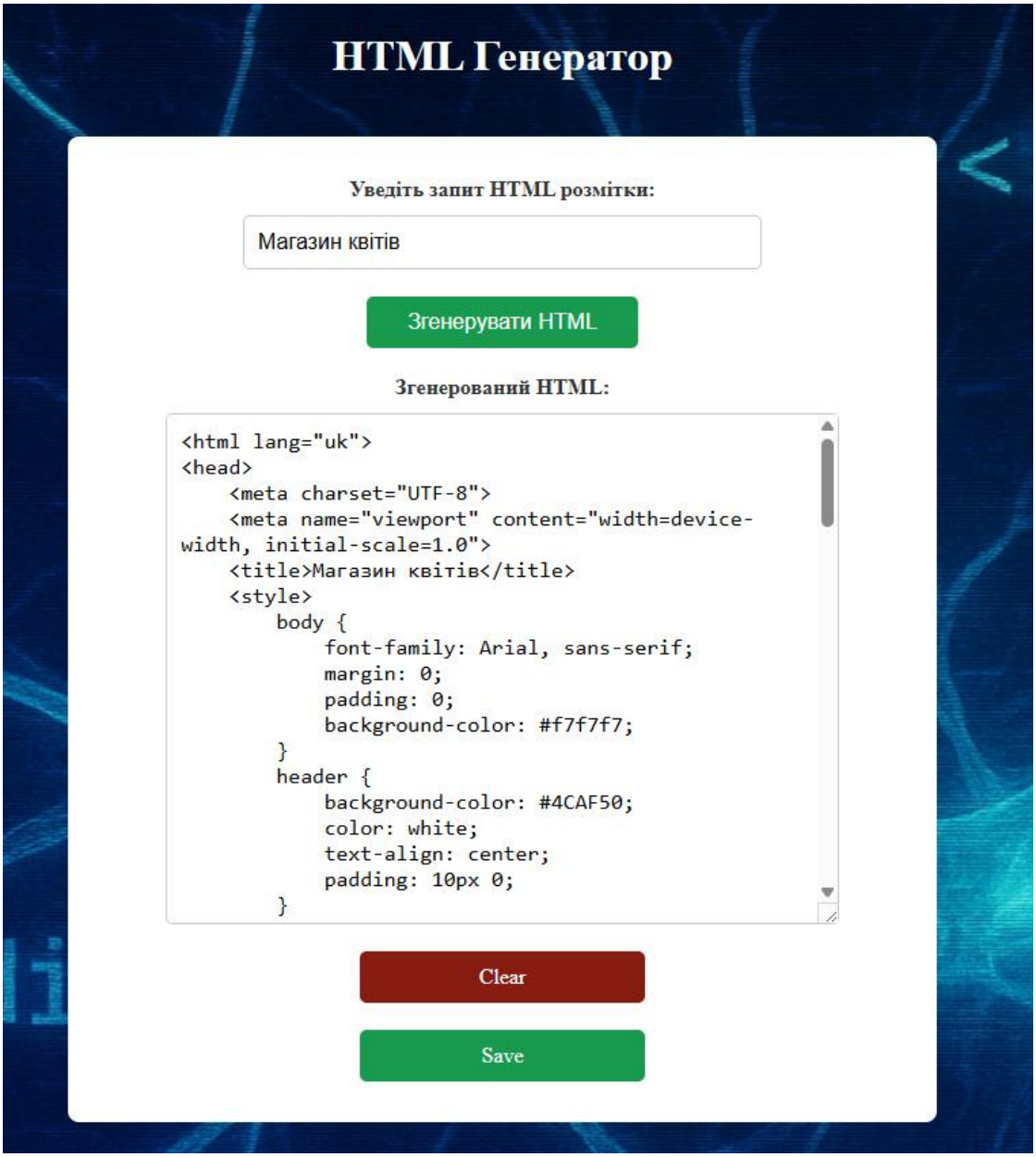

Рис 6. Сторінка генератора HTML-розмітки

Особистий кабінет (збережене)

У цьому розділі користувач може переглядати усі збережені раніше результати. Кожен HTML-код відображається у окремому блоці.

Доступні:

- Кнопка “Видалити” — очищує відповідний запис із бази;
- HTML можна виділити та скопіювати самостійно;
- Кнопки “Копіювати” або “Переглянути” не потрібні (все реалізовано максимально просто).

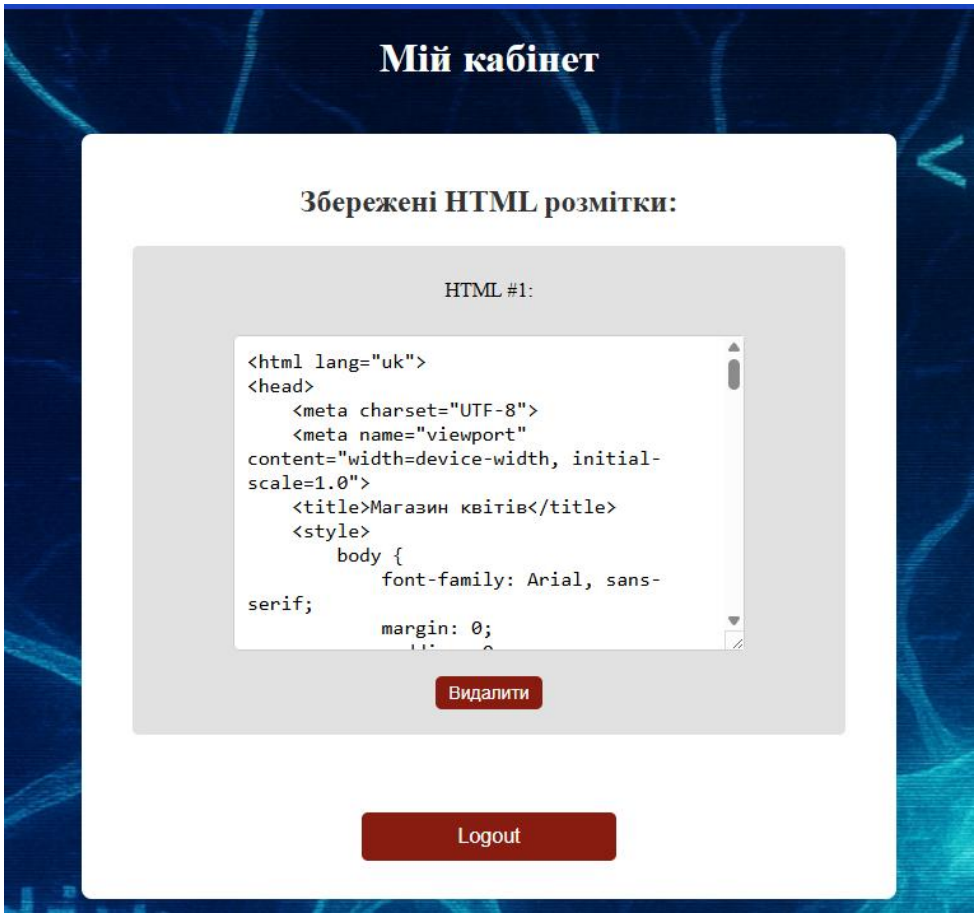

Рис 7. Сторінка особистого кабінету зі збереженими результатами

Кнопка виходу із системи

Кнопка “Log out” доступна з будь-якої сторінки — вона:

- очищує авторизаційну сесію;
- перекидає користувача назад на сторінку логіну.

4.6 Оцінка стабільності та надійності

Для перевірки якості та стабільності вебсайту було здійснено:

- тестування авторизації;тестування всіх дій із Firestore (запис, читання, видалення);
- ручне тестування UI на різних розмірах екранів;
- відправлення запитів до AI з різною довжиною та складністю;
- перевірка на помилки вводу (порожні поля, занадто довгі запити тощо).

4.7 Висновки до розділу 4

У четвертому розділі дипломної роботи було проведено аналіз практичної реалізації вебзастосунку, охарактеризовано процес його розгортання, експлуатаційні особливості, вимоги до інфраструктури та проведено оцінку його функціональності й стабільності.

Було визначено, що розроблена система є сучасним, повністю клієнтським вебрішенням, яке не потребує встановлення або локального серверного середовища. Усі дії виконуються безпосередньо у браузері користувача, а ключові функції (авторизація, збереження даних, виклик AI) реалізовані через хмарні сервіси Firebase та OpenAI API. Це значно зменшує вартість підтримки, підвищує надійність і дозволяє масштабувати систему без потреби у значних інфраструктурних змінах.

За допомогою діаграми розгортання було детально представлено логіку взаємодії між клієнтською частиною, зовнішніми сервісами (AI API), а також компонентами Firebase (Auth та Firestore). Така архітектура забезпечує простоту в експлуатації, надійний захист персональних даних та високу швидкодію.

Були описані технічні вимоги до пристроїв користувача — як клієнтської сторони, так і хмарної інфраструктури. Система виявилася повністю сумісною з основними ОС та браузерами, що свідчить про її універсальність. З боку розробника наведено покрокову інструкцію запуску та деплою, що дозволяє швидко перевірити, адаптувати чи розгорнути застосунок на інших майданчиках.

Особливу увагу було приділено тестуванню функціональності. Описані основні сценарії використання були ретельно перевірені як вручну, так і частково автоматизовано за допомогою Jest. Перевірка включала не тільки функціональну частину, а й UX/UI компоненти: зручність авторизації, коректність роботи збереження/видалення HTML-коду, відгук системи на запити до AI.

Під час оцінки інтерфейсу встановлено, що система є інтуїтивно зрозумілою, доступною для користувача без технічних знань. Простота у використанні — одна з ключових переваг, яка дає змогу використовувати

програму в широкому колі випадків: від навчання до швидкої генерації прототипів інтерфейсів.

Завдяки використанню Firebase Authentication збережено високий рівень безпеки. Авторизація, перевірка UID, ізольоване зберігання даних і захист сесій дозволяють запобігти несанкціонованому доступу до персональної інформації користувачів.

Проведена оцінка надійності підтвердила, що застосунок стабільно працює у різних сценаріях, витримує навантаження, не допускає критичних збоїв навіть при неправильних діях користувача (наприклад, при пустому запиті або втраті з’єднання з мережею).

Таким чином, можна зробити висновок, що вебзастосунок успішно впроваджено у вигляді повністю робочої системи, яка відповідає усім сучасним вимогам до доступності, надійності, простоти використання та масштабованості. Це доводить практичну доцільність його використання як інструменту для автоматизованої генерації HTML-розмітки за допомогою штучного інтелекту

ВИСНОВКИ

У цій дипломній роботі було розроблено повноцінний вебзастосунок під назвою «Програмне забезпечення для автоматизації створення HTML-розмітки інтерфейсів веб-сторінок за допомогою нейронних мереж». Система дозволяє користувачам в інтуїтивному середовищі отримувати згенерований HTML-код за допомогою штучного інтелекту, зберігати його та керувати результатами через особистий кабінет.

На основі поставленої мети було виконано такі завдання:

проведено системний аналіз предметної області:досліджено існуючі сервіси генерації коду;визначено проблематику та сформульовано вимоги до майбутньої системи;

проведено моделювання системи: побудовано діаграми прецедентів, класів, активностей, ER-діаграму; описано основні об'єкти, їх атрибути та функціональні залежності;

розроблено інформаційне та програмне забезпечення: реалізовано сайт із функціоналом генерації HTML через AI; організовано збереження результатів у Firestore; впроваджено безпечну авторизацію через Firebase Authentication; адаптовано інтерфейс для використання на різних пристроях;

проведено тестування та оцінку стабільності: перевірено усі ключові сценарії взаємодії користувача із системою; підтверджено надійність роботи як у десктопному середовищі.

У результаті реалізовано простий, інтуїтивний та ефективний інструмент, який може бути використаний:

- веброзробниками для швидкого створення шаблонів HTML;
- початківцями для навчання основ верстки;
- усіма охочими для генерації HTML-інтерфейсів без потреби у знаннях коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tyson, M. H. "Next.js and React: Building Modern Web Applications." Apress, 2022. – [Електронний ресурс] – Режим доступу: https://www.apress.com/gp/book/9781484278724

2. Smith, J. "Real-Time Web Applications with Socket.io." O'Reilly Media, 2021. – [Електронний ресурс] – Режим доступу: https://www.oreilly.com/library/view/real-time-web-applications/9781491950162/

3. Firebase Documentation. "Firebase for Web: Authentication and Firestore." Google, 2024. – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs

4. Mozilla Developers. "HTML Reference Guide." MDN Web Docs, 2024. – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML

5. Mozilla Developers. "CSS Layouts: Flexbox and Grid." MDN Web Docs, 2024. – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS

6. Mozilla Developers. "JavaScript Guide." MDN Web Docs, 2024. – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

7. Firebase Authentication Rules – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/rules

8. Firebase Firestore Security – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/firestore/security/get-started

9. Google Cloud. "Realtime Database vs. Firestore." Google Cloud Platform, 2023. – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/database/rtdb-vs-firestoreGitHub Pages Documentation – [Електронний ресурс] – Режим доступу: https://docs.github.com/en/pages

10. Jest Testing Framework Documentation – [Електронний ресурс] – Режим доступу: https://jestjs.io/docs

11. Prettier Documentation – [Електронний ресурс] – Режим доступу: https://prettier.io/docs/en/index.html

12. ESLint Documentation – [Електронний ресурс] – Режим доступу: https://eslint.org/docs/latest/

13. Google AI. "PaLM 2: Pathways Language Model." Google Research, 2023. – [Електронний ресурс] – Режим доступу: https://ai.google/discover/palm2

14. OpenAI API Documentation – [Електронний ресурс] – Режим доступу: https://platform.openai.com/docs

15. Best Practices for Front-End Security – [Електронний ресурс] – Режим доступу: https://owasp.org/www-project-top-ten/

16. Building Web Apps without a Backend – [Електронний ресурс] – Режим доступу: https://blog.logrocket.com/build-react-app-without-backend/

17. Проектування інтерфейсів HTML – [Електронний ресурс] – Режим доступу: https://studfile.net/preview/5802137/

18. Firebase Hosting: Deployment Guide – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/hosting

19. Вибір технологій для створення вебінтерфейсів – [Електронний ресурс] – Режим доступу: https://studopedia.org/14-30809.html