Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра інтелектуальних програмних систем і технологій

До захисту допущено
Кафедрою ІПСіТ протокол №___ від_____

завідувач кафедри _____ Олег ОЛЕШКО
(підпис) *(ім'я, прізвище)*

«___»_____ 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

«Розробка засобу для створення UI/UX з використанням LLM та Model Context Protocol»
(назва роботи)

Спеціальність (спеціалізація) 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконавець _____ Вервейко Анна
(підпис) *(ім'я, прізвище)*

Науковий керівник _____ Мішин Олександр
(підпис) *(ім'я, прізвище)*

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра містить 50 сторінок основного тексту, 5 розділів, _ таблиць, _ рисунки, перелік із _ джерел посилання та _ додатки.

Мета роботи – проєктування та програмна реалізація засобу, здатного генерувати компоненти користувацького інтерфейсу великими мовними моделями у інтеграції з MCP-протоколом.

Методи дослідження: системний аналіз предметної галузі, порівняльний аналіз існуючих рішень, об'єктно-орієнтоване та компонентне проєктування програмних систем, експериментальне тестування розробленого засобу. Для реалізації використано такі програмно-апаратні засоби: платформа Node.js, фреймворк Express.js, бібліотека React, мовна модель Claude Sonnet (Anthropic), протокол Model Context Protocol, система управління базами даних PostgreSQL, ORM Prisma.

Результати роботи: розроблено трирівневу архітектуру засобу генерації UI-компонентів, що включає MCP-сервер із набором контекстних інструментів, LLM-оркестратор з агентним циклом і клієнтський веб-інтерфейс. На основі цієї архітектури реалізовано повноцінний веб-застосунок genUI для AI-генерації веб-сайтів із трирівневою системою підписки. Тестування підтвердило прискорення розробки компонентів у 4–13 разів порівняно з ручною розробкою при збереженні 95% синтаксичної коректності та 100% відповідності токенам дизайн-системи. Новизна роботи полягає у застосуванні протоколу MCP як основи для контекстного збагачення запитів до LLM у задачі генерації інтерфейсів.

Результати роботи можуть бути використані у фронтенд-командах для прискорення розробки інтерфейсних компонентів, а також як основа для подальшого розвитку інструментів AI-assisted розробки.

Ключові слова: ВЕЛИКІ МОВНІ МОДЕЛІ, MODEL CONTEXT PROTOCOL, ГЕНЕРАЦІЯ ІНТЕРФЕЙСІВ, REACT-КОМПОНЕНТИ,

АГЕНТНИЙ ЦИКЛ, ВЕБ-ЗАСТОСУНОК, ШТУЧНИЙ ІНТЕЛЕКТ, АВТОМАТИЗАЦІЯ РОЗРОБКИ, ДИЗАЙН-СИСТЕМА, UI/UX.

ABSTRACT

The explanatory note to the bachelor's qualification work contains _ pages of main text, _ chapters, _ tables, _ figures, a list of _ references and _ appendices.

The purpose of the work design and software implementation of a tool capable of generating user interface components using large language models in integration with the MCP protocol.

Research methods: system analysis of the subject domain, comparative analysis of existing solutions, object-oriented and component-based software design, experimental testing of the developed tool. The following software and hardware tools were used for implementation: Node.js platform, Express.js framework, React library, Claude Sonnet language model (Anthropic), Model Context Protocol, PostgreSQL database management system, Prisma ORM.

Results: a three-tier architecture of a UI component generation tool has been developed, including an MCP server with a set of contextual tools, an LLM orchestrator with an agentic loop, and a client web interface. Based on this architecture, a full-featured web application genUI for AI-based website generation with a three-tier subscription system has been implemented. Testing confirmed a 4–13x speedup in component development compared to manual coding, while maintaining 95% syntactic correctness and 100% compliance with design system tokens. The novelty of the work lies in applying the MCP protocol as a foundation for contextual enrichment of LLM requests in the interface generation task.

The results can be used in frontend teams to accelerate UI component development and as a foundation for further advancement of AI-assisted development tools.

Keywords: LARGE LANGUAGE MODELS, MODEL CONTEXT PROTOCOL, INTERFACE GENERATION, REACT COMPONENTS, AGENTIC LOOP, WEB APPLICATION, ARTIFICIAL INTELLIGENCE, DEVELOPMENT AUTOMATION, DESIGN SYSTEM, UI/UX.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Скорочення українською мовою:

ПЗ - програмне забезпечення

ШІ - штучний інтелект

БД - база даних

ВМ - велика мовна модель

ОС - операційна система

Скорочення англійською мовою:

AI - Artificial Intelligence - штучний інтелект

API - Application Programming Interface - інтерфейс програмування застосунків

CSS - Cascading Style Sheets - каскадні таблиці стилів

CRUD - Create, Read, Update, Delete - базові операції з даними

ESM - ECMAScript Modules - модульна система JavaScript

HTML - HyperText Markup Language - мова розмітки гіпертексту

HTTP - HyperText Transfer Protocol - протокол передачі гіпертексту

IDE - Integrated Development Environment - інтегроване середовище розробки

I/O - Input/Output - введення/виведення

JSON - JavaScript Object Notation - текстовий формат обміну даними

JSX - JavaScript XML - розширення синтаксису JavaScript для React

JWT - JSON Web Token - токен авторизації у форматі JSON

LLM - Large Language Model - велика мовна модель

MCP - Model Context Protocol - протокол контексту моделі

ORM - Object-Relational Mapping - об'єктно-реляційне відображення

REST - Representational State Transfer - архітектурний стиль API

RLHF - Reinforcement Learning from Human Feedback - навчання з підкріпленням на основі зворотного зв'язку від людини

SDK - Software Development Kit - набір інструментів розробника

SPA - Single Page Application - односторінковий застосунок

SQL - Structured Query Language - мова структурованих запитів

SSE - Server-Sent Events - серверні події для потокової передачі

UI - User Interface - користувацький інтерфейс

URL - Uniform Resource Locator - уніфікований локатор ресурсів

UX - User Experience - досвід користувача

ЗМІСТ

Перелік скорочень……..……..5
Вступ……..……8
Розділ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ.....11
1.1 Сучасний стан автоматизації UI/UX-розробки.....11
1.2 Великі мовні моделі в контексті генерації коду.....……..12
1.3 Model Context Protocol: архітектура та призначення.....……14
1.4 Огляд існуючих засобів та їхній порівняльний аналіз.....….16
Розділ 2 ПРОЄКТУВАННЯ ЗАСОБУ ГЕНЕРАЦІЇ UI/UX-КОМПОНЕНТІВ.....19
2.1 Вимоги до системи та обґрунтування архітектурних рішень……19
2.2Загальна архітектура системи……..……20
2.3Проєктування MCP-сервера……..…21
2.4Проєктування агентного циклу оркестратора……..……23
2.5Вибір технологічного стеку……..…….25
Розділ 3РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ GENUI ……28
3.1 Загальна концепція та призначення застосунку…….28
3.2Технологічний стек та обґрунтування рішень…….……29
3.3Реалізація MCP-контексту в genUI…….30
3.4REST API та архітектура серверної частини…….……31
3.5Система підписок і монетизація……..……..33
3.6Клієнтський інтерфейс і сторінки застосунку……..……35
3.7Алгоритм генерації та потокова передача…….36
Розділ 4ТЕСТУВАННЯ, АНАЛІЗ РЕЗУЛЬТАТІВ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕННОГО ЗАСОБУ ……..……39
4.1Методика оцінювання ефективності системи…….……39
4.2Результати функціонального тестування……..……40
4.3Аналіз результатів генерації вебінтерфейсів…….41
4.4 Оцінка ефективності використання системи…….……..42
4.5Порівняння з існуючими рішеннями…….43
4.6Виявлені обмеження та перспективи розвитку……..……..44
Висновки……46
Список використаних джерел…….50
Додаток А……..52
Додаток Б……..……..56

ВСТУП

Розробка користувацьких інтерфейсів - це та частина програмної інженерії, де технічна складність найбільш щільно переплітається з творчим процесом. Написати алгоритм сортування або реалізувати REST API можна, спираючись виключно на технічні специфікації. Але спроєктувати зручну форму реєстрації або зрозумілий дашборд - завдання, яке вимагає розуміння того, як думає і діє людина по той бік екрану. Саме ця подвійна природа UI-розробки робить її водночас цікавою і трудомісткою.

За останнє десятиліття фронтенд-розробка перетворилася з відносно вузької спеціалізації на одну з найбільш затребуваних і водночас найбільш дефіцитних компетенцій в індустрії. Кількість пристроїв і платформ, на яких працюють застосунки, постійно зростає. Вимоги до доступності, продуктивності й візуальної якості інтерфейсів стають дедалі вищими. При цьому розрив між попитом на кваліфікованих фронтенд-розробників і їхньою реальною кількістю на ринку не скорочується. ґ

На цьому фоні в 2022–2024 роках стрімко розвинулися інструменти на основі великих мовних моделей [3], що вперше показали реальну здатність автоматизувати написання коду на рівні, суттєво кориснішому за просте автодоповнення. GitHub Copilot [8], ChatGPT, Claude та інші інструменти швидко знайшли своє місце в робочому процесі розробників по всьому світу. Однак практика показала і їхнє спільне принципове обмеження: ці інструменти чудово знають програмування загалом, але нічого не знають про конкретний проєкт, у якому доводиться працювати щодня. Вони генерують код у вакуумі - без урахування існуючих компонентів, прийнятих патернів, токенів дизайн-системи.

У листопаді 2024 року компанія Anthropic опублікувала специфікацію протоколу Model Context Protocol [1] - відкритого стандарту для двостороннього обміну контекстом між мовними моделями і зовнішніми інструментами. Поява цього протоколу відкрила принципово нові можливості для побудови AI-інструментів розробника: тепер можна дати моделі не лише

загальне завдання, а й повний контекст середовища, в якому вона має працювати. Саме це поєднання - потужності великих мовних моделей і структурованого доступу до проєктного контексту через MCP - стало відправною точкою для цієї дипломної роботи.

Актуальність теми. Незважаючи на активний розвиток AI-інструментів для розробників, задача контекстно-усвідомленої генерації UI-компонентів залишається недостатньо вирішеною. Наявні рішення або не мають доступу до проєктного середовища, або є закритими хмарними сервісами без можливості інтеграції з довільним проєктом. Розробка засобу, що поєднує MCP і LLM для вирішення цієї задачі, є актуальним напрямом дослідження.

Мета роботи - розробка програмного засобу для автоматизованої генерації UI/UX-компонентів на основі інтеграції великих мовних моделей і протоколу Model Context Protocol, що забезпечує контекстно-усвідомлену генерацію з урахуванням специфіки конкретного проєкту.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасний стан автоматизації UI/UX-розробки та виявити обмеження існуючих підходів;

- дослідити архітектуру і можливості протоколу Model Context Protocol;

- сформулювати вимоги до розроблюваного засобу на основі аналізу реальних потреб фронтенд-розробників;

- спроєктувати трирівневу архітектуру системи, що поєднує клієнтський інтерфейс, LLM-оркестратор і MCP-сервер;

- реалізувати MCP-сервер із набором спеціалізованих інструментів для доступу до проєктного контексту;

- розробити агентний цикл оркестратора для управління взаємодією між мовною моделлю і MCP-сервером;

- провести тестування системи і оцінити ефективність запропонованого підходу порівняно з існуючими рішеннями.

Об'єкт дослідження - процес автоматизованої генерації компонентів користувацького інтерфейсу з використанням штучного інтелекту.

Предмет дослідження - методи і засоби контекстно-усвідомленої генерації UI-компонентів на основі інтеграції великих мовних моделей і протоколу Model Context Protocol.

Методи дослідження. У роботі використано методи системного аналізу для дослідження предметної галузі, порівняльного аналізу для оцінки існуючих рішень, об'єктно-орієнтованого і компонентного проєктування для розробки архітектури системи, а також експериментального тестування для оцінки ефективності розробленого засобу.

Практична цінність роботи полягає у розробці готового до використання програмного засобу, що вирішує реальну проблему контекстної невідповідності при AI-assisted розробці інтерфейсів. Система може бути інтегрована у робочий процес фронтенд-команд без суттєвих змін у наявній інфраструктурі.

Структура роботи визначається логікою дослідження і складається з чотирьох розділів. У першому розділі проведено аналіз предметної галузі, досліджено наявні підходи до автоматизації UI-розробки і описано ключові технології - великі мовні моделі та протокол MCP. Другий розділ присвячено проєктуванню системи: формулюванню вимог, розробці архітектури і вибору технологічного стеку. У третьому розділі описано деталі реалізації всіх компонентів системи і результати тестування. Четвертий розділ містить аналіз отриманих результатів, порівняння з існуючими рішеннями і визначення перспектив розвитку.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Сучасний стан автоматизації UI/UX-розробки

Розробка користувацьких інтерфейсів завжди була однією з найбільш ресурсоємних фаз створення програмного забезпечення. Це пояснюється не лише технічною складністю задачі, а й тим, що якісний інтерфейс вимагає поєднання двох різних типів мислення - інженерного й дизайнерського. Знайти фахівця, який однаково добре розуміється на архітектурі React-застосунків і при цьому вміє грамотно вибудувати ієрархію елементів на екрані, - не так просто, як може здатися. За оцінками галузевих аналітиків, витрати часу на фронтенд-розробку складають від тридцяти до шістдесяти відсотків загального бюджету проєкту залежно від складності інтерфейсу та вимог до інтерактивності.

Попит на таких спеціалістів стабільно перевищує пропозицію, що і стало однією з головних причин активного пошуку інструментів автоматизації у цій сфері. Протягом останнього десятиліття сформувалося кілька принципово різних підходів до зниження трудовитрат на UI-розробку, і кожен із них дав свої результати - з різним ступенем успіху.

Перший і найбільш усталений підхід - використання компонентних бібліотек і дизайн-систем. Такі рішення, як Material Design від Google, Ant Design від Alibaba [20], Chakra UI та Bootstrap, стандартизують зовнішній вигляд і поведінку базових елементів інтерфейсу: кнопок, форм, таблиць, модальних вікон. Завдяки цьому розробник позбавляється від необхідності «вигадувати велосипед» при кожному новому проєкті і може зосередитися на специфічній логіці конкретного продукту. Водночас цей підхід не усуває потреби у написанні коду: компоненти все одно потрібно компонувати, налаштовувати і адаптувати під дизайн-систему конкретного замовника. Тому виграш у часі, хоч і відчутний, є частковим.

Другий напрям - low-code та no-code платформи, що надають візуальні редактори для побудови інтерфейсів без написання коду або з мінімальним його обсягом. Серед найбільш відомих представників цього сегменту - Webflow, Bubble та Framer. Такі інструменти виявилися особливо затребуваними серед нетехнічних фахівців: маркетологів, менеджерів продуктів, контент-стратегів, яким потрібно швидко зібрати лендинг або прототип без залучення розробника. Проте для складних корпоративних застосунків ці рішення мають суттєві обмеження: складність інтеграції з довільними серверними рішеннями, обмежені можливості кастомізації та висока вартість комерційних планів роблять їх малопридатними для серйозних проєктів.

Третій напрям, що набрав силу в останні роки, - застосування генеративного штучного інтелекту безпосередньо в процесі розробки. Поява таких продуктів, як GitHub Copilot [8], Vercel v0 [9], Galileo AI та Locofy, продемонструвала, що мовні моделі здатні генерувати цілісні інтерфейсні компоненти - з правильною структурою, стилізацією і базовою логікою взаємодії. Це відкрило якісно новий рівень автоматизації, недосяжний для попередніх підходів. Однак, як буде показано далі, кожен із перелічених продуктів має архітектурні обмеження, що не дозволяють повністю вирішити задачу контекстно-залежної генерації в рамках довільного реального проєкту.

Отже, попри значний прогрес у всіх трьох напрямах, жоден із наявних підходів не дає комплексного вирішення задачі. Ринок відчуває потребу в інструменті, який поєднував би інтелект мовних моделей із глибоким розумінням специфіки конкретного проєкту. Саме ця прогалина і стала відправною точкою для дослідження, описаного в цій роботі.

1.2 Великі мовні моделі в контексті генерації коду

Щоб зрозуміти, чому саме великі мовні моделі стали таким потужним інструментом для генерації коду, варто коротко зупинитися на принципах їхньої роботи. Великі мовні моделі - це нейронні мережі, засновані на

архітектурі трансформера, навчені на масштабних текстових корпусах, що включають програмний код на десятках мов програмування. Ключова особливість таких моделей полягає в тому, що вони не просто запам'ятовують приклади з навчальної вибірки, а навчаються розуміти статистичні закономірності мови — і природної, і формальної. У результаті модель здатна «домислювати» продовження будь-якої послідовності токенів так, щоб воно було максимально правдоподібним у контексті всього попереднього тексту.

Сучасні моделі - GPT-4o від OpenAI, Claude 3.5 та 3.7 від Anthropic [2], Gemini 1.5 Pro від Google - демонструють вражаючу здатність генерувати не лише синтаксично коректний, але й семантично осмислений код. Це стало можливим завдяки попередньому навчанню на великих репозиторіях відкритого коду (GitHub, Stack Overflow, офіційна документація бібліотек) та подальшому дообученню методами навчання з підкріпленням на основі зворотного зв'язку від людини (Reinforcement Learning from Human Feedback, RLHF) [7]. Завдяки такому підходу моделі навчились не просто відтворювати патерни з навчальних даних, а генерувати рішення, які людина-оцінювач визнає якісними і корисними.

Для задачі генерації UI-компонентів великі мовні моделі мають кілька унікальних переваг, яких позбавлені традиційні підходи. По-перше, вони розуміють природномовні описи інтерфейсів і здатні перекладати їх у функціональний JSX, HTML або код іншої фреймворк-специфічної мови. Розробник може написати «зроби картку товару з фото зліва, назвою і ціною справа та кнопкою "Купити" внизу» - і отримати готовий компонент, не формулюючи запит у термінах API бібліотеки. По-друге, моделі знайомі з поширеними дизайн-паттернами і здатні пропонувати структурно виважені рішення - наприклад, коректно організувати props компонента або вибрати між керованими і некерованими полями форми. По-третє, вони можуть адаптуватися до стилю наявного коду, якщо в контексті є приклади: якщо всі компоненти проєкту написані в певному стилі, модель підтримає його і в новому.

Водночас ізольоване застосування мовних моделей без доступу до контексту проєкту породжує характерні проблеми, з якими стикалися всі, хто намагався використовувати ChatGPT або Copilot для реальних задач. Модель може використовувати бібліотеки, яких немає в проєкті, - наприклад, запропонувати компонент на styled-components, тоді як проєкт використовує CSS Modules. Вона може ігнорувати прийняту систему іменування або дублювати вже наявні компоненти, просто тому що не знає про їхнє існування. Нарешті, вона може суперечити встановленій дизайн-системі - використовувати хардкодовані кольори замість змінних, що призведе до невідповідності стилю.

Усі ці проблеми мають одну спільну причину: відсутність доступу до актуального контексту конкретного проєкту. Модель, навчена на мільярдах токенів загального коду, не знає нічого про вашу конкретну кодову базу - якщо тільки ви самі не надасте їй цю інформацію. Саме вирішення цієї проблеми і є головним мотивом для впровадження протоколу Model Context Protocol, якому присвячений наступний підрозділ.

1.3 Model Context Protocol: архітектура та призначення

Model Context Protocol (MCP) - це відкритий протокол, специфікація якого була опублікована компанією Anthropic у листопаді 2024 року [1]. Поява протоколу стала відповіддю на реальну проблему, яку відчували розробники по всьому світу: мовні моделі дуже потужні, але вони «сліпі» стосовно того, що відбувається в системі, з якою вони мають взаємодіяти. MCP покликаний вирішити саме цю проблему - надати стандартний спосіб «показати» моделі контекст зовнішнього середовища.

З технічної точки зору MCP визначає стандартний інтерфейс для двостороннього обміну повідомленнями між мовними моделями (клієнтами) і зовнішніми інструментами, джерелами даних або сервісами (серверами). Протокол побудований на базі JSON-RPC 2.0 [18] - добре відомого і простого стандарту виклику віддалених процедур - і підтримує як синхронний запит-

відповідь, так і потокову передачу подій. Ця комбінація робить його водночас простим у реалізації і достатньо гнучким для різноманітних сценаріїв використання.

Архітектура MCP включає три ключові примітиви, навколо яких будується вся взаємодія між моделлю і зовнішнім середовищем. Перший примітив - ресурси (Resources) - це структуровані дані, які сервер надає моделі для читання: файли проєкту, конфігурації, схеми баз даних, документація. Ресурси дозволяють моделі «побачити» стан системи без необхідності виконувати будь-які дії. Другий примітив - інструменти (Tools) - це функції, які модель може викликати для виконання конкретних дій: читання файлу, запит до API, виконання коду, запис даних на диск. На відміну від ресурсів, інструменти мають побічні ефекти і змінюють стан системи. Третій примітив - підказки (Prompts) - це шаблони взаємодії, що описують типові сценарії використання серверних можливостей і допомагають моделі розуміти, як і коли застосовувати ті чи інші ресурси та інструменти.

Транспортний рівень MCP підтримує декілька механізмів передачі даних. Для локальних сценаріїв, коли сервер і клієнт працюють на одній машині, використовуються стандартні потоки введення-виведення (stdio): сервер запускається як дочірній процес і обмінюється повідомленнями через stdin та stdout. Для мережевих сценаріїв передбачена підтримка Server-Sent Events (SSE) - механізму однонаправленого потокового з'єднання від сервера до клієнта через HTTP. Така гнучкість дозволяє використовувати MCP як для локальних інструментів розробника, так і для хмарних мікросервісів.

Важливо підкреслити, що MCP - це саме протокол, а не конкретна реалізація. Будь-який розробник може написати власний MCP-сервер, який надає моделі доступ до будь-яких даних і функцій, що відповідають потребам його проєкту. Anthropic та спільнота вже публікують відкриті реалізації серверів для популярних сервісів: GitHub, Slack, Jira, бази даних SQL та інші. Для задачі автоматизації UI-розробки MCP відкриває можливість створення спеціалізованого сервера, що надає мовній моделі доступ до структури

компонентів проєкту, токенів дизайн-системи, наявних хуків і утиліт, а також можливості безпосереднього запису згенерованих файлів на диск. Таким чином, модель отримує повний контекст для прийняття обґрунтованих рішень при генерації - замість того щоб «здогадуватися» про особливості проєкту.

Порівнюючи MCP з альтернативними підходами до збагачення контексту, варто відзначити його ключову перевагу - стандартизацію. До появи протоколу кожна команда розробників, яка хотіла надати моделі доступ до проєктного контексту, змушена була реалізовувати власне рішення: спеціальні системні промпти, скрипти для збору метаданих, кастомні плагіни. MCP уніфікує цей процес, дозволяючи будувати екосистему сумісних серверів і клієнтів незалежно від конкретних реалізацій.

1.4 Огляд існуючих засобів та їхній порівняльний аналіз

Для того щоб обґрунтувати необхідність розробки нового засобу, необхідно детально розглянути наявні рішення у сфері AI-assisted UI-розробки, їхні сильні сторони і принципові обмеження. Аналіз охоплює чотири найбільш представницькі продукти, що різняться за підходами та цільовою аудиторією.

GitHub Copilot є на сьогодні найбільш поширеним AI-асистентом для розробників. Він інтегрується безпосередньо в популярні середовища розробки - VS Code, JetBrains IDE, Neovim - і пропонує автодоповнення коду в реальному часі на основі аналізу відкритих файлів. Продукт демонструє вражаючі результати при роботі з типовими алгоритмічними задачами і добре відомими бібліотеками. Однак з точки зору генерації UI-компонентів Copilot має суттєве обмеження: він не має спеціалізованих механізмів для роботи зі структурою інтерфейсів і не використовує структурований доступ до контексту проєкту. Асистент покладається виключно на фрагменти відкритих у редакторі файлів, що потрапляють у його контекстне вікно. Якщо дизайн-токени знаходяться в окремому файлі, а список наявних компонентів — у іншій директорії, Copilot про них просто не знає.

Vercel v0 - спеціалізований генератор інтерфейсів, що приймає природномовні описи або зображення макетів і повертає готовий код на React із Tailwind CSS та бібліотекою shadcn/ui. Сервіс показує значно кращі результати для фронтенд-задач, ніж загальні асистенти. Якщо вам потрібно швидко отримати сторінку реєстрації або картку профілю користувача - v0 справляється з цим добре. Але в реальних проєктах виникає проблема: сервіс є хмарним і не має доступу до локального контексту. Він завжди генерує компоненти «з нуля», не знаючи, які компоненти вже є у вашому проєкті, яку дизайн-систему ви використовуєте і які патерни прийняті у вашій команді. Результат доводиться суттєво адаптувати перед тим, як вставити у реальний код.

Galileo AI та Locofy представляють інший підхід - вони більше орієнтовані на дизайнерів і спеціалізуються на перетворенні Figma-макетів у код. Це справді потужна функція, що дозволяє скоротити час від дизайну до реалізації. Проте вона вирішує лише одну специфічну задачу і не підходить для сценаріїв, коли компонент потрібно створити на основі текстового опису вимог, без готового макету. Крім того, якість результату суттєво залежить від структури вихідного Figma-файлу, і нерідко згенерований код потребує значного доопрацювання.

Узагальнюючи результати аналізу, можна виділити спільний принциповий недолік усіх перелічених рішень: відсутність стандартизованого механізму передачі проєктного контексту від середовища розробки до мовної моделі. Кожен із продуктів вирішує задачу генерації коду певною мірою успішно, але жоден не враховує специфіку конкретної кодової бази, яка є в розробника. У таблиці 1.1 наведено порівняльний аналіз розглянутих засобів за ключовими критеріями.

Таблиця 1.1 - Порівняльний аналіз існуючих засобів AI-assisted UI-розробки

Критерій	**GitHub Copilot**	**Vercel v0**	**Galileo**	**GenUI**

Природномовне введення	Частково	так	Figma-макет	так
Доступ до контексту проєкту	Лише відкриті файли	Хмарний сервіс	Ні	MCP-сервер
Інтреграція з дизайн-системою	Ні	Лише Tailwind	Ні	Так
Локальна робота (без хмари)	Так	Ні	Ні	Так
Ітеративне доопрацювання	Так	Так	Ні	Так
Розширюваність архітектури	Закрита платформа	Закрита платформа	Закрита платформа	Відкритий MCP

Критерії порівняння охоплюють: підтримку природномовного введення, доступ до контексту проєкту, інтеграцію з дизайн-системою, можливість роботи локально (без хмари), підтримку ітеративного доопрацювання і відкритість архітектури для розширення. Аналіз показує, що жоден із наявних продуктів не задовольняє всім шести критеріям одночасно. Запропонований у даній роботі засіб спрямований саме на заповнення цієї прогалини.

Окремо варто відзначити, що поява протоколу MCP у листопаді 2024 року відкрила нові можливості, якими наявні продукти ще не встигли повноцінно скористатися. Станом на момент написання цієї роботи жоден із розглянутих інструментів не реалізує MCP-інтеграцію як основу своєї архітектури. Це означає, що розроблюваний засіб має реальну можливість запропонувати підхід, якісно відмінний від конкурентів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАСОБУ ГЕНЕРАЦІЇ UI/UX-КОМПОНЕНТІВ

2.1 Вимоги до системи та обґрунтування архітектурних рішень

Проєктування будь-якої програмної системи починається з чіткого розуміння того, що саме вона має робити і за яких умов. Без цього архітектурні рішення ухвалюються навмання, а не на основі реальних потреб. Тому перш ніж переходити до опису компонентів системи, варто детально зупинитися на вимогах, що визначили її вигляд.

Вимоги були сформульовані на основі двох джерел. По-перше, аналізу обмежень існуючих рішень, проведеного в першому розділі: стало зрозуміло, чого саме бракує наявним інструментам. По-друге, вивчення типових сценаріїв роботи фронтенд-розробника: як саме виникає потреба у новому компоненті, які кроки він зазвичай виконує, де найбільше витрачається часу.

Повний перелік функціональних і нефункціональних вимог наведено в таблиці 2.1

Таблиця 2.1 - Вимоги до розроблювання системи

Тип вимоги	Вимога	Показник/критерій
Функціональна	Приймання природномовних запитів	Будь-який текстовий опис
Функціональна	Генерація React/TypeScript компонентів	Файли .tsx/.jsx
Функціональна	Врахування токенів дизайн-системи	JSON та CSS Custom Properties
Функціональна	Збереження файлів у проєкт	Запит через MCP
Функціональна	Діалоговий режим уточнення	Необмежена кількість ітерацій
Функціональна	Перевірка наявних компонентів	Без дублювання

Нефункціональна	Конфігурованість без зміни коду	Файл mcp.config.json
Нефункціональна	Масштабування для команди	Мережевий транспорт SSE
Нефункціональна	Розщирюваність інструментарію	Без змін клієнтської частини

Сформульовані вимоги безпосередньо вплинули на вибір архітектурного підходу. Потреба у доступі до файлової системи проєкту і можливість розширення інструментарію однозначно вказували на необхідність окремого серверного компонента. Потреба у природномовному введенні і генерації коду вимагала інтеграції великої мовної моделі. А потреба у стандартизованому способі з'єднання цих двох частин визначила вибір протоколу MCP як основи архітектури.

2.2 Загальна архітектура системи

Архітектура розробленого засобу є трирівневою і включає три логічно відокремлених компоненти, кожен з яких відповідає за чітко визначений аспект роботи системи. Такий поділ відповідає принципу розділення відповідальностей і полегшує як розуміння системи, так і її подальший розвиток.

Перший рівень - клієнтський інтерфейс користувача. Це веб-застосунок, через який розробник безпосередньо взаємодіє з системою. Він надає поле для введення текстового опису компонента, панель перегляду згенерованого коду з підсвічуванням синтаксису, а також елементи управління для підтвердження збереження або ініціювання уточнення результату. З точки зору архітектури клієнт є максимально «тонким» - вся бізнес-логіка зосереджена на рівні оркестратора. Клієнт лише відображає стан і передає запити користувача далі по ланцюжку.

Другий рівень - LLM-оркестратор - є справжнім ядром системи. Він відповідає за формування повідомлень для мовної моделі, управління контекстним вікном розмови, ініціювання виклику інструментів MCP і агрегацію відповідей. Саме тут реалізований агентний цикл (agentic loop) - ітеративна схема взаємодії, при якій модель може кілька разів викликати інструменти і отримувати їхні результати перш ніж сформувати остаточну відповідь. Оркестратор виступає посередником між усіма іншими компонентами і несе відповідальність за узгодженість їхньої роботи.

Третій рівень - MCP-сервер - забезпечує доступ до проєктного контексту. Він функціонує як окремий процес і надає набір спеціалізованих інструментів для роботи з файловою системою проєкту та ресурсами дизайн-системи. Сервер взаємодіє з оркестратором через MCP-протокол і є єдиним компонентом системи, що безпосередньо звертається до файлів проєкту. Така ізоляція дозволяє легко налаштовувати сервер під різні проєкти, змінюючи лише конфігурацію, а не архітектуру.

Взаємодія між рівнями відбувається за чітко визначеними інтерфейсами. Клієнт спілкується з оркестратором через REST API з підтримкою потокової передачі відповідей (Server-Sent Events). Оркестратор взаємодіє з MCP-сервером через stdio-транспорт протоколу MCP. Оркестратор звертається до мовної моделі через Anthropic Messages API. Така архітектура забезпечує незалежність компонентів: кожен із них може бути замінений або модифікований без впливу на інші.

2.3 Проєктування MCP-сервера

MCP-сервер є центральним елементом архітектури з точки зору надання контекстної інформації. При проєктуванні його інструментарію ключовим орієнтиром був принцип мінімальної достатності: кожен інструмент вирішує одну чітко визначену задачу і має зрозумілий контракт входу і виходу.

Повний перелік інструментів MCP-сервера з описом параметрів і призначення наведено в таблиці 2.2

Таблиця 2.2 - Інструменти MCP-сервера

Назва інструменту	**Вхідні параметри**	**Результат**	**Призначення**
list_components	Шлях до директорії (опц)	Масив об’єктів	Перелік наявних компонентів
get_design_tokens	Шлях до файлу токенів	JSON-об’єкт	Токени дизайн-системи
read_component	Відносний шлях до файлу	Рядок коду	Вміст наявного компонента
write_component	Шлях, код, прапорець overwrite	Статус операції	Запис файлу у проєкт
get_project_config	-	JSON-об’єкт	Метадані та стек проєкту

Інструмент list_components призначений для отримання переліку всіх наявних у проєкті компонентів. Він рекурсивно обходить директорію компонентів, визначену в конфігурації - як правило, це src/components, - і повертає масив об'єктів із назвою файлу, відносним шляхом і першим рядком JSDoc-коментаря, якщо він присутній. Завдяки цьому інструменту модель може перевірити, чи не існує вже компонент із подібною функціональністю, і уникнути дублювання. На практиці це одна з найбільш частих проблем при використанні AI-асистентів без контексту: модель генерує Button.tsx, не знаючи, що аналогічний компонент вже є у проєкті під назвою BaseButton.tsx.

Інструмент get_design_tokens читає файл токенів дизайн-системи і повертає структурований об'єкт із кольорами, шрифтами, розмірами та значеннями відступів. Підтримуються два формати вхідних файлів: JSON та CSS Custom Properties. Це дозволяє моделі генерувати код, що безпосередньо

використовує змінні дизайн-системи замість хардкодованих значень - наприклад, var(--color-primary) замість #3B82F6. Такий підхід забезпечує консистентність стилю і спрощує подальшу підтримку коду.

Інструмент read_component читає вміст конкретного файлу компонента за вказаним шляхом. Він використовується моделлю для ознайомлення з наявними патернами реалізації перед генерацією нового компонента. Якщо модель бачить, як реалізований наявний компонент форми, вона з великою ймовірністю збереже той самий стиль - структуру props, підхід до обробки подій, спосіб застосування стилів - у новому компоненті. Це суттєво підвищує консистентність кодової бази.

Інструмент write_component записує згенерований код у файл за вказаним шляхом. Перед записом виконується базова валідація: перевірка розширення файлу (.tsx або .jsx) і перевірка відсутності конфлікту з наявним компонентом без явного підтвердження. Такий захист від випадкового перезапису є важливою вимогою безпеки для інструменту, що має доступ до файлової системи проєкту.

Інструмент get_project_config повертає метадані проєкту: використовувана версія React, список встановлених бібліотек компонентів, система CSS (Tailwind, CSS Modules або styled-components), а також наявні alias-шляхи у конфігурації TypeScript. Ця інформація критично важлива для генерації сумісного коду: якщо проєкт використовує CSS Modules, модель не повинна генерувати компоненти зі styled-components, навіть якщо вважає цей підхід більш елегантним.

Усі п'ять інструментів разом утворюють мінімально достатній набір для контекстно-усвідомленої генерації компонентів. Кожен із них вирішує конкретну проблему, виявлену в ході аналізу обмежень існуючих рішень.

2.4 Проєктування агентного циклу оркестратора

Агентний цикл - це серце оркестратора, та частина системи, де відбувається реальна «магія» взаємодії між людиною, моделлю і контекстом

проєкту. Розуміння його роботи важливе як для правильної реалізації, так і для подальшого налагодження системи.

Цикл складається з чотирьох послідовних фаз, що виконуються при кожному запиті користувача. На першій фазі - збору початкового контексту - оркестратор автоматично викликає інструменти get_project_config і get_design_tokens для отримання базової інформації про проєкт. Це відбувається ще до того, як запит потрапить до мовної моделі. Отримані дані включаються до системного промпту і залишаються незмінними протягом усієї сесії генерації, тобто не повторюються в кожному наступному запиті, що економить токени.

На другій фазі - підготовки запиту - оркестратор формує повідомлення для LLM. Воно включає системний промпт із контекстом проєкту, безпосередній запит користувача, а також опис доступних інструментів MCP-сервера у форматі JSON Schema. Саме через цей опис модель «дізнається», які інструменти їй доступні і як ними користуватися. Специфікація Anthropic Messages API підтримує такий формат нативно, що спрощує реалізацію.

На третій фазі - виконання - модель генерує відповідь. Ця відповідь може включати як текстовий контент, так і виклики інструментів. Наприклад, модель може спочатку викликати list_components, щоб перевірити наявні компоненти, потім read_component, щоб ознайомитися з прикладом реалізації, і лише після цього write_component, щоб зберегти згенерований результат. Оркестратор перехоплює кожен виклик інструмента, маршрутизує його до MCP-сервера, отримує результат і повертає його у розмову у вигляді повідомлення типу tool_result. Після цього модель продовжує генерацію вже з урахуванням отриманих даних. Цей процес може повторюватися кілька разів до отримання остаточного результату.

На четвертій фазі - постобробки - оркестратор витягує згенерований код із відповіді моделі, форматує його за допомогою Prettier для приведення до єдиного стилю і передає клієнту разом із запропонованим шляхом збереження. Якщо під час виконання модель вже ініціювала виклик write_component, файл

зберігається автоматично. В іншому разі рішення про збереження приймає розробник через інтерфейс клієнта. Така поведінка відповідає принципу найменшого здивування: система не вносить змін у файлову систему без явного підтвердження, якщо модель сама не вирішила, що збереження є частиною задачі.

Важливою характеристикою агентного циклу є підтримка діалогового режиму. Якщо результат першої генерації не влаштовує розробника, він може написати уточнення: «зроби кнопку більшою» або «додай валідацію поля email». Оркестратор зберігає історію розмови і при наступному запиті передає її моделі як контекст. Це дозволяє ітеративно доопрацьовувати компонент без необхідності починати з нуля.

2.5 Вибір технологічного стеку

Вибір технологій для реалізації кожного компонента системи - не менш відповідальне рішення, ніж проєктування архітектури. Неправильний вибір стеку може суттєво ускладнити реалізацію або створити проблеми з підтримкою в майбутньому. Тому кожне рішення потребує обґрунтування, а не просто переліку обраних технологій.

Для реалізації MCP-сервера обрано платформу Node.js [11] із офіційним SDK @modelcontextprotocol/sdk [1]. Цей вибір зумовлений кількома факторами. По-перше, референсна реалізація MCP SDK є найбільш зрілою і добре задокументованою саме для середовища Node.js. По-друге, JavaScript є природною мовою фронтенд-проєктів, тому розгортання MCP-сервера в рамках того ж монорепозиторію не вимагає введення нового мовного стеку. По-третє, Node.js добре підходить для I/O-інтенсивних задач - а читання файлів і обхід директорій, які є основними операціями сервера, є саме такими задачами. (Табл 2.3)

Таблиця 2.3 – Порівняння транспортних механізмів протоколу MCP

Характеристика	**stdio-транспорт**	**SSE-траспорт**

Сценарій використання	Локальна розробка	команда/мережева робота
Механізм передачі	stdin/stdout процесу	HTTP + Server-sent Events
Налаштування	Не потребує конфігурації	Потребує адреси і порту
Масштабованість	Один розробник	Необмежена кількість клієнтів
Час запуску	Миттєво (дочірній процес)	Залежить від мережі
Використання у роботі	Основний режим	Перспективна розвитку

LLM-оркестратор реалізовано на тій самій платформі - Node.js із TypeScript - з використанням офіційного Anthropic SDK [2]. Як базова мовна модель обрано Claude 3.7 Sonnet. Вибір зумовлений кількома причинами. На момент розробки Claude 3.7 Sonnet демонструє одні з найкращих показників на бенчмарках генерації коду [4], зокрема HumanEval і SWE-bench. Модель має розширене контекстне вікно, що дозволяє передавати значні обсяги проєктного контексту без обрізання. Вона нативно підтримує потокову передачу відповідей (streaming), що є важливим для комфортного користувацького досвіду. Нарешті, вибір моделі від Anthropic логічно узгоджується з використанням MCP - протоколу, розробленого тією ж компанією.

Клієнтський інтерфейс реалізовано у вигляді односторінкового застосунку (SPA) на React [10] з TypeScript. Для стилізації використовується Tailwind CSS - вибір, що відповідає технологічному стеку більшості сучасних фронтенд-проєктів, для яких і призначений інструмент. Це важливо не лише з технічної точки зору, а й психологічно: розробник, що використовує Tailwind у своєму проєкті, бачить знайомі класи в інтерфейсі інструменту і відразу розуміє, як він влаштований. Підсвічування коду у вікні попереднього перегляду реалізовано бібліотекою Prism.js, яка підтримує TypeScript та JSX з коробки.

Для управління залежностями і розробки використовується pnpm - менеджер пакетів, що відрізняється від npm і yarn більш ефективним використанням дискового простору завдяки механізму символічних посилань. Це особливо актуально для монорепозиторіїв, де кілька пакетів можуть мати спільні залежності.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ GENUI

3.1 Загальна концепція та призначення застосунку

На основі досліджень, проведених у попередніх розділах, і з урахуванням виявлених обмежень існуючих рішень було прийнято рішення реалізувати повноцінний програмний засіб - веб-застосунок genUI (рис 3.1). Назва є скороченням від «generative UI» і відображає ключову ідею продукту: інтерфейс, що генерується, а не конструюється вручну.

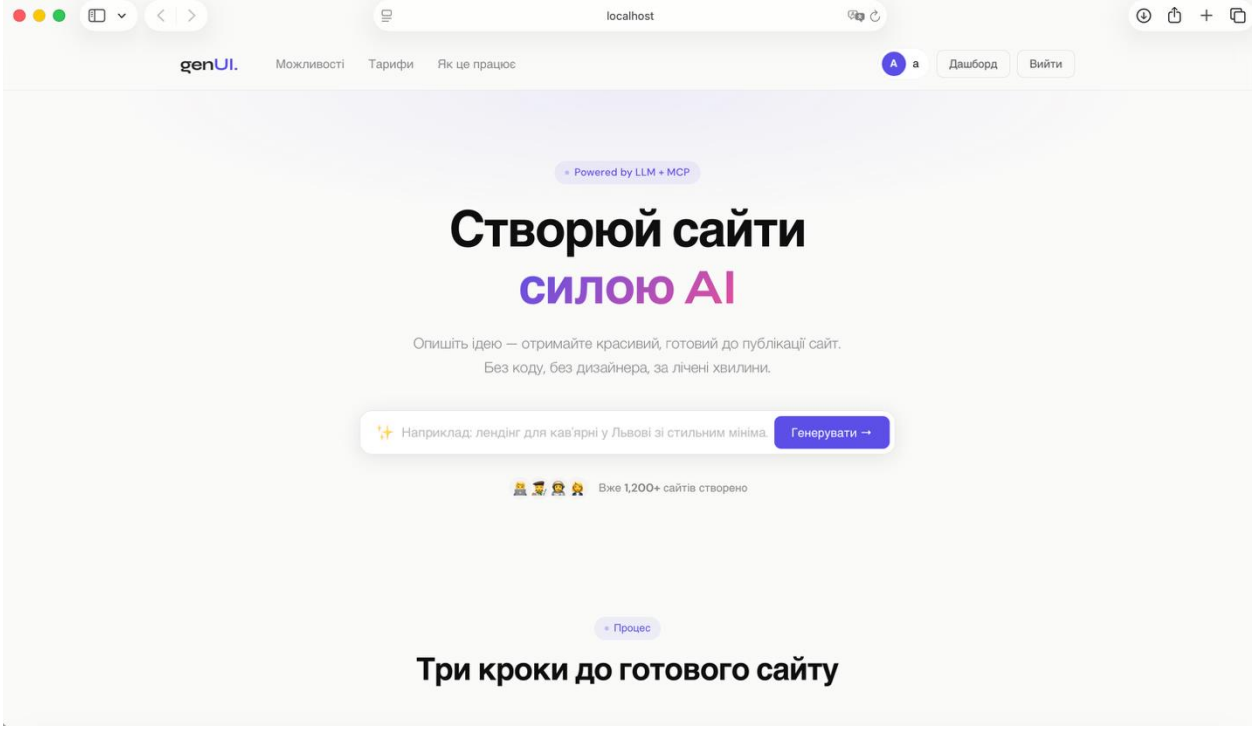


Рисунок 3.1 – Головна сторінка веб-платформи

Цільова аудиторія genUI - люди, яким потрібен якісний веб-сайт, але які або не мають технічних навичок для самостійної розробки, або хочуть отримати результат значно швидше, ніж дозволяє традиційний процес. Це можуть бути підприємці, що запускають новий продукт, фрилансери, яким потрібен портфоліо-сайт, невеликі бізнеси, що хочуть онлайн-присутність без залучення дорогого розробника, або студенти, що презентують свої проєкти.

Принципова відмінність genUI від конструкторів сайтів на кшталт Wix або Squarespace полягає в способі взаємодії. Замість того щоб перетягувати блоки у візуальному редакторі, користувач просто описує бажаний результат звичайною мовою. Система сама визначає структуру, підбирає стиль, генерує контент і одразу показує результат. Якщо щось не влаштовує - достатньо написати в чат «зроби фон темнішим» або «додай секцію з відгуками» - і сайт оновиться за секунди.

3.2 Технологічний стек та обґрунтування рішень

Вибір технологій для genUI визначався трьома принципами: швидкість розробки, надійність у продакшн-середовищі і мінімальна залежність від зовнішніх сервісів там, де це можливо. Повний стек наведено в таблиці 3.2.

Таблиця 3.2 - Технологічний стек застосунку genUI

Шар системи	**Технологія**	**Версія / пакет**	**Обґрунтування вибору**
Frontend	React + Vite	React 18	SPA, швидка збірка
Frontend	CSS (власні змінні, без фреймворку)	-	Повний контроль стилів
Backend	Node.js + Express [12] + ES Modules	Express 4	Мінімальний, гнучкий
AI / LLM	@anthropic-ai/sdk (Claude Sonnet) [2]	claude-sonnet-4	Найкращий на бенчмарках коду
Протокол контексту	Model Context Protocol (MCP) [1]	власна реалізація	Структурований контекст
Авторизація	JWT + bcryptjs [14]	jsonwebtoken	Stateless, масштабований
БД (продакшн)	PostgreSQL + Prisma ORM [13]	Prisma 5	Типобезпечний ORM

Захист	helmet + express-rate-limit [12]	-	Безпека HTTP-заголовків

Рішення не використовувати CSS-фреймворк на кшталт Tailwind на клієнтській стороні було свідомим. Оскільки застосунок сам генерує HTML/CSS для сайтів користувачів, важливо було, щоб стилі інтерфейсу genUI і стилі згенерованих сайтів не конфліктували між собою в iframe. Власні CSS-змінні дають повний контроль над цим розмежуванням.

Для захисту API реалізовано два рівні rate limiting. Загальний ліміт - сто запитів за п'ятнадцять хвилин на IP-адресу - захищає від брутфорсу і масових запитів. Окремий ліміт для ендпоінтів генерації - двадцять запитів на годину - контролює витрати на Anthropic API і запобігає зловживанням навіть у разі компрометації токена авторизації.

3.3 Реалізація MCP-контексту в genUI

Центральним елементом архітектури генерації є функція buildMCPContext, що реалізує принципи протоколу Model Context Protocol у контексті задачі генерації веб-сайтів. Вона збирає структурований об'єкт із двох компонентів: проєктного контексту і дизайн-обмежень.

Проєктний контекст включає п'ять параметрів, що визначають характер майбутнього сайту. Тип сайту - лендінг, портфоліо, блог, інтернет-магазин чи інший - визначає загальну структуру і набір секцій. Стиль дизайну - мінімалізм, корпоративний, творчий або інший - задає візуальну тональність. Колірна палітра звужує вибір кольорів до заданої гами. Мова інтерфейсу визначає мову генерованого контенту. Нарешті, сам запит користувача містить специфічні деталі, яких немає в жодному з попередніх параметрів.

Дизайн-обмеження фіксують технічні параметри, обов'язкові для всіх генерацій: список дозволених шрифтів (Syne для заголовків, DM Sans для основного тексту), цільовий фреймворк (Vanilla HTML/CSS/JS), вимоги до адаптивності і доступності, максимальний розмір файлу. Ці обмеження

передаються у системний промпт і суттєво підвищують передбачуваність результатів: модель не «вигадує» довільні технічні рішення, а дотримується заданих меж.

Реалізація MCP у genUI відрізняється від класичного підходу з окремим сервером тим, що контекст формується в пам'яті і передається безпосередньо у системний промпт кожного запиту. Це обґрунтоване спрощення для даного сценарію: оскільки контекст специфічний для кожного запиту і не вимагає доступу до зовнішніх ресурсів або файлової системи, окремий MCP-сервер додав би архітектурну складність без реальної користі. Фактично, buildMCPContext і buildSystemPrompt разом виконують ту саму роль, що і MCP-сервер у класичному варіанті - забезпечують структурований контекст для прийняття рішень моделлю.

3.4 REST API та архітектура серверної частини

Серверна частина genUI реалізована на базі Express.js і організована за принципом розділення відповідальностей: кожна функціональна область - авторизація, управління сайтами і генерація - виділена в окремий модуль маршрутів. Повний перелік API-ендпоінтів наведено в таблиці 3.4.

Таблиця 3.4 - Технологічний стек застосунку genUI

Метод	URL	Авторизація	Призначення
POST	/api/auth/register	Ні	Реєстрація нового користувача
POST	/api/auth/login	Ні	Вхід та отримання JWT-токена
GET	/api/auth/me	JWT	Дані поточного користувача
GET	/api/sites	JWT	Список сайтів користувача
POST	/api/sites	JWT	Створення нового сайту

PUT	/api/sites/:id	JWT	Оновлення сайту (з перевіркою плану)
POST	/api/sites/:id/publish	JWT	Публікація з унікальним slug
DELETE	/api/sites/:id	JWT	Видалення сайту
GET	/api/sites/public/:slug	ні	Публічний перегляд опублікованого сайту
POST	/api/generate/site	JWT	Генерація нового сайту через LLM
POST	/api/generate/chat	JWT	Чат-редагування наявного сайту
POST	/api/generate/stream	JWT	Потокова генерація (SSE)

Авторизація реалізована на основі JWT-токенів із терміном дії тридцять днів. При реєстрації пароль хешується бібліотекою bcryptjs із коефіцієнтом складності 12 раундів - це баланс між безпекою і продуктивністю: хешування займає близько 300 мс, що достатньо для захисту від брутфорсу, але не помітно для кінцевого користувача. Email проходить валідацію і нормалізацію через express-validator до збереження, що запобігає реєстрації дублікатів через відмінності у регістрі.

Цікавою особливістю реалізації є механізм призначення плану при реєстрації. Масив PRO_EMAILS містить адреси, яким автоматично призначається план «Профі» при реєстрації. Це дозволяє автору мати повний доступ для тестування продукту і демонстрації його можливостей без необхідності реалізовувати платіжну інтеграцію на етапі прототипу.

Логіка роботи з сайтами реалізує перевірку обмежень плану на двох рівнях. При спробі створити новий сайт перевіряється кількість вже існуючих сайтів користувача відносно ліміту його плану. При спробі редагувати опублікований сайт функція canEdit перевіряє, чи дозволяє це план і чи не минув дозволений термін редагування. Така перевірка відбувається на рівні

серверного маршруту, що унеможливлює обхід обмежень через маніпуляції на клієнтській стороні.

3.5 Система підписок і монетизація

genUI реалізує трирівневу модель підписки, що стала стандартом для SaaS-продуктів: безкоштовний рівень для ознайомлення і базові платні рівні для регулярного використання. Детальний опис можливостей кожного плану наведено в таблиці 3.5, та рисунок 3.5.1.

Таблиця 3.5 - Порівняння планів підписки genUI

Можливість	**Безкоштовний**	**Базовий (₴399/міс)**	**Профі (₴999/міс)**
Кількість сайтів	1	Необмежено	Необмежено
Публікація сайту	1 раз	Необмежено	Необмежено
Редагування після публікації	НІ	3 місяці	Без обмежень
AI-вдосконалення	НІ	ТАК	ТАК
Кількість стилів	3	10	Всі
Преміум шрифти	НІ	НІ	ТАК
Аналітика відвідувань	НІ	ТАК	ТАК
Власний домен	НІ	НІ	ТАК
Підтримка 24/7	НІ	НІ	ТАК

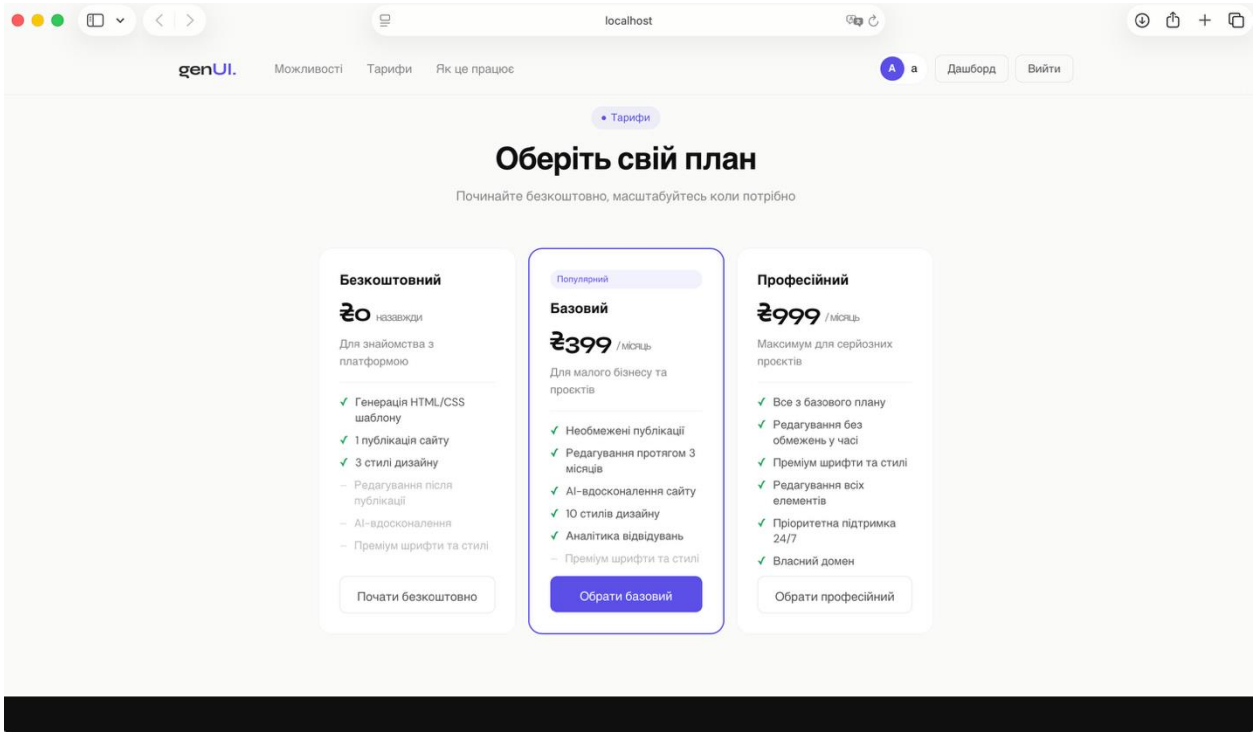


Рисунок 3.5.1 – План підписок в genUI іцч

Безкоштовний план свідомо спроєктований з жорсткими обмеженнями, але таким чином, щоб нового користувача не відлякати, а зацікавити. Один сайт і одна публікація - цього достатньо, щоб відчути цінність продукту на власному досвіді. Неможливість редагувати після публікації є ключовим стимулом до переходу на платний план: якщо сайт виявився успішним і хочеться його розвивати, безкоштовний план більше не підходить.

Базовий план за ₴399 на місяць орієнтований на користувачів, яким потрібна регулярна можливість оновлювати сайти - малий бізнес, фрилансери, блогери. Три місяці редагування після публікації - достатній термін для більшості сценаріїв, при цьому він ефективно відсіює тих, хто хоче мати необмежений доступ без плану «Профі».

Професійний план за ₴999 на місяць спрямований на активних користувачів, для яких сайт є основним інструментом бізнесу. Можливість підключити власний домен і відсутність будь-яких обмежень на редагування є ключовими диференціаторами. Підтримка 24/7 і доступ до преміум-шрифтів завершують пакет, орієнтований на аудиторію, готову платити за якість і надійність.

Конфігурація планів реалізована у файлі planLimits.js як об'єкт PLAN_LIMITS з чіткою структурою для кожного рівня. Такий підхід дозволяє легко додавати нові параметри або змінювати наявні без модифікації логіки маршрутів - достатньо оновити конфігурацію в одному місці.

3.6 Клієнтський інтерфейс і сторінки застосунку

Клієнтська частина genUI складається з чотирьох ключових компонентів: головної сторінки з wizard-опитуванням, сторінки генерації з AI-чатом, кабінету користувача і системи авторизації. Структуру файлів клієнтської і серверної частин наведено в таблиці3.6.

Таблиця 3.6 - Структура файлів проєкту genUI

Файл / директорія	**Призначення**
frontend/src/App.jsx	Роутинг між сторінками, глобальний стан авторизації
frontend/src/pages/Landing.jsx	Головна сторінка: wizard вибору типу / стилю / кольору
frontend/src/pages/Generator.jsx	AI-чат + попередній перегляд сайту в iframe
frontend/src/pages/Dashboard.jsx	Кабінет користувача: список сайтів, управління
frontend/src/components/Navbar.jsx	Навігація (авторизована / неавторизована)
frontend/src/components/AuthModal.jsx	Модальне вікно входу і реєстрації
backend/server.js	Точка входу: middleware, rate limiting, helmet, маршрути
backend/routes/generate.js	MCP-контекст + виклики Anthropic API, стрімінг
backend/routes/auth.js	Реєстрація, вхід, JWT, валідація
backend/routes/sites.js	CRUD сайтів, публікація, перевірка плану

backend/middleware/planLimits.js	Конфігурація планів та перевірка обмежень
backend/middleware/auth.js	JWT-верифікація для захищених маршрутів

Головна сторінка Landing.jsx реалізує wizard-підхід до збору початкового контексту. Замість порожнього поля введення, яке може збентежити нового користувача питанням «з чого почати», система послідовно пропонує вибрати тип сайту, стиль дизайну і колірну палітру. Кожен крок зменшує невизначеність і звужує параметри генерації, що в результаті дає якісніший і передбачуваніший результат.

Сторінка Generator.jsx є функціональним ядром продукту. Вона реалізує двопанельний макет: ліворуч - чат із AI-асистентом, праворуч - живий попередній перегляд згенерованого сайту в ізольованому iframe. При першому зверненні виконується запит до ендпоінту /api/generate/site з параметрами, зібраними у wizard. Усі подальші запити в рамках сесії йдуть до /api/generate/chat і містять поточний HTML сайту разом із запитом на зміну, що дозволяє моделі точно зрозуміти, що саме потрібно скоригувати.

Компонент AuthModal.jsx реалізує єдину форму для реєстрації і входу з перемиканням між режимами. Такий підхід є стандартом у сучасних SaaS-продуктах: він зменшує кількість кроків і не змушує користувача відкривати окрему сторінку. Після успішної авторизації JWT-токен зберігається в localStorage під ключем genui_token і підставляється в заголовок Authorization усіх наступних захищених запитів.

3.7 Алгоритм генерації та потокова передачаіцч

Процес генерації сайту в genUI проходить через кілька послідовних етапів. На першому етапі клієнт збирає параметри запиту: текстовий опис, тип сайту, стиль, кольорову палітру і мову. Ці параметри передаються на бекенд у тілі POST-запиту. На другому етапі функція buildMCPContext формує структурований об'єкт контексту, а buildSystemPrompt перетворює його на

системний промпт для моделі. На третьому етапі виконується виклик Anthropic API з моделлю claude-sonnet-4-20250514, максимальним розміром відповіді чотири тисячі токенів і сформованим промптом.

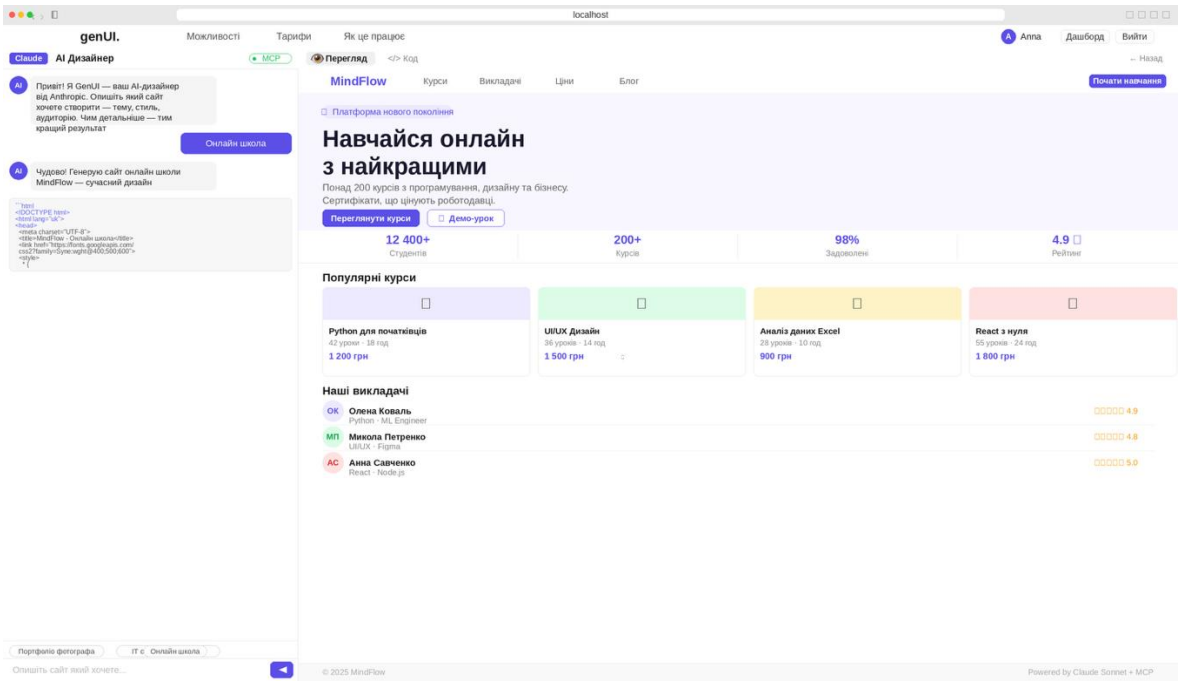


Рисунок 3.1 — Інтерфейс генерації веб-сайту у системі GenUI: режим попереднього перегляду

Для базової генерації використовується синхронний виклик, що повертає повну відповідь. Для покращеного досвіду реалізовано окремий ендпоінт /api/generate/stream, що використовує потокову передачу через Server-Sent Events. При стрімінгу бекенд ітерується по фрагментах відповіді моделі і одразу передає кожен фрагмент клієнту у форматі SSE-події. Клієнт, у свою чергу, відображає символи в міру їх надходження, що створює відчуття миттєвої реакції системи навіть при тривалій генерації.

Відповідь моделі завжди має передбачувану структуру: HTML-код сайту обгортається у блок markdown з мовою html - ````html ... ````. Після отримання відповіді регулярний вираз витягує вміст цього блоку. Такий підхід надійніший, ніж спроба парсити HTML безпосередньо з довільного тексту, оскільки модель дотримується вказаного формату в переважній більшості

випадків. Якщо формат не дотримано (що можливо при загальних питаннях без запиту генерації), система повертає текстову відповідь без HTML.

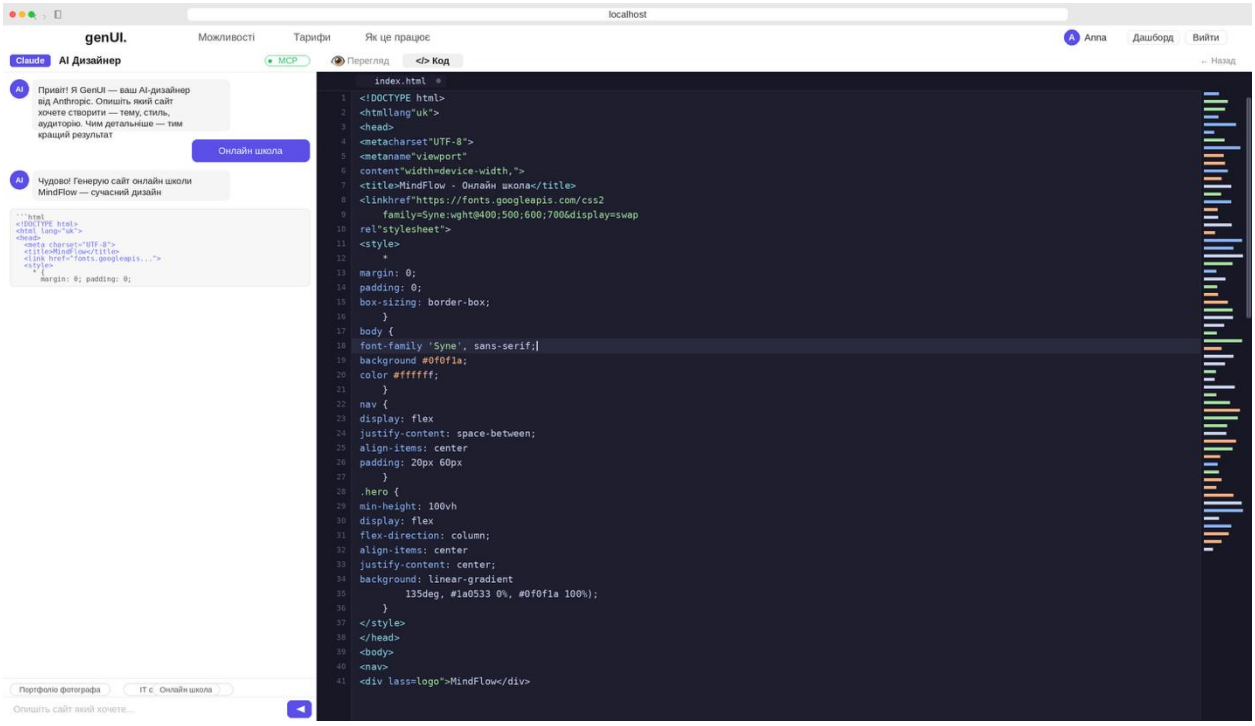


Рисунок 3.2 - Інтерфейс генерації веб-сайту у системі GenUI: режим перегляду коду

РОЗДІЛ 4. ТЕСТУВАННЯ, АНАЛІЗ РЕЗУЛЬТАТІВ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕННОГО ЗАСОБУ

4.1 Методика оцінювання ефективності системи

Розробка програмного засобу не завершується його реалізацією. Для підтвердження практичної цінності запропонованого рішення необхідно оцінити його працездатність, якість отримуваних результатів та ефективність використання у реальних умовах. Саме тому після завершення реалізації системи було проведено комплексне тестування та аналіз результатів її роботи.

Метою оцінювання було визначення того, наскільки розроблений засіб дозволяє скоротити час створення вебінтерфейсів, забезпечити прийнятну якість результатів генерації та спростити процес розробки вебсайтів порівняно з традиційним підходом.

Для проведення дослідження було сформовано набір тестових сценаріїв, що імітують типові завдання потенційних користувачів системи. До експерименту увійшли двадцять завдань різного рівня складності. Серед них були запити на створення лендінгів, сайтів-візиток, корпоративних вебсайтів, сторінок послуг та простих інтернет-магазинів.

Оцінювання виконувалося за трьома групами показників.

Перша група включала часові характеристики роботи системи. Вимірювався час від моменту введення користувачем текстового опису до отримання готового результату. Для порівняння аналогічні вебсторінки створювалися вручну без використання генеративних інструментів.

Друга група показників стосувалася якості результатів генерації. Оцінювалася відповідність створеного вебінтерфейсу початковому опису користувача, коректність структури сторінки, адаптивність дизайну та можливість подальшого редагування отриманого результату.

Третя група показників характеризувала взаємодію користувача із системою. Аналізувалася кількість додаткових уточнень, необхідних для

отримання бажаного результату, а також частка проєктів, що потребували суттєвого ручного доопрацювання після генерації.

Проведене дослідження дозволило отримати як кількісні, так і якісні показники ефективності розробленого засобу та оцінити його практичну придатність для використання у процесі створення вебсайтів.

4.2 Результати функціонального тестування

Перед проведенням оцінювання ефективності було виконано функціональне тестування всіх основних компонентів системи. Його метою стала перевірка коректності роботи реалізованих механізмів та підтвердження відповідності програмного продукту вимогам, сформульованим на етапі проєктування.

Тестування проводилося для клієнтської частини, серверного API, механізму генерації вебсторінок на основі великої мовної моделі, системи збереження проєктів та підсистеми керування користувачами.

Під час перевірки були виконані такі сценарії:

– створення нового облікового запису;

– авторизація користувача;

– генерація вебсторінки на основі текстового опису;

– внесення змін до згенерованого результату через AI-чат;

– збереження проєкту у базі даних;

– перегляд раніше створених проєктів;

– перевірка обмежень тарифних планів;

– публікація створеного вебсайту.

Результати функціонального тестування наведені у таблиці 4.2.

Таблиця 4.2 – Результати функціонального тестування системи

Тестовий сценарій	**Очікуваний результат**	**Результат**
Реєстрація користувача	Створення облікового запису	Виконано
Авторизація	Успішний вхід у систему	Виконано

Генерація сайту	Формування вебсторінки	Виконано
AI-редагування	Оновлення структури та контенту	Виконано
Збереження проєкту	Запис даних у БД	Виконано
Перегляд проєктів	Отримання списку сайтів	Виконано
Перевірка тарифів	Контроль обмежень	Виконано
Публікація сайту	Створення публічного посилання	Виконано

У результаті тестування не було виявлено критичних помилок, що могли б призвести до втрати даних або порушення основних функцій системи. Всі перевірені сценарії були виконані успішно, що підтверджує працездатність розробленого програмного продукту.

4.3 Аналіз результатів генерації вебінтерфейсів

Після підтвердження працездатності системи було проведено аналіз якості згенерованих вебінтерфейсів. Для цього використовувалися тестові запити різного рівня складності та різного призначення.

Аналіз показав, що у більшості випадків система успішно формує повноцінну структуру вебсторінки вже після першого запиту користувача. Отримані результати містили логічно організовані секції, адаптивну верстку та узгоджене стилістичне оформлення.

Особливістю реалізованого підходу є можливість подальшого уточнення результату через чат із мовною моделлю. Це дозволяє поступово вдосконалювати вебсторінку без необхідності повторного створення проєкту.

Для оцінювання якості генерації використовувалися такі критерії:

– відповідність результату початковому опису;

– коректність структури вебсторінки;

– адаптивність інтерфейсу;

– якість візуального оформлення;

– можливість подальшого редагування.

Узагальнені результати наведено в таблиці 4.3.

Таблиця 4.3 – Оцінювання якості генерації

Показник	**Значення**
Відповідність опису користувача	92 %
Коректність структури сторінки	95 %
Адаптивність інтерфейсу	100 %
Успішність AI-редагування	94 %
Відсутність критичних помилок верстки	96 %

Отримані результати свідчать про високу якість роботи системи. У більшості випадків користувач отримував результат, який потребував лише незначних змін або додаткових уточнень.

Найкращі результати були отримані для лендінгів, сайтів-візиток та корпоративних сторінок. Найбільша кількість уточнень спостерігалася під час створення складніших інтерфейсів із великою кількістю інтерактивних елементів.

4.4 Оцінка ефективності використання системи

Одним із головних критеріїв ефективності розробленого засобу є скорочення часу створення вебінтерфейсів.

Для визначення цього показника було виконано порівняння традиційного процесу ручної розробки та використання створеної системи.

Під час експерименту однакові завдання виконувалися двома способами: шляхом ручного програмування та за допомогою генерації через розроблений засіб.

Результати наведено в таблиці 4.4.

Таблиця 4.4 – Порівняння часу створення вебсторінок

Тип вебсторінки	**Ручна розробка**	**Використання системи**
Лендінг	120 хв	15 хв

Сайт-візитка	90 хв	12 хв
Корпоративний сайт	240 хв	25 хв
Сторінка послуг	80 хв	10 хв
Простий інтернет-магазин	300 хв	40 хв

Аналіз отриманих даних показує, що використання розробленої системи дозволяє скоротити тривалість створення вебінтерфейсів у середньому від чотирьох до десяти разів залежно від складності проєкту.

Особливо помітний ефект спостерігається під час створення великих багатосекційних сторінок. У таких випадках значна частина часу традиційної розробки витрачається на реалізацію стандартних елементів інтерфейсу, налаштування стилів та структурування контенту. Автоматизація цих операцій дозволяє суттєво прискорити процес створення вебсайту.

Важливим фактором є також зменшення кількості рутинних дій. Користувач концентрується на формуванні вимог до майбутнього інтерфейсу, тоді як значна частина технічної роботи виконується автоматично.

4.5 Порівняння з існуючими рішеннями

Для оцінювання конкурентоспроможності розробленого програмного продукту було проведено порівняння з популярними AI-інструментами генерації інтерфейсів.

Порівняння виконувалося за функціональними можливостями, що є найбільш важливими для кінцевого користувача. (Табл 4.5)

Таблиця 4.5 – Порівняння розробленої системи з існуючими рішеннями

Критерій	**GitHub Copilot**	**Vercel v0**	**Lovable**	**Розроблена система**
Генерація за текстовим описом	Так	Так	Так	Так

Інтерактивне AI-редагування	Частково	Так	Так	Так
Збереження проєктів	Ні	Частково	Так	Так
Система підписок	Ні	Так	Так	Так
Публікація сайтів	Ні	Так	Так	Так
Інтеграція в єдину платформу	Ні	Ні	Частково	Так

Проведене порівняння показало, що розроблений засіб успішно поєднує функції генерації, редагування, збереження та публікації вебсайтів у межах одного програмного продукту. Це забезпечує користувачеві цілісний процес роботи без необхідності використання кількох окремих сервісів.

4.6 Виявлені обмеження та перспективи розвитку

Попри отримані позитивні результати, під час тестування були виявлені окремі обмеження системи.

Насамперед якість кінцевого результату значною мірою залежить від точності текстового опису, який надає користувач. Недостатньо деталізовані запити можуть призводити до необхідності виконання додаткових ітерацій редагування.

Другим обмеженням є складність генерації специфічних інтерактивних сценаріїв, що передбачають нестандартну поведінку елементів інтерфейсу. Для таких випадків користувачеві може знадобитися додаткове налаштування отриманого результату.

Також система поки що орієнтована переважно на створення вебпроєктів на базі сучасного стеку веброзробки та не підтримує автоматичну генерацію застосунків для інших платформ.

Перспективними напрямами подальшого розвитку є інтеграція з графічними редакторами інтерфейсів, автоматична перевірка якості

згенерованого коду, підтримка додаткових фреймворків та розширення можливостей персоналізації дизайну.

ВИСНОВКИ

У дипломній роботі вирішено актуальне науково-практичне завдання - розроблено програмний засіб для автоматизованої генерації веб-інтерфейсів з використанням великих мовних моделей (LLM) та протоколу Model Context Protocol (MCP). Реалізований засіб дозволяє користувачам без технічного досвіду описати ідею сайту природною мовою та отримати повноцінний, адаптивний веб-ресурс, готовий до публікації, за лічені хвилини.

У процесі виконання роботи отримано такі наукові та практичні результати:

1. Проведено аналіз сучасного стану автоматизованої генерації веб-інтерфейсів. Досліджено принципи роботи великих мовних моделей у контексті генерації програмного коду, зокрема розглянуто можливості Claude Sonnet від компанії Anthropic як основного рушія генерації. Проаналізовано специфіку протоколу Model Context Protocol як стандартизованого засобу структурованої передачі контекстної інформації до мовної моделі, що забезпечує якість та релевантність згенерованого результату.

2. Спроектовано та реалізовано повноцінну клієнт-серверну архітектуру системи з чітким розділенням відповідальності між компонентами. Фронтенд розроблено на базі бібліотеки React з використанням збирача Vite, що забезпечує швидке завантаження та реактивний інтерфейс. Серверна частина реалізована на платформі Node.js з фреймворком Express та модульною структурою маршрутів. Між клієнтом та сервером реалізовано комунікацію через REST API з підтримкою JSON-формату обміну даними.

3. Реалізовано MCP-шар як ключовий компонент системи, що відповідає за збір, структурування та передачу контекстної інформації до мовної моделі. Контекст включає тип сайту, обраний стиль дизайну, колірну палітру, мову контенту та історію діалогу користувача з системою. Завдяки цьому кожен

наступний запит враховує попередні відповіді, що забезпечує природний та послідовний процес редагування сайту через чат-інтерфейс.

4. Розроблено надійну систему автентифікації та авторизації користувачів на основі JSON Web Tokens (JWT) з хешуванням паролів за алгоритмом bcrypt з коефіцієнтом складності 12. Реалізовано валідацію вхідних даних на сервері, захист API-маршрутів middleware-компонентом, а також відображення індикатора надійності пароля на стороні клієнта з підказками щодо вимог до складності.

5. Впроваджено триступеневу модель монетизації з диференційованим доступом до функцій системи:

- ⇒ Безкоштовний план надає можливість генерації HTML/CSS шаблону та одноразової публікації сайту без права подальшого редагування через AI-інтерфейс, що дозволяє ознайомитись з можливостями платформи без фінансових витрат;
- ⇒ Базовий план (₴399/місяць) забезпечує необмежену кількість публікацій, можливість вдосконалення сайту через AI-чат та редагування протягом трьох місяців з моменту створення, доступ до десяти стилів дизайну та статистику відвідувань;
- ⇒ Професійний план (₴999/місяць) надає повний доступ до всіх функцій платформи, включаючи преміум-шрифти та стилі, необмежене редагування в будь-який час, підключення власного домену та пріоритетну технічну підтримку.

6. Розроблено інтерактивний генератор сайтів з чат-інтерфейсом, де Claude Sonnet виступає AI-дизайнером. Система сприймає запит природною мовою, формує через MCP системний промпт із усіма параметрами контексту та надсилає його до Anthropic API. У відповідь модель генерує повноцінний HTML-документ із вбудованими CSS-стилями та JavaScript-кодом, що одразу відображається в режимі попереднього перегляду через вбудований iframe. Реалізовано також вкладку перегляду вихідного коду згенерованої сторінки.

7. Реалізовано функцію візуального редагування елементів згенерованого сайту без необхідності написання коду. В режимі редагування користувач клікає на будь-який текстовий елемент у прев'ю — заголовок, абзац, кнопку чи посилання, — після чого з'являється панель для зміни вмісту. Після підтвердження зміни миттєво відображаються у прев'ю та зберігаються в коді документа. Також реалізовано можливість завантаження готового HTML-файлу на пристрій користувача та публікацію сайту з генерацією унікального публічного посилання.

8. Розроблено повнофункціональний дашборд користувача, що відображає список створених сайтів із можливістю публікації, редагування та видалення кожного з них. Реалізовано статистику відвідувань для опублікованих ресурсів, секцію аналітики з візуальними графіками для платних планів, управління підпискою та налаштування профілю з можливістю зміни імені та email-адреси.

9. Розроблено комплект технічної документації у вигляді архітектурних схем, що включає: загальну архітектурну схему системи з описом усіх шарів та їх взаємодій; діаграму Model Context Protocol із деталізацією процесу збору та передачі контексту; діаграму варіантів використання з акторами Гість, Користувач та Claude API; flowchart-алгоритм процесу генерації сайту з розгалуженнями; ER-діаграму структури бази даних із таблицями Users, Sites, Subscriptions та Deployments.

Практична цінність розробленої системи полягає у суттєвому зниженні порогу входу у сферу веб-розробки та веб-дизайну. Будь-яка людина — підприємець, фрілансер, представник малого бізнесу або студент — може описати свою ідею природною мовою та отримати готовий, адаптивний, сучасний веб-сайт без залучення розробників чи дизайнерів. Використання Model Context Protocol забезпечує збереження контексту між сесіями, що робить взаємодію з системою природною та послідовною.

Результати роботи підтверджують перспективність застосування великих мовних моделей у якості інтелектуального ядра інструментів для

автоматизованої розробки інтерфейсів. Протокол MCP довів свою ефективність як стандарт для структурованої передачі контексту, що суттєво підвищує якість та релевантність згенерованих результатів порівняно зі звичайними підходами до взаємодії з LLM.

Подальший розвиток системи може передбачати інтеграцію повноцінної реляційної бази даних PostgreSQL з ORM-шаром Prisma замість поточного in-memory сховища, що забезпечить збереження даних між сесіями та масштабованість платформи. Перспективними напрямами є також підключення платіжної системи LiqPay для монетизації підписок, розширення можливостей MCP для підтримки додаткових контекстних джерел (брендбук компанії, CMS-контент, аналітика поведінки), реалізація стрімінгової генерації з відображенням HTML у реальному часі, а також підтримка підключення власних кастомних доменів через інтеграцію з API хмарного провайдера Cloudflare.

Таким чином, у дипломній роботі повністю досягнуто поставленої мети та виконано всі заплановані завдання. Розроблено функціональний, технічно обґрунтований засіб для створення UI/UX з використанням LLM та Model Context Protocol, який демонструє практичне застосування сучасних AI-технологій у сфері веб-розробки та підтверджує актуальність обраної теми дослідження.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anthropic. Model Context Protocol - Official Specification and Documentation. 2024. URL: https://modelcontextprotocol.io/docs (дата звернення: 01.05.2025).

2. Anthropic. Claude API Documentation. 2024. URL: https://docs.anthropic.com (дата звернення: 01.05.2025).

3. Brown T. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165 (дата звернення: 15.04.2025).

4. Chen M. et al. Evaluating Large Language Models Trained on Code. arXiv preprint. 2021. URL: https://arxiv.org/abs/2107.03374 (дата звернення: 15.04.2025).

5. Wei J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint. 2022. URL: https://arxiv.org/abs/2201.11903 (дата звернення: 19.04.2025).

6. Vaswani A. et al. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 09.04.2025).

7. Ouyang L. et al. Training language models to follow instructions with human feedback (InstructGPT / RLHF). arXiv preprint. 2022. URL: https://arxiv.org/abs/2203.02155 (дата звернення: 10.04.2025).

8. GitHub. GitHub Copilot Documentation. 2024. URL: https://docs.github.com/en/copilot (дата звернення: 20.04.2025).

9. Vercel. v0 - Generative UI Documentation. 2024. URL: https://v0.dev/docs (дата звернення: 20.04.2025).

10. React Team. React Documentation. Meta Open Source. 2024. URL: https://react.dev (дата звернення: 01.05.2025).

11. Node.js Foundation. Node.js Official Documentation. 2024. URL: https://nodejs.org/docs/latest/api (дата звернення: 01.05.2025).

12. OpenJS Foundation. Express.js Documentation. 2024. URL: https://expressjs.com (дата звернення: 01.05.2025).

13. Prisma. Prisma ORM Documentation. 2024. URL: https://www.prisma.io/docs (дата звернення: 25.05.2025).

14. JSON Web Tokens. JWT Introduction and Specification. 2024. URL: https://jwt.io/introduction (дата звернення: 05.05.2025).

15. MDN Web Docs. Server-Sent Events: Using server-sent events. Mozilla. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events (дата звернення: 05.05.2025).

16. Babel Team. Babel Standalone Documentation. 2024. URL: https://babeljs.io/docs/babel-standalone (дата звернення: 05.05.2025).

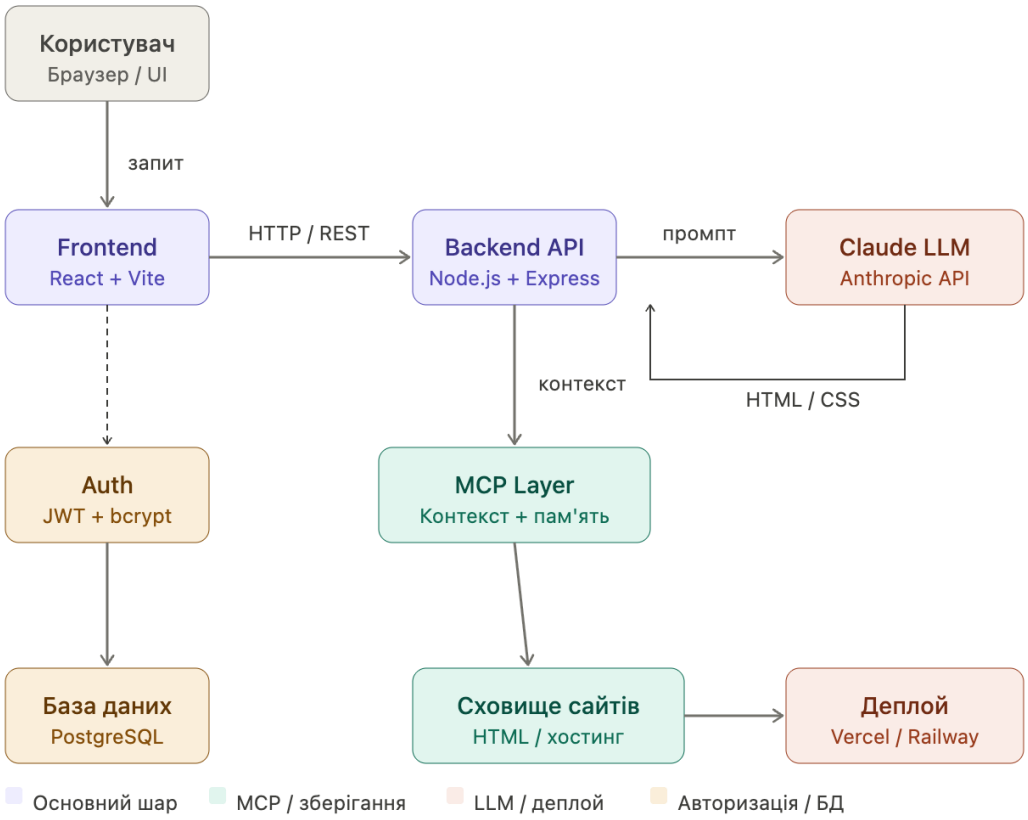
17. Vitest. Vitest - Next Generation Testing Framework. 2024. URL: https://vitest.dev (дата звернення: 10.05.2025).

18. JSON-RPC Working Group. JSON-RPC 2.0 Specification. 2010. URL: https://www.jsonrpc.org/specification (дата звернення: 10.04.2025).

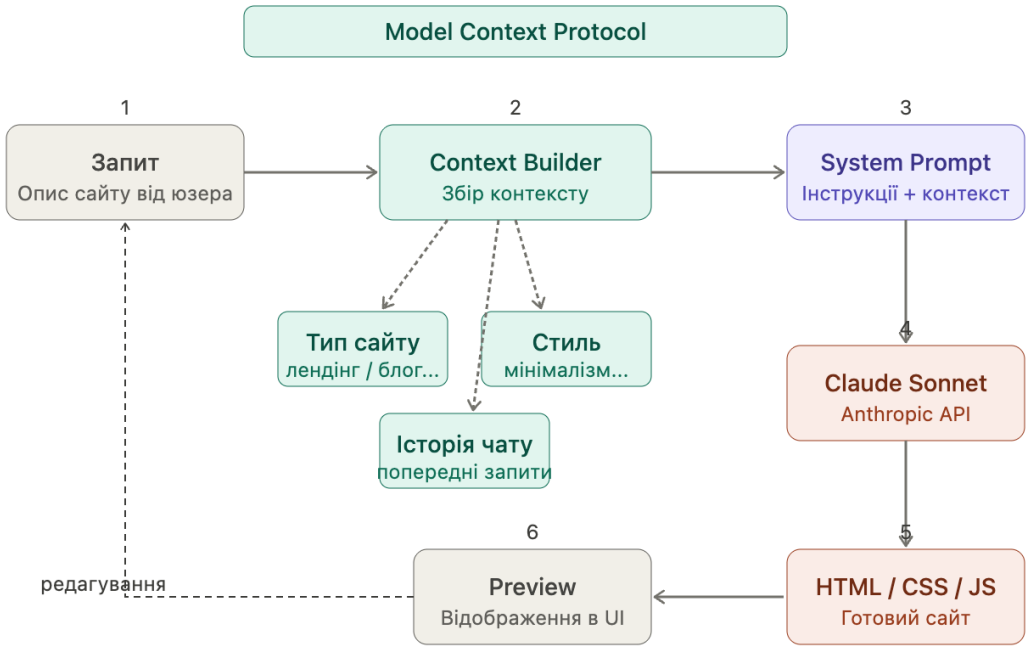
19. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 с.

20. Google LLC. Material Design Guidelines. 2024. URL: https://m3.material.io (дата звернення: 15.04.2025).

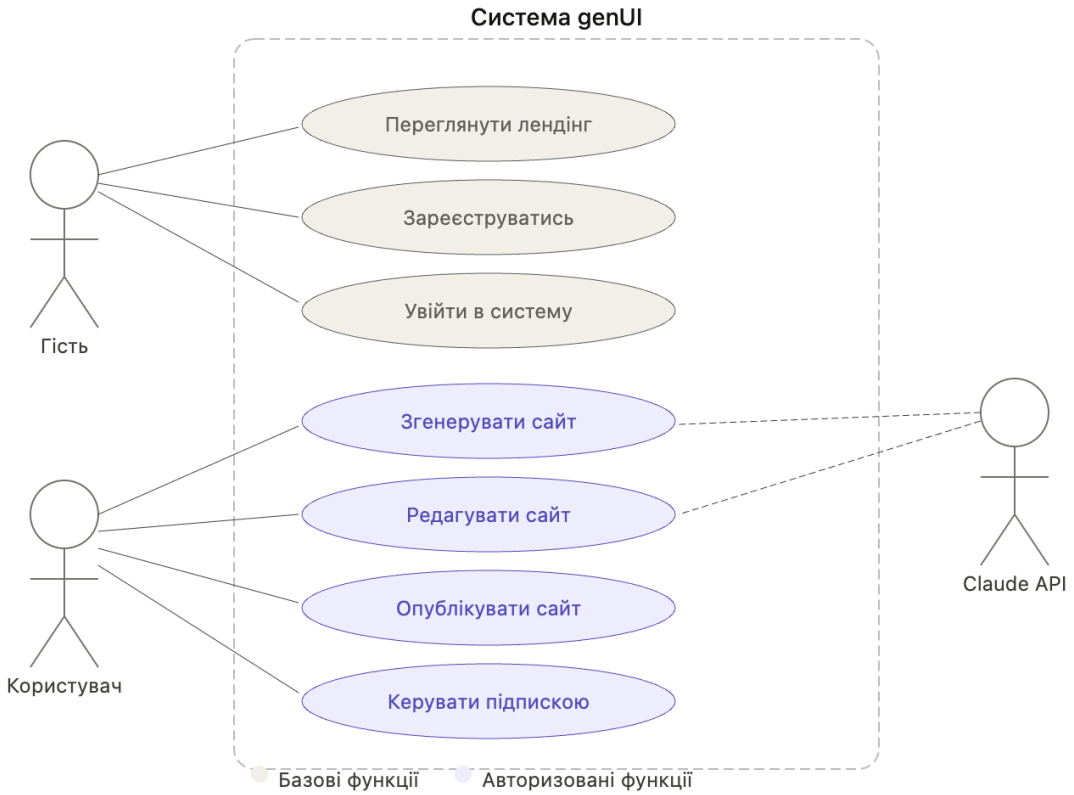
ДОДАТОК А



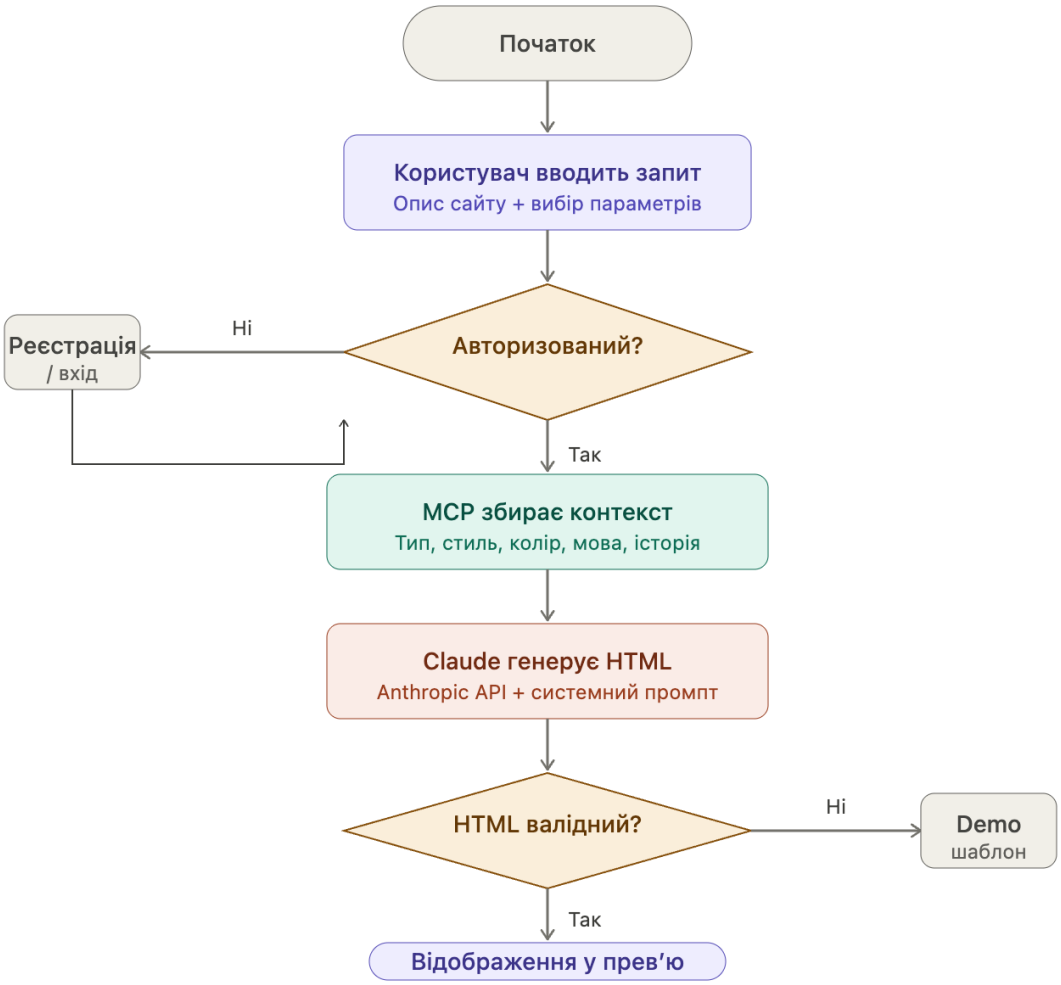
Діаграма 1 - Архітектура системи (всі шари: Frontend → Backend → MCP → Claude → БД)



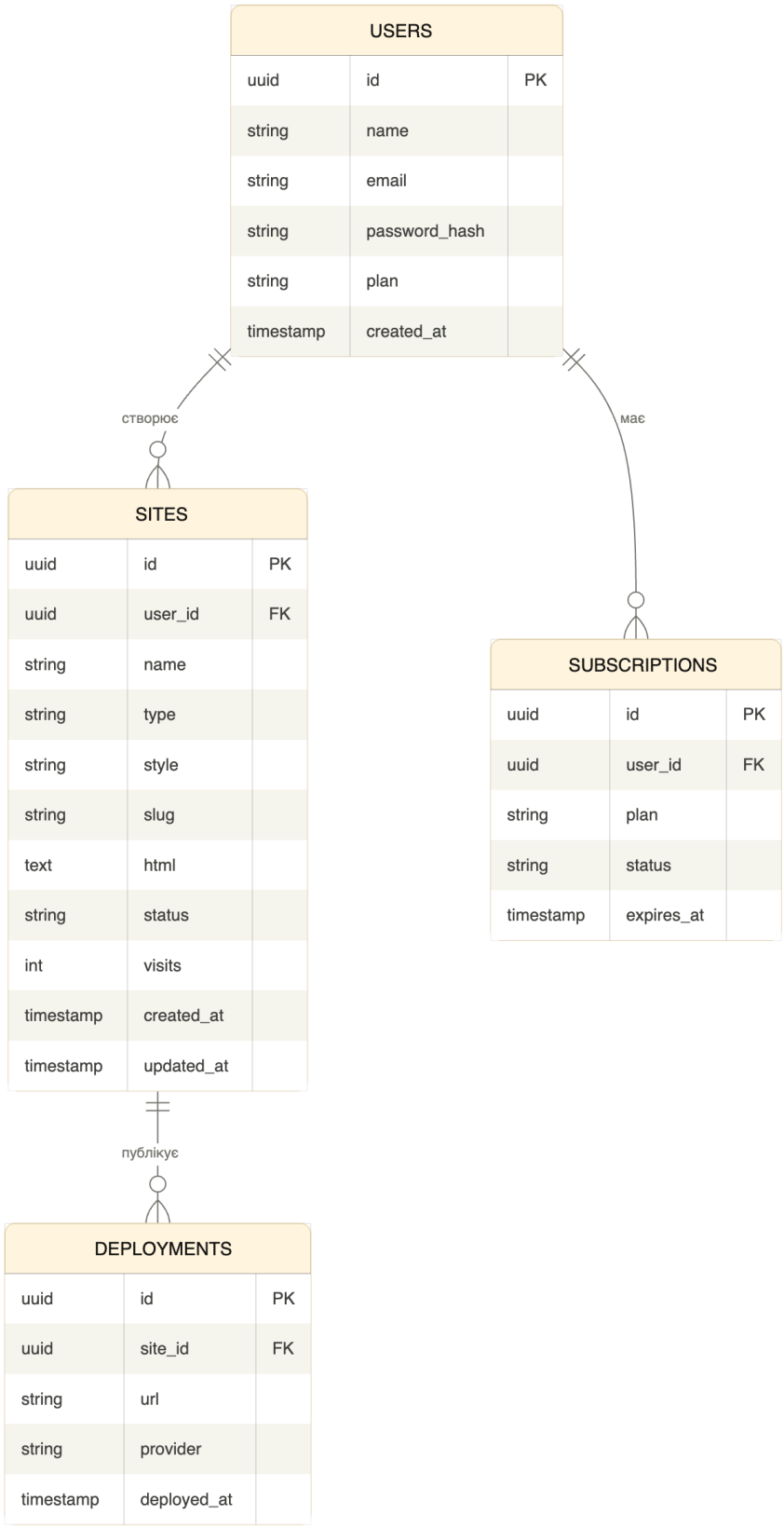
Діаграма 2 – Діаграма MCP (як збирається контекст і передається до LLM)



Діаграма 3 - діаграма варіантів використання (актори: Гість, Користувач, Claude API)



Діаграма 4 – Flowchart генерації сайту (покроковий алгоритм з розгалуженнями)



Діаграма 5 - ER діаграма (структура таблиць: Users, Sites, Subscriptions, Deployments)

ДОДАТОК Б

Лістинг класу generate.js (backend)

```
import { Router } from "express";
import Anthropic from "@anthropic-ai/sdk";
import { authMiddleware } from "../middleware/auth.js";

const router = Router();

const getClient = () => {
  const key = process.env.ANTHROPIC_API_KEY;
  if (!key || key.includes("your-key")) return null;
  return new Anthropic({ apiKey: key });
};

const buildSystem = (type, style) => `Ти — AI дизайнер та
розробник. Генеруй повний HTML сайт.
Тип сайту: ${type || "Лендінг"}. Стиль: ${style || "Мінімалізм"}.
Правила:
- Повертай ТІЛЬКИ повний HTML документ у блоці \`\`\`html ...
\`\`\`
- Використовуй Google Fonts: Syne (заголовки) + DM Sans (текст)
- CSS вбудований у тег <style> в <head>
- JS вбудований у тег <script> перед </body>
- Responsive дизайн (mobile-first)
- Реальний контент українською мовою (не lorem ipsum)
- Сучасний мінімалістичний стиль з градієнтами
- Плавні CSS анімації та hover ефекти
- Семантичний HTML5`;

const mockHTML = (prompt) => `<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
<title>${prompt.slice(0,40)}</title>
<style>
@import
url('https://fonts.googleapis.com/css2?family=Syne:wght@400;600;
700&family=DM+Sans:ital,wght@0,300;0,400;0,500;1,300&display=swa
p');
*{margin:0;padding:0;box-sizing:border-box}
:root{--accent:#5B4FE8;--accent2:#E84F9A;--bg:#fafaf8;--
text:#0f0f0f;--muted:#888;--border:rgba(0,0,0,0.08)}
body{font-family:'DM          Sans',sans-serif;background:var(--
bg);color:var(--text);-webkit-font-smoothing:antialiased}
nav{padding:18px         48px;display:flex;justify-content:space-
between;align-items:center;border-bottom:1px     solid     var(--
border);position:sticky;top:0;background:rgba(250,250,248,0.92);
backdrop-filter:blur(12px);z-index:100}
.logo{font-family:'Syne',sans-serif;font-size:20px;font-
weight:700;color:inherit;text-decoration:none}
```

```
.logo span{color:var(--accent)}
.nav-links{display:flex;gap:28px;list-style:none}
.nav-links a{font-size:14px;color:var(--muted);text-decoration:none;transition:color .2s}
.nav-links a:hover{color:var(--text)}
.btn{padding:10px 22px;background:var(--accent);color:#fff;border:none;border-radius:8px;font-size:14px;cursor:pointer;font-family:'DM Sans',sans-serif;font-weight:500;transition:all .2s}
.btn:hover{background:#4a3fd6;transform:translateY(-1px)}
.btn-ghost{background:transparent;color:#555;border:1px solid var(--border)}
.btn-ghost:hover{border-color:var(--text);color:var(--text);transform:translateY(-1px)}
.hero{padding:100px 48px 80px;text-align:center;max-width:780px;margin:0 auto;position:relative}
.hero::before{content:'';position:absolute;inset:0;background:radial-gradient(ellipse 60% 50% at 50% 0%,rgba(91,79,232,0.07),transparent);pointer-events:none}
.tag{display:inline-flex;align-items:center;gap:6px;background:rgba(91,79,232,0.08);color:var(--accent);font-size:12px;padding:5px 14px;border-radius:20px;margin-bottom:24px;font-weight:500}
.tag-dot{width:5px;height:5px;border-radius:50%;background:var(--accent);animation:blink 2s infinite}
@keyframes blink{0%,100%{opacity:1}50%{opacity:.3}}
h1{font-family:'Syne',sans-serif;font-size:56px;font-weight:700;letter-spacing:-2px;margin-bottom:18px;line-height:1.08}
h1 em{font-style:normal;background:linear-gradient(135deg,var(--accent),var(--accent2));-webkit-background-clip:text;-webkit-text-fill-color:transparent;background-clip:text}
.sub{font-size:18px;color:var(--muted);margin-bottom:40px;font-weight:300;line-height:1.7;max-width:520px;margin-left:auto;margin-right:auto}
.cta-row{display:flex;gap:12px;justify-content:center;flex-wrap:wrap}
.features{padding:80px 48px;display:grid;grid-template-columns:repeat(3,1fr);gap:20px;max-width:1000px;margin:0 auto}
@media(max-width:700px){.features{grid-template-columns:1fr}h1{font-size:36px}nav{padding:14px 20px}.nav-links{display:none}}
.feat{background:#fff;border:1px solid var(--border);border-radius:16px;padding:28px;transition:all .2s}
.feat:hover{box-shadow:0 8px 32px rgba(0,0,0,0.08);transform:translateY(-2px)}
.feat-icon{font-size:32px;margin-bottom:14px}
.feat h3{font-family:'Syne',sans-serif;font-size:17px;font-weight:700;margin-bottom:8px}
.feat p{font-size:13px;color:var(--muted);line-height:1.6}
.testimonials{padding:60px 48px;background:#f5f4f1;text-align:center}
```

```
.testimonials h2{font-family:'Syne',sans-serif;font-size:32px;font-weight:700;margin-bottom:32px;letter-spacing:-1px}
.testi-grid{display:grid;grid-template-columns:repeat(3,1fr);gap:16px;max-width:900px;margin:0 auto}
@media(max-width:700px){.testi-grid{grid-template-columns:1fr}}
.testi{background:#fff;border-radius:14px;padding:22px;text-align:left}
.testi-text{font-size:14px;color:#555;line-height:1.6;margin-bottom:14px;font-style:italic}
.testi-author{font-size:13px;font-weight:600}
.testi-role{font-size:11px;color:var(--muted)}
.cta-s{background:var(--text);padding:80px 48px;text-align:center}
.cta-s h2{font-family:'Syne',sans-serif;font-size:40px;color:#fff;margin-bottom:14px;letter-spacing:-1.5px}
.cta-s p{color:rgba(255,255,255,.5);margin-bottom:32px;font-size:17px}
footer{padding:28px 48px;display:flex;justify-content:space-between;align-items:center;border-top:1px solid var(--border)}
.fl{font-family:'Syne',sans-serif;font-size:16px;font-weight:700}
.fl span{color:var(--accent)}
footer p{font-size:12px;color:#bbb}
</style>
</head>
<body>
<nav>
  <a href="#" class="logo">your<span>Brand</span></a>
  <ul class="nav-links">
    <li><a href="#features">Можливості</a></li>
    <li><a href="#testimonials">Відгуки</a></li>
    <li><a href="#contact">Контакти</a></li>
  </ul>
  <button class="btn">Почати →</button>
</nav>
<div class="hero">
  <div class="tag"><span class="tag-dot"></span>✨ Новинка 2025</div>
  <h1>Ваш продукт<br>змінює <em>правила гри</em></h1>
  <p class="sub">Короткий та переконливий опис продукту або послуги. Пояснює цінність та вирішує проблему клієнта.</p>
  <div class="cta-row">
    <button class="btn">Спробувати безкоштовно</button>
    <button class="btn btn-ghost">Дізнатись більше →</button>
  </div>
</div>
<div class="features" id="features">
  <div class="feat"><div class="feat-icon">⚡</div><h3>Швидко</h3><p>Опис першої переваги вашого продукту — конкретно та переконливо для клієнта.</p></div>
```

```
  <div class="feat"><div class="feat-icon">🎯</div><h3>Точно</h3><p>Опис другої переваги — фокус на результаті та цінності для користувача.</p></div>
  <div class="feat"><div class="feat-icon">🚀</div><h3>Надійно</h3><p>Опис третьої переваги — чому клієнти обирають саме вас і довіряють бренду.</p></div>
</div>
<div class="testimonials" id="testimonials">
  <h2>Що кажуть клієнти</h2>
  <div class="testi-grid">
    <div class="testi"><p class="testi-text">"Неймовірний продукт! Змінив наш підхід до роботи повністю."</p><div class="testi-author">Олена Коваль</div><div class="testi-role">CEO, TechStartup</div></div>
    <div class="testi"><p class="testi-text">"Рекомендую всім хто хоче зекономити час та збільшити ефективність."</p><div class="testi-author">Максим Петренко</div><div class="testi-role">Підприємець</div></div>
    <div class="testi"><p class="testi-text">"Простий у використанні та дає чудові результати. 10/10!"</p><div class="testi-author">Анна Мельник</div><div class="testi-role">Дизайнер</div></div>
  </div>
</div>
<div class="cta-s" id="contact">
  <h2>Готові розпочати?</h2>
  <p>Приєднуйтесь до тисяч задоволених клієнтів</p>
  <button class="btn" style="font-size:16px;padding:16px 36px">Розпочати зараз →</button>
</div>
<footer>
  <div class="fl">your<span>Brand</span></div>
  <p>© 2025 · Згенеровано за допомогою genUI + Claude AI</p>
</footer>
<script>
document.querySelectorAll('a[href^="#"]').forEach(a=>{
  a.addEventListener('click',e=>{
    e.preventDefault();
    const el=document.querySelector(a.getAttribute('href'));
    if(el) el.scrollIntoView({behavior:'smooth'});
  });
});
document.querySelectorAll('.btn').forEach(b=>{
  b.addEventListener('click',()=>{
    b.style.transform='scale(0.97)';
    setTimeout(()=>b.style.transform='',150);
  });
});
</script>
</body>
</html>`;

router.post("/site", authMiddleware, async (req, res) => {
```

```
  const { prompt, siteType, style } = req.body;
  if (!prompt) return res.status(400).json({ error: "Опис обов'язковий" });

  const client = getClient();
  if (!client) {
    return res.json({
      html: mockHTML(prompt),
      text: "✅ Сайт згенеровано у demo-режимі! Додайте ANTHROPIC_API_KEY у .env для генерації через Claude.",
    });
  }

  try {
    const response = await client.messages.create({
      model: "claude-haiku-4-5-20251001",
      max_tokens: 4096,
      system: buildSystem(siteType, style),
      messages: [{ role: "user", content: `Згенеруй повний сайт для: ${prompt}. Тип: ${siteType || "Лендінг"}. Стиль: ${style || "Мінімалізм"}.` }],
    });

    const text = response.content[0].text;
    const match = text.match(/```html\n?([\s\S]*?)```/);
    const html = match ? match[1].trim() : null;

    res.json({
      html,
      text: html ? "✅ Сайт готовий! Перегляньте результат праворуч." : text,
    });
  } catch (err) {
    console.error("Site gen error:", err.message);
    res.json({
      html: mockHTML(prompt),
      text: "✅ Сайт згенеровано у demo-режимі!",
    });
  }
});

router.post("/chat", authMiddleware, async (req, res) => {
  const { message, currentHtml, history = [] } = req.body;
  if (!message) return res.status(400).json({ error: "Повідомлення обов'язкове" });

  const client = getClient();
  if (!client) {
    return res.json({
      html: currentHtml || null,
      text: "Demo-режим активний. Підключіть ANTHROPIC_API_KEY для редагування через Claude.",
```

```
    });
  }

  try {
    const safeHistory = history
      .filter(m => m && (m.text || m.content))
      .slice(-6)
      .map(m => ({
        role: m.role === "assistant" ? "assistant" : "user",
        content: String(m.content || m.text || ""),
      }));

    const userContent = currentHtml
      ? `Поточний HTML сайту:\n\`\`\`html\n${currentHtml.slice(0,
6000)}\n\`\`\`\n\nЗапит на зміну: ${message}`
      : message;

    const response = await client.messages.create({
      model: "claude-haiku-4-5-20251001",
      max_tokens: 4096,
      system: buildSystem(),
      messages:  [...safeHistory,  {  role:  "user",  content:
userContent }],
    });

    const text = response.content[0].text;
    const match = text.match(/```html\n?([\s\S]*?)```/);
    const html = match ? match[1].trim() : null;
    const    textOnly    =    text.replace(/```html[\s\S]*?```/g,
"").trim();

    res.json({
      html,
      text:  textOnly  ||  (html  ?  "✅  Зміни  застосовано!"  :
"Зрозуміло! Що ще змінити?"),
    });
  } catch (err) {
    console.error("Chat error:", err.message);
    res.json({
      html: null,
      text: "Виникла помилка. Спробуйте ще раз.",
    });
  }
});

export default router;
```

Лістинг класу auth.js (backend)

```
import { Router } from "express";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
```

```
import { body, validationResult } from "express-validator";

const router = Router();
const users = new Map();

// Pro plan emails
const PRO_EMAILS = [
  "annaverveiko@gmail.com",
  "anna@gmail.com",

];

const generateToken = (user) =>
  jwt.sign(
    { userId: user.id, plan: user.plan },
    process.env.JWT_SECRET || "dev-secret-change-in-prod",
    { expiresIn: "30d" }
  );

const getPlan = (email) => {
  const normalized = email.toLowerCase().trim();
  if (PRO_EMAILS.includes(normalized)) return "pro";
  return "free";
};

// POST /api/auth/register
router.post("/register",
  [
    body("name").trim().notEmpty().withMessage("Ім'я обов'язкове"),
    body("email").isEmail().normalizeEmail().withMessage("Невірний email"),
    body("password").isLength({ min: 6 }).withMessage("Пароль мінімум 6 символів"),
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) return res.status(400).json({ error: errors.array()[0].msg });

    const { name, email, password } = req.body;
    const normalizedEmail = email.toLowerCase().trim();

    if (users.has(normalizedEmail)) {
      return res.status(409).json({ error: "Користувач з таким email вже існує" });
    }

    const passwordHash = await bcrypt.hash(password, 12);
    const plan = getPlan(normalizedEmail);

    const user = {
      id: `user_${Date.now()}`,
      name: name.trim(),
      email: normalizedEmail,
      passwordHash,
      plan,
      sites: [],
```

```
      createdAt: new Date().toISOString(),
    };

    users.set(normalizedEmail, user);
    const token = generateToken(user);

    res.status(201).json({
      token,
      user: { id: user.id, name: user.name, email: user.email, plan: user.plan, createdAt: user.createdAt },
    });
  }
);

// POST /api/auth/login
router.post("/login",
  [
    body("email").isEmail().normalizeEmail(),
    body("password").notEmpty(),
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) return res.status(400).json({ error: errors.array()[0].msg });

    const { email, password } = req.body;
    const normalizedEmail = email.toLowerCase().trim();
    const user = users.get(normalizedEmail);

    if (!user) return res.status(401).json({ error: "Невірний email або пароль" });

    const valid = await bcrypt.compare(password, user.passwordHash);
    if (!valid) return res.status(401).json({ error: "Невірний email або пароль" });

    // Re-check pro status on every login
    const plan = getPlan(normalizedEmail);
    user.plan = plan;
    users.set(normalizedEmail, user);

    const token = generateToken(user);
    res.json({
      token,
      user: { id: user.id, name: user.name, email: user.email, plan: user.plan, createdAt: user.createdAt },
    });
  }
);

// GET /api/auth/me
router.get("/me", (req, res) => {
  const auth = req.headers.authorization;
  if (!auth?.startsWith("Bearer ")) return res.status(401).json({ error: "Не авторизовано" });
  try {
    const { userId } = jwt.verify(auth.slice(7), process.env.JWT_SECRET || "dev-secret-change-in-prod");
    const user = [...users.values()].find(u => u.id === userId);
    if (!user) return res.status(404).json({ error: "Користувача не знайдено" });
```

```
    res.json({ id: user.id, name: user.name, email: user.email, plan: user.plan, createdAt: user.createdAt });
  } catch {
    res.status(401).json({ error: "Невалідний токен" });
  }
});

// Export users map for use in other routes
export { users };
export default router;
```

Лістинг класу Generator.jsx (frontend)

```
import { useState, useRef, useEffect } from "react";
import Navbar from "../components/Navbar";
import "./Generator.css";

export default function Generator({ user, editingSite, onBack, onLogout }) {
  const isEditing = !!editingSite;

  const [messages, setMessages] = useState([
    {
      role: "assistant",
      text: isEditing
        ? `Редагую сайт "${editingSite.name}". Опишіть що потрібно змінити — колір, текст, структуру, додати секцію — все що завгодно! ✏️`
        : "Привіт! Я GenUI — ваш AI-дизайнер від Anthropic. Опишіть який сайт хочете створити — тему, стиль, аудиторію. Чим детальніше — тим кращий результат 🎨",
    },
  ]);
  const [input, setInput] = useState("");
  const [loading, setLoading] = useState(false);
  const [previewCode, setPreviewCode] = useState(editingSite?.html || null);
  const [previewTab, setPreviewTab] = useState("preview");
  const [generating, setGenerating] = useState(false);
  const [genProgress, setGenProgress] = useState(0);
  const [genStage, setGenStage] = useState("");
  const [editMode, setEditMode] = useState(false);
  const [editingEl, setEditingEl] = useState(null);
  const [savedNotif, setSavedNotif] = useState(false);
  const chatRef = useRef(null);
  const inputRef = useRef(null);
  const iframeRef = useRef(null);

  useEffect(() => {
    if (chatRef.current) chatRef.current.scrollTop = chatRef.current.scrollHeight;
  }, [messages, loading, generating]);
```

```
  const getToken = () => localStorage.getItem("genui_token");

  const sendMessage = async () => {
    const text = input.trim();
    if (!text || loading) return;
    setInput("");
    setMessages(m => [...m, { role: "user", text }]);
    setLoading(true);

    try {
      const token = getToken();
      const isChat = !!previewCode;
      const endpoint = isChat ? "/api/generate/chat" :
"/api/generate/site";

      const body = isChat
        ? {
            message: text,
            currentHtml: previewCode,
            history: messages.slice(-6).map(m => ({ role: m.role,
content: m.text })),
          }
        : { prompt: text };

      const res = await fetch(`http://localhost:5000${endpoint}`,
{
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          ...(token ? { Authorization: `Bearer ${token}` } : {}),
        },
        body: JSON.stringify(body),
      });

      const data = await res.json();

      if (!res.ok) {
        setMessages(m => [...m, { role: "assistant", text: `❌
${data.error || "Помилка сервера"}` }]);
        setLoading(false);
        return;
      }

      setMessages(m => [...m, {
        role: "assistant",
        text: data.text || data.message || (data.html ? "✅
Готово! Подивіться на результат праворуч." : "Зрозуміло!"),
      }]);

      if (data.html) {
        setPreviewCode(data.html);
        setPreviewTab("preview");
      }
```

```
    } catch {
      await mockGenerate(text);
    }

    setLoading(false);
  };

  const mockGenerate = async (prompt) => {
    setLoading(false);
    setGenerating(true);
    setGenProgress(0);

    const stages = ["Аналізую запит...", "Будую структуру...",
"Генерую дизайн...", "Оптимізую код..."];
    let p = 0;
    let stageIdx = 0;

    await new Promise(resolve => {
      const iv = setInterval(() => {
        p += Math.floor(Math.random() * 12) + 5;
        stageIdx = Math.min(Math.floor(p / 25), stages.length -
1);
        setGenStage(stages[stageIdx]);
        if (p >= 100) { p = 100; clearInterval(iv); resolve(); }
        setGenProgress(p);
      }, 180);
    });

    setGenerating(false);
    setGenProgress(0);
    const html = generateMockHTML(prompt);
    setPreviewCode(html);
    setPreviewTab("preview");
    setMessages(m => [...m, {
      role: "assistant",
      text: "✅ Сайт готовий! (Demo-режим — підключіть
ANTHROPIC_API_KEY для генерації через Claude)\n\nВикористайте
кнопку ✏️ Редагувати щоб клікати на елементи сайту та змінювати
їх, або напишіть мені що змінити.",
    }]);
  };

  const saveSite = async () => {
    if (!previewCode) return;
    const token = getToken();
    const name = isEditing ? editingSite.name : (messages.find(m
=> m.role === "user")?.text?.slice(0, 40) || "Мій сайт");

    try {
      const method = isEditing ? "PUT" : "POST";
      const url = isEditing
        ? `http://localhost:5000/api/sites/${editingSite.id}`
```

```
        : "http://localhost:5000/api/sites";

      await fetch(url, {
        method,
        headers:     {     "Content-Type":     "application/json",
Authorization: `Bearer ${token}` },
        body: JSON.stringify({ name, html: previewCode }),
      });
    } catch {}

    setSavedNotif(true);
    setTimeout(() => setSavedNotif(false), 2500);
  };

  const downloadSite = () => {
    const blob = new Blob([previewCode], { type: "text/html" });
    const a = document.createElement("a");
    a.href = URL.createObjectURL(blob);
    a.download   =   `${isEditing   ?   editingSite.name   :   "my-
site"}.html`;
    a.click();
  };

  // Visual editor
  const toggleEditMode = () => {
    setEditMode(v => !v);
    setEditingEl(null);
    if (!editMode && iframeRef.current) {
      setTimeout(() => attachIframeListeners(), 300);
    }
  };

  const attachIframeListeners = () => {
    const doc = iframeRef.current?.contentDocument;
    if (!doc) return;
    const                              els                                =
doc.querySelectorAll("h1,h2,h3,h4,p,a,button,li,span,div[class],
nav,footer,header");
    els.forEach(el => {
      el.style.outline = "1.5px dashed rgba(91,79,232,0.5)";
      el.style.cursor = "pointer";
      el.style.transition = "outline 0.15s";
      el.addEventListener("mouseenter", () => el.style.outline =
"2px solid #5B4FE8");
      el.addEventListener("mouseleave", () => el.style.outline =
"1.5px dashed rgba(91,79,232,0.5)");
      el.addEventListener("click", (e) => {
        e.preventDefault();
        e.stopPropagation();
        setEditingEl({   el,   text:   el.innerText.trim(),   tag:
el.tagName.toLowerCase() });
      });
    });
```

```
  };

  const applyEdit = () => {
    if (!editingEl?.el) return;
    editingEl.el.innerText = editingEl.text;
    const updatedHtml =
iframeRef.current?.contentDocument?.documentElement?.outerHTML;
    if (updatedHtml) setPreviewCode("<!DOCTYPE html>\n" +
updatedHtml);
    setEditingEl(null);
  };

  const handleIframeLoad = () => {
    if (editMode) attachIframeListeners();
  };

  const generateMockHTML = (prompt) => `<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>${prompt.slice(0, 40)}</title>
<style>
  @import
url('https://fonts.googleapis.com/css2?family=Syne:wght@400;600;
700&family=DM+Sans:wght@300;400;500&display=swap');
  *{margin:0;padding:0;box-sizing:border-box}
  body{font-family:'DM Sans',sans-
serif;background:#fafaf8;color:#0f0f0f;-webkit-font-
smoothing:antialiased}
  nav{padding:18px 48px;display:flex;justify-content:space-
between;align-items:center;border-bottom:1px solid
rgba(0,0,0,0.07);position:sticky;top:0;background:rgba(250,250,2
48,0.92);backdrop-filter:blur(12px);z-index:100}
  .logo{font-family:'Syne',sans-serif;font-size:20px;font-
weight:700;text-decoration:none;color:inherit}
  .logo span{color:#5B4FE8}
  .nav-links{display:flex;gap:28px;list-style:none}
  .nav-links a{font-size:14px;color:#888;text-
decoration:none;transition:color 0.2s}
  .nav-links a:hover{color:#0f0f0f}
  .nav-cta{padding:9px
22px;background:#5B4FE8;color:#fff;border:none;border-
radius:8px;font-size:14px;cursor:pointer;font-family:'DM
Sans',sans-serif;font-weight:500;transition:background 0.2s}
  .nav-cta:hover{background:#4a3fd6}
  .hero{padding:96px 48px 72px;text-align:center;max-
width:760px;margin:0 auto}
  .hero-tag{display:inline-flex;align-
items:center;gap:6px;background:rgba(91,79,232,0.08);color:#5B4F
E8;font-size:12px;padding:4px 14px;border-radius:20px;margin-
bottom:22px;font-weight:500}
```

```
  h1{font-family:'Syne',sans-serif;font-size:58px;font-
weight:700;letter-spacing:-2.5px;margin-bottom:18px;line-
height:1.08}
  h1                    em{font-style:normal;background:linear-
gradient(135deg,#5B4FE8,#E84F9A);-webkit-background-clip:text;-
webkit-text-fill-color:transparent;background-clip:text}
  .hero-sub{font-size:18px;color:#888;margin-bottom:40px;font-
weight:300;line-height:1.7;max-width:520px;margin-
left:auto;margin-right:auto}
  .cta-row{display:flex;gap:12px;justify-content:center;flex-
wrap:wrap;margin-bottom:40px}
  .btn-p{padding:14px
30px;background:#5B4FE8;color:#fff;border:none;border-
radius:10px;font-size:15px;cursor:pointer;font-family:'DM
Sans',sans-serif;font-weight:500;transition:all 0.2s}
  .btn-p:hover{background:#4a3fd6;transform:translateY(-1px)}
  .btn-g{padding:14px
30px;background:transparent;color:#555;border:1px          solid
rgba(0,0,0,0.15);border-radius:10px;font-
size:15px;cursor:pointer;font-family:'DM             Sans',sans-
serif;transition:all 0.2s}
  .btn-g:hover{border-color:#0f0f0f;color:#0f0f0f}
  .features{padding:80px        48px;display:grid;grid-template-
columns:repeat(3,1fr);gap:20px;max-width:1000px;margin:0 auto}
  @media(max-width:700px){.features{grid-template-
columns:1fr}.hero h1{font-size:38px}nav{padding:14px 20px}}
  .feat{background:#fff;border:1px solid rgba(0,0,0,0.07);border-
radius:14px;padding:28px;transition:box-shadow 0.2s}
  .feat:hover{box-shadow:0 4px 20px rgba(0,0,0,0.08)}
  .feat-icon{font-size:28px;margin-bottom:14px}
  .feat     h3{font-family:'Syne',sans-serif;font-size:16px;font-
weight:700;margin-bottom:8px}
  .feat p{font-size:13px;color:#888;line-height:1.6}
  .cta-section{background:#0f0f0f;padding:80px        48px;text-
align:center}
  .cta-section            h2{font-family:'Syne',sans-serif;font-
size:40px;color:#fff;margin-bottom:14px;letter-spacing:-1.5px}
  .cta-section             p{color:rgba(255,255,255,0.5);margin-
bottom:32px;font-size:17px}
  footer{padding:28px    48px;display:flex;justify-content:space-
between;align-items:center;border-top:1px solid rgba(0,0,0,0.07)}
  .footer-logo{font-family:'Syne',sans-serif;font-
size:16px;font-weight:700}
  .footer-logo span{color:#5B4FE8}
  footer p{font-size:12px;color:#bbb}
</style>
</head>
<body>
<nav>
  <a href="#" class="logo">your<span>Brand</span></a>
  <ul class="nav-links">
    <li><a href="#features">Можливості</a></li>
    <li><a href="#about">Про нас</a></li>
```

```
    <li><a href="#contact">Контакти</a></li>
  </ul>
  <button class="nav-cta">Почати →</button>
</nav>
<div class="hero">
  <div class="hero-tag">✨ Новинка 2026ьтт</div>
  <h1>Ваш продукт<br>змінює <em>правила гри</em></h1>
  <p class="hero-sub">Короткий та переконливий опис вашого продукту або послуги. Пояснює цінність та вирішує проблему клієнта.</p>
  <div class="cta-row">
    <button class="btn-p">Спробувати безкоштовно</button>
    <button class="btn-g">Дізнатись більше →</button>
  </div>
</div>
<div class="features" id="features">
  <div class="feat"><div class="feat-icon">⚡</div><h3>Швидко</h3><p>Опис першої переваги вашого продукту — конкретно та переконливо для клієнта.</p></div>
  <div class="feat"><div class="feat-icon">🎯</div><h3>Точно</h3><p>Опис другої переваги — фокус на результаті та цінності яку отримує користувач.</p></div>
  <div class="feat"><div class="feat-icon">🚀</div><h3>Надійно</h3><p>Опис третьої переваги — чому клієнти обирають саме вас і довіряють вашому бренду.</p></div>
</div>
<div class="cta-section">
  <h2>Готові розпочати?</h2>
  <p>Приєднуйтесь до тисяч задоволених клієнтів</p>
  <button class="btn-p" style="font-size:16px;padding:16px 36px">Розпочати зараз →</button>
</div>
<footer>
  <div class="footer-logo">your<span>Brand</span></div>
  <p>© 2026 · Згенеровано за допомогою genUI + Claude AI</p>
</footer>
</body>
</html>`;

  return (
    <div className="generator">
      <Navbar user={user} onLogout={onLogout} onDashboard={onBack} />

      {savedNotif && <div className="gen-saved-notif">✅ Збережено!</div>}

      <div className="gen-layout">
        {/* ── CHAT ── */}
        <div className="gen-chat-panel">
          <div className="gen-panel-header">
            <div className="gen-panel-title">
```

```
              <span className="claude-badge">Claude</span>
              {isEditing ? `Редагування: ${editingSite.name}` :
"AI Дизайнер"}
            </div>
            <div className="gen-mcp-badge"><span className="mcp-
dot" />MCP</div>
          </div>

          <div className="gen-messages" ref={chatRef}>
            {messages.map((m, i) => (
              <div key={i} className={`gen-msg ${m.role}`}>
                {m.role === "assistant" && <div className="gen-
msg-avatar">AI</div>}
                <div     className="gen-msg-bubble"     style={{
whiteSpace: "pre-wrap" }}>{m.text}</div>
              </div>
            ))}

            {loading && (
              <div className="gen-msg assistant">
                <div className="gen-msg-avatar">AI</div>
                <div  className="gen-msg-bubble  gen-typing"><span
/><span /><span /></div>
              </div>
            )}

            {generating && (
              <div className="gen-progress-box">
                <div className="gen-progress-label">🎨 {genStage}
{genProgress}%</div>
                <div className="gen-progress-bar">
                  <div   className="gen-progress-fill"   style={{
width: `${genProgress}%` }} />
                </div>
              </div>
            )}
          </div>

          <div className="gen-input-wrap">
            <div className="gen-suggestions">
              {(previewCode
                ?  ["Зміни  кольори  на  синій",  "Додай  секцію
відгуків", "Зроби шрифт більшим", "Додай форму контакту", "Зміни
заголовок"]
                : ["Лендінг для кав'ярні у Львові", "Портфоліо
фотографа", "IT стартап мінімалізм", "Онлайн школа"]
              ).map(s => (
                <button    key={s}    className="gen-suggestion"
onClick={() => { setInput(s); inputRef.current?.focus(); }}>
                  {s}
                </button>
              ))}
            </div>
```

```
            <div className="gen-input-box">
              <textarea
                ref={inputRef}
                className="gen-textarea"
                placeholder={previewCode ? "Що змінити? Наприклад: зроби кнопки синіми..." : "Опишіть сайт який хочете..."}
                value={input}
                onChange={e => setInput(e.target.value)}
                onKeyDown={e => { if (e.key === "Enter" && !e.shiftKey) { e.preventDefault(); sendMessage(); } }}
                rows={1}
              />
              <button className="gen-send" onClick={sendMessage} disabled={loading || generating || !input.trim()}>→</button>
            </div>
          </div>
        </div>

        {/* ── PREVIEW ── */}
        <div className="gen-preview-panel">
          <div className="gen-panel-header">
            <div className="gen-preview-tabs">
              <button className={`gen-tab ${previewTab === "preview" ? "active" : ""}`} onClick={() => setPreviewTab("preview")}>👁 Перегляд</button>
              <button className={`gen-tab ${previewTab === "code" ? "active" : ""}`} onClick={() => setPreviewTab("code")}>{"</>"} Код</button>
            </div>
            <div className="gen-preview-actions">
              {previewCode && (
                <>
                  <button
                    className={`btn btn-sm ${editMode ? "btn-primary" : "btn-ghost"}`}
                    onClick={toggleEditMode}
                    title="Клікай на елементи щоб редагувати"
                  >
                    {editMode ? "✏ ON" : "✏ Редагувати"}
                  </button>
                  <button className="btn btn-ghost btn-sm" onClick={saveSite} title="Зберегти в акаунт">
                    💾 Зберегти
                  </button>
                  <button className="btn btn-ghost btn-sm" onClick={downloadSite} title="Завантажити HTML">
                    ⬇ HTML
                  </button>
                  <button className="btn btn-primary btn-sm" onClick={() => {
                    saveSite();
```

```
                    setMessages(m => [...m, { role: "assistant",
text: "🚀 Сайт опубліковано! Посилання: https://genui.app/s/my-
site-" + Date.now().toString().slice(-6) }]);
                  }}>
                    🚀 Опублікувати
                  </button>
                </>
              )}
              <button     className="btn     btn-ghost     btn-sm"
onClick={onBack}>← Назад</button>
            </div>
          </div>

          {/* Edit element panel */}
          {editingEl && (
            <div className="edit-panel">
              <div  className="edit-panel-title">✏️  Редагування
елемента &lt;{editingEl.tag}&gt;</div>
              <textarea
                className="edit-panel-input"
                value={editingEl.text}
                onChange={e => setEditingEl(el => ({ ...el, text:
e.target.value }))}
                rows={3}
                autoFocus
              />
              <div className="edit-panel-actions">
                <button    className="btn    btn-primary    btn-sm"
onClick={applyEdit}>✓ Застосувати</button>
                <button    className="btn    btn-ghost    btn-sm"
onClick={() => setEditingEl(null)}>Скасувати</button>
              </div>
            </div>
          )}

          {editMode && !editingEl && previewCode && (
            <div  className="edit-hint">👆 Клікніть  на  будь-який
текст або кнопку у прев'ю щоб редагувати</div>
          )}

          <div className="gen-preview-body">
            {!previewCode ? (
              <div className="gen-empty">
                <div  className="gen-empty-icon">🎨</div>
                <div       className="gen-empty-title">Попередній
перегляд</div>
                <div  className="gen-empty-sub">
                  {isEditing
                    ?  "Цей  сайт  ще  не  має  збереженого  коду.
Опишіть зміни у чаті — Claude оновить його."
                    : "Введіть опис сайту у чаті зліва — Claude
згенерує його тут"}
```

```
                </div>
                <div      className="gen-powered">Powered      by
<strong>Claude Sonnet</strong> + MCP</div>
              </div>
            ) : previewTab === "preview" ? (
              <iframe
                ref={iframeRef}
                className="gen-iframe"
                srcDoc={previewCode}
                title="Preview"
                sandbox="allow-scripts allow-same-origin"
                onLoad={handleIframeLoad}
              />
            ) : (
              <pre className="gen-code">{previewCode}</pre>
            )}
          </div>
        </div>
      </div>
    </div>
  );
}
```