МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
_____
(підпис)
« » грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інформаційна технологія реалізації концепції лаконічного підходу у вебдизайні
здобувача групи ІН.м – 33 Дрижова Ростислава Євгеновича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ростислав ДРИЖОВ
(підпис)

Керівник
асистент, к.ф.-м.н. _____ Олександр ВЛАСЕНКО
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-33 Дрижова Ростислава Євгеновича

1. Тема роботи: Інформаційна технологія реалізації концепції лаконічного підходу у вебдизайні затверджую наказом по СумДУ від *«03» грудня 2024 року № 1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до «06» грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Проектування технології для підвищення надійності біометричної аутентифікації 3) Вибір програмних засобів та реалізація прототипу технології. 4) Реалізація демонстраційного додатку 5) Валідація технології та аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «19» серпня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	14.10.2024	Виконав
2	*Огляд сучасних концепцій та аналіз аналогів*	21.10.2024	Виконав
3	*Вибір програмних засобів та реалізація прототипу технології*	28.10.2024	Виконав
4	*Реалізація демонстраційного додатку*	11.11.2024	Виконав
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	25.11.2024	Виконав

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 62 стр., 19 рис., 4 додатки, 20 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки сучасний вебдизайн потребує ефективних методів створення інтерфейсів, що поєднують естетику, функціональність та продуктивність. Концепція лаконічного підходу дозволяє підвищити зручність використання вебресурсів, оптимізувати їх роботу та забезпечити відповідність вимогам користувачів.

Об’єкт дослідження — процес створення вебсайтів.

Предмет дослідження — інформаційна технологія генерації вебресурсів із мінімалістичним інтерфейсом.

Мета роботи — розробка інформаційної технології, що забезпечує генерацію шаблонів сайтів за заданими параметрами, реалізуючи концепцію лаконічного дизайну.

Методи дослідження — використання методів структурного та об’єктно-орієнтованого проєктування, прототипування, а також бібліотек для фронтенд-розробки (React.js, Tailwind CSS). Застосовано принципи мінімалізму та доступності.

Результати — Розроблено інформаційну технологію, яка забезпечує створення вебресурсів на основі лаконічного підходу. Реалізовано прототип системи, що відповідає вимогам продуктивності та доступності. Впроваджено механізми оптимізації взаємодії користувача з інтерфейсом, мінімізуючи надмірність елементів і акцентуючи увагу на ключовій функціональності.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ВЕБДИЗАЙН, АДАПТИВНИЙ ІНТЕРФЕЙС, REACT.JS, ПРОДУКТИВНІСТЬ, WCAG.

ЗМІСТ

ВСТУП

Одним із трендів сучаного вебдизайну є зменьшення інформаціного навантаження на користувача, що реалізується завдяки стороннб прости, мінмалістичних інтерфесів. Концепція лаконічного підходу не лише полегшує сприйняття інформації користувачем, але й сприяє оптимізації продуктивності вебзастосунків, забезпечуючи швидке завантаження сторінок та адаптивність до різних пристроїв. Проте, автоматизація процесу створення мінімалістичних вебінтерфейсів залишається викликом, що потребує інноваційних рішень.

Об’єкт дослідження — процес створення вебсайтів.

Предмет дослідження — інформаційна технологія генерації вебресурсів із мінімалістичним інтерфейсом.

Мета роботи — розробка інформаційної технології, що забезпечує генерацію шаблонів сайтів за заданими параметрами, реалізуючи концепцію лаконічного дизайну.

Для досягнення поставної мети сформульовано наступні задачі роботи:

1. Провести дослідження інструментів, засобів та дизайнерських підходів, що можна поєднати загальною парадигмою мінімалістичного дизайну.
2. Спроектувати систему генерації шаблонів сайтів на основі користувацького запиту.
3. Імплементувати інформаційну технологію у вигляді веб-сервісу.
4. Провести тестування розробленого рішення.

Методи дослідження — використання структурного та об’єктно-орієнтованого проєктування, прототипування, а також бібліотек для фронтенд-розробки (React.js, Tailwind CSS). Застосовано принципи

мінімалізму та доступності.

Новизна дослідження полягає у створенні інформаційної технології, яка автоматизує процес реалізації концепції лаконічного підходу у вебдизайні, використовуючи сучасні мовні моделі.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, розробки інформаційної технології для реалізації концепції лаконічного підходу у вебдизайні, опису програмного забезпечення розробленого вебзастосунку, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Сучасний розвиток вебдизайну характеризується активним переходом до мінімалізму, що стає домінуючою тенденцією у створенні цифрових продуктів.

По-перше, лаконічний підхід, орієнтований на простоту, функціональність і естетику, дозволяє досягти ефективного користувацького досвіду за рахунок зменшення візуального шуму (рис. 1.1) та підвищення уваги до ключових елементів.

По-друге, такий підхід не лише відповідає сучасним вимогам користувачів, але й сприяє оптимізації продуктивності вебзастосунків, забезпечуючи швидке завантаження сторінок, адаптивність до різних пристроїв та дотримання стандартів доступності.

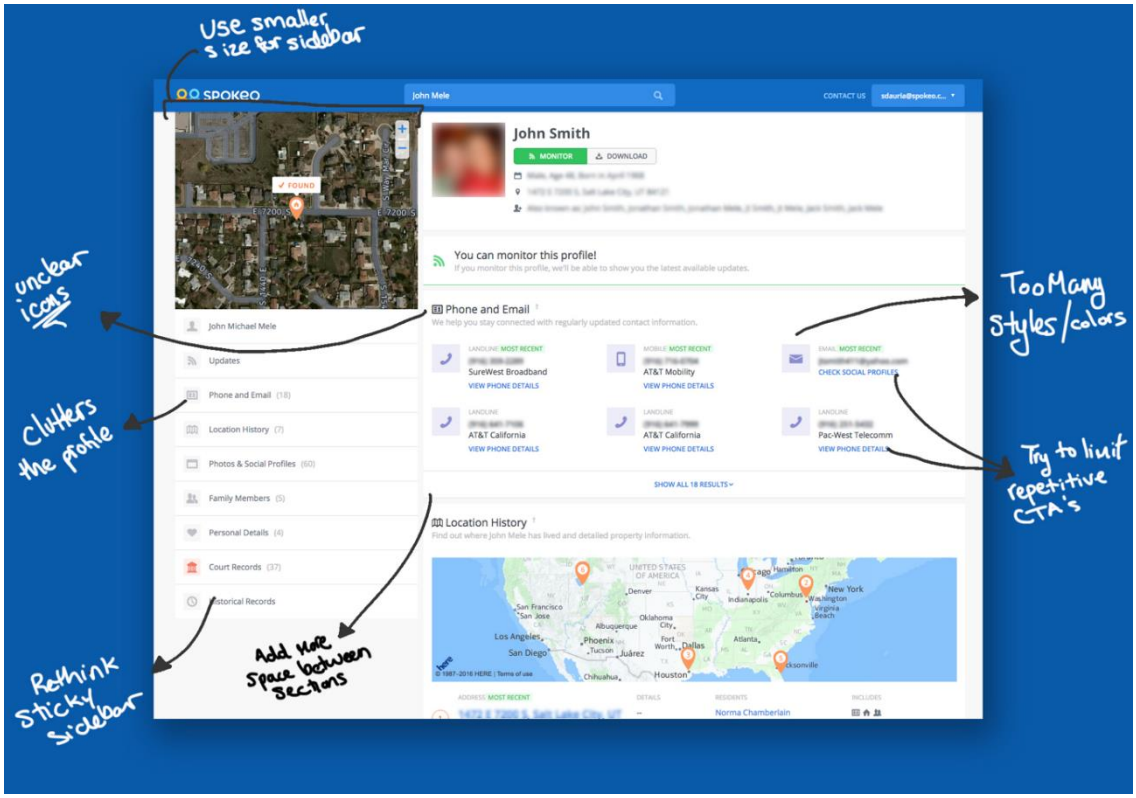

Рисунок 1.1 – Візуальний шум на прикладі сайту Spokeo

По-третє, автоматизація створення мінімалістичних вебінтерфейсів залишається складним завданням, особливо з урахуванням потреб користувачів різних категорій.

Таким чином, актуальність роботи визначається необхідністю розробки автоматизованих рішень для реалізації концепції лаконічного вебдизайну, які б задовольняли сучасні стандарти та підвищували ефективність процесу розробки вебзастосунків.

1.2 Аналіз аналогічних проєктів

Розробка вебзастосунків для автоматизації створення інтерфейсів із використанням принципів лаконічного дизайну є одним із сучасних напрямів у галузі вебтехнологій. У ході аналізу аналогічних проєктів було розглянуто кілька популярних інструментів і платформ, які пропонують автоматизовані рішення для вебдизайну.

1.2.1 Wix ADI (Artificial Design Intelligence)

Одним із таких інструментів є Wix ADI (Artificial Design Intelligence) (рис. 1.2), який використовує алгоритми штучного інтелекту для автоматичного створення вебсайтів. Система дозволяє враховувати базові вимоги користувача, такі як тип бізнесу, кольорова палітра та необхідні функціональні блоки. Однак, обмеженнями цієї платформи є недостатня гнучкість у кастомізації інтерфейсів і недостатній акцент на мінімалістичному дизайні.

1.2.2 Squarespace

Іншим популярним проєктом є Squarespace (рис. 1.3), який забезпечує високу якість дизайну шаблонів із сильним акцентом на естетику. Проте його функціональність здебільшого зосереджена на шаблонному підході,

що робить неможливою автоматичну генерації дизайну на основі специфічних вимог користувача.

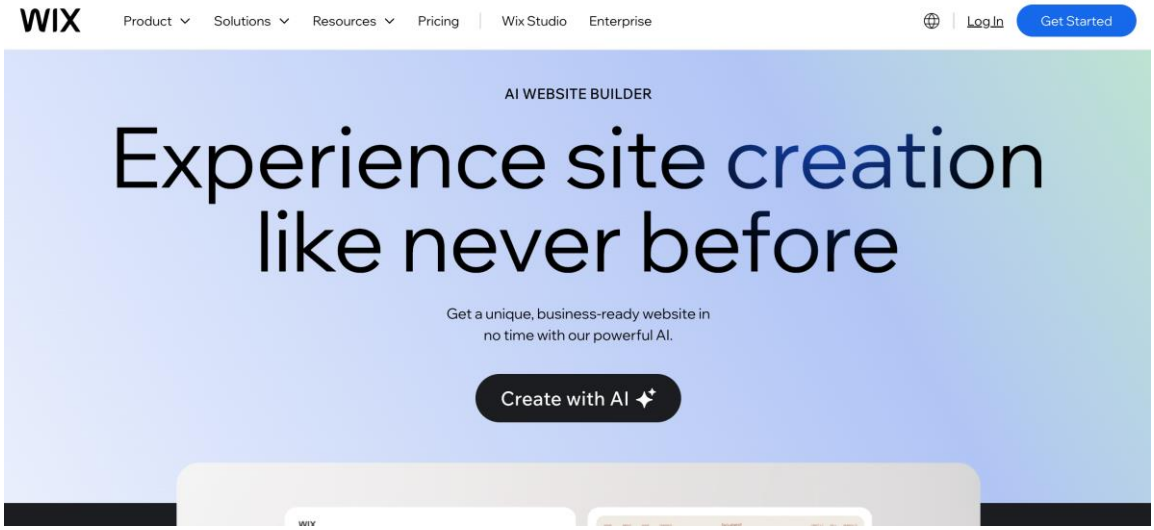

Рисунок 1.2 – Головна сторінка Wix ADI

Рисунок 1.3 – Головна сторінка Squarespace

1.2.3 Webflow

Webflow (рис. 1.4) є ще одним прикладом, який дозволяє створювати адаптивні інтерфейси без кодування. Його головною перевагою є повний контроль над дизайном та інтерактивністю.

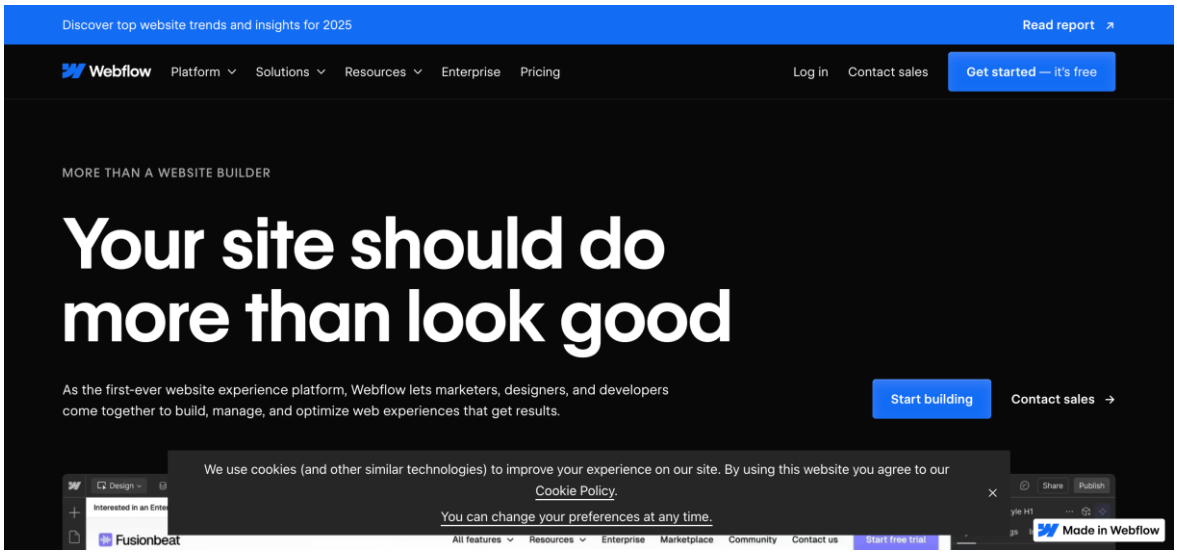

Рисунок 1.4 – Головна сторінка Webflow

Але, незважаючи на широкий спектр можливостей, Webflow вимагає від користувачів певних знань про дизайн і принципи побудови інтерфейсів, що зменшує його привабливість для непідготовленої аудиторії.

Таблиця 1.1 – Порівняльна таблиця для реалізації аналізу

Характеристика / Сайт	Wix ADI	Squarespace	Webflow
Сучасний дизайн	+	+	+
Зручний інтерфейс	+	+	-
Доступність (WCAG 2.1 AA)	-	-	+
Генерація коду	-	-	+
Реальний час перегляду	+	+	+
Адаптивність	+	+	+
Експорт дизайну	-	-	+
Персоналізація (шрифти, кольори, CTA)	+	+	+

Аналіз показує, що існуючі рішення здебільшого пропонують шаблонний підхід або вимагають значної участі користувача у створенні дизайну. Жодна з проаналізованих систем не забезпечує автоматичної генерації лаконічних вебінтерфейсів на основі детальних вимог користувача із застосуванням мовних моделей GPT. Це вказує на перспективність розробки інноваційного рішення, яке поєднує штучний інтелект, автоматизацію та принципи мінімалізму у вебдизайні.

1.3 Постановка задачі

Основною задачею даного дослідження є розробка інформаційної технології, яка дозволяє автоматизувати процес створення лаконічних вебінтерфейсів, дотримуючись принципів мінімалізму та високих стандартів доступності. Зокрема, завданням є:

- Розробити систему, що на основі введених користувачем вимог (демографія цільової аудиторії, тип вебсайту, основні розділи, колірна схема) автоматично генерує HTML5 структуру та CSS стильові таблиці, які відповідають принципам макету лаконічного дизайну.

- Створити систему, яка забезпечує генерацію адаптивних, функціональних інтерфейсів UI макетів, орієнтуючись на мобільні та десктопні версії одночасно.

- Реалізувати систему, що надає користувачам можливість вносити індивідуальні налаштування у дизайн (коригування кольорів) та отримувати відразу готовий код у форматах HTML/CSS.

Основною метою проекту є спрощення процесу створення вебсайтів, зменшення час та ресурсів, необхідні для розробки макету вебінтерфейсів.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір мов програмування

Для виконання поставлених задач було обрано наступні технології та мови програмування: HTML та CSS, JavaScript/TypeScript, з іншого боку серверна частина була підключена, за допомогою онлайн-сервісу Hugging Face.

2.1.1 HTML – Hyper Text Markup Language

HTML (HyperText Markup Language) — це стандартна мова розмітки, що використовується для створення та структурування вебсторінок і вебзастосунків. HTML визначає структуру вмісту вебсторінки за допомогою елементів, які представлені тегами. Ці теги описують різні частини вебсторінки, такі як заголовки, абзаци, зображення, гіперпосилання, списки та інші елементи. HTML працює як основа для вебконтенту, дозволяючи браузерам інтерпретувати та відображати дані у зрозумілому для користувача форматі.

Основою HTML є теги, які обмежують або обгортають різні частини контенту, визначаючи його роль або функціональність. Наприклад, тег <p> використовується для створення абзаців, <h1> – <h6> — для заголовків різних рівнів, а <img> – для вставки зображень. Теги можуть містити атрибути, які надають додаткову інформацію про елемент, наприклад, його розмір, ідентифікатор або стиль. HTML також підтримує вкладені структури, дозволяючи створювати складні документи, де елементи можуть містити інші елементи для забезпечення чіткої ієрархії вмісту.

Сучасний HTML, зокрема версія HTML5, надає нові можливості для розробників. Він включає в себе семантичні теги, такі як <header>, <nav>, <section> і <footer>, які допомагають краще структурувати сторінку та

полегшують доступність для користувачів і пошукових систем. Крім того, HTML5 забезпечує інтеграцію мультимедійного контенту через теги <audio> і <video>, що дозволяє відтворювати медіафайли без використання додаткових плагінів. Він також підтримує інтерактивність завдяки формам, валідації даних та можливості поєднання з іншими технологіями, такими як CSS та JavaScript.

Загалом, HTML є фундаментальною технологією для розробки вебінтерфейсів, яка забезпечує базову структуру будь-якої вебсторінки. Без нього неможливо створити вебресурс, оскільки саме HTML відповідає за розташування елементів на сторінці та їхню взаємодію з користувачем. Завдяки своїй простоті, гнучкості та широкому застосуванню HTML став основою для всієї веброзробки, підтримуючи як статичний, так і динамічний контент у поєднанні з іншими мовами програмування та технологіями.

2.1.2 CSS - Cascading Style Sheets

CSS (Cascading Style Sheets) — це мова стилів, яка використовується для опису зовнішнього вигляду та форматування HTML-документів. Вона дозволяє задавати стилі для елементів вебсторінки, таких як колір тексту, шрифти, розташування, розмір елементів, відступи та інші аспекти дизайну. CSS відокремлює структуру вмісту (HTML) від його оформлення, що дозволяє веброзробникам створювати чистий і зручний для підтримки код. Завдяки CSS можна легко налаштовувати дизайн вебсторінки та забезпечувати її адаптивність для різних пристроїв.

Основною перевагою CSS є можливість каскадування. Це означає, що стилі можна визначати на різних рівнях — у зовнішніх файлах, у внутрішніх блоках стилів або безпосередньо в HTML-елементах за допомогою атрибута style. При конфлікті стилів браузер визначає пріоритет, враховуючи специфічність та ієрархію правил. Каскадність дозволяє

розробникам централізовано керувати стилями всього проєкту, зменшуючи дублювання коду та спрощуючи його оновлення.

CSS підтримує різні селектори, які дозволяють застосовувати стилі до конкретних елементів на сторінці. Селектори можуть бути простими, наприклад, для тегів або класів (p, .header), або складними, використовуючи вкладеність, ідентифікатори та псевдокласи. Крім того, CSS дає можливість створювати ефективні дизайни за допомогою таких концепцій, як Flexbox та Grid. Ці технології дозволяють легко розміщувати елементи на сторінці, забезпечуючи їх гнучке та адаптивне відображення для різних розмірів екранів.

Завдяки сучасним можливостям CSS, веброзробники можуть створювати складні анімації, переходи та візуальні ефекти без використання JavaScript або інших мов програмування. Використання властивостей, як-от transform, transition та animation, дозволяє оживляти сторінки, роблячи їх привабливішими та інтерактивними. Такі технології, як медіа-запити (Media Queries), дають змогу адаптувати вебсайти під різні пристрої та роздільну здатність екранів, забезпечуючи сучасний адаптивний дизайн.

CSS також підтримує концепцію змінних (CSS Variables), що дозволяє задавати значення кольорів, шрифтів або інших властивостей глобально та використовувати їх у всьому документі. Це робить код більш гнучким і полегшує внесення змін у проєкті. З розвитком CSS препроцесорів, таких як SASS і LESS, можливості мови розширились ще більше: вони дозволяють використовувати функції, вкладеність, міксіни та інші інструменти для підвищення продуктивності розробки.

Таким чином, CSS є невід'ємною частиною сучасної веброзробки. Завдяки своїм потужним інструментам та можливостям, вона дозволяє створювати красиві, адаптивні та функціональні вебсайти. CSS дає змогу

відокремлювати контент від дизайну, що покращує гнучкість, швидкість розробки та підтримку проєктів різної складності.

2.1.3 JavaScript/TypeScript

JavaScript – це високорівнева мова програмування, що є однією з основних технологій для створення сучасних вебсайтів і вебдодатків. Вона використовується для додавання інтерактивності на вебсторінки, таких як анімації, форми з валідацією, динамічне оновлення вмісту без перезавантаження сторінки та обробка подій користувача. JavaScript працює у браузерах завдяки вбудованому інтерпретатору, що дозволяє виконувати код безпосередньо на стороні клієнта. Останнім часом мова активно використовується і на сервері, завдяки платформі Node.js, що розширює можливості для розробників.

TypeScript, у свою чергу, є надмножиною JavaScript, що додає підтримку статичної типізації. Це означає, що розробники можуть визначати типи змінних, функцій та об'єктів під час написання коду, що допомагає уникнути багатьох помилок ще на етапі розробки. TypeScript був розроблений компанією Microsoft і став популярним завдяки своїй здатності робити великі та складні проєкти більш структурованими й зрозумілими. Важливо зазначити, що TypeScript компілюється в звичайний JavaScript, тому його можна використовувати будь-де, де працює JavaScript.

Одна з ключових переваг JavaScript полягає в його гнучкості та підтримці величезною кількістю бібліотек і фреймворків, таких як React, Angular та Vue.js, які полегшують створення сучасних вебінтерфейсів. Крім того, з появою TypeScript розробники отримали можливість краще контролювати код, завдяки чіткій типізації та інструментам, що покращують автодоповнення й рефакторинг. Це особливо важливо для великих команд, які працюють над масштабними програмними продуктами.

Загалом, JavaScript і TypeScript є важливими складовими сучасної веброзробки. JavaScript відповідає за швидку інтерактивність та динамічний контент, а TypeScript робить процес розробки більш передбачуваним і менш схильним до помилок. Використовуючи їх разом, можна створювати надійні та продуктивні додатки для вебу, серверів і навіть мобільних пристроїв.

2.1.4 Онлайн сервіс Hugging Face

Hugging Face – це популярний онлайн-сервіс та платформа, яка стала одним із ключових гравців у сфері машинного навчання та обробки природної мови (NLP). Спершу Hugging Face позиціонувався як компанія, що створює чатботи, але згодом переріс у глобальний центр для роботи з моделями штучного інтелекту, особливо для розробників і дослідників. Сьогодні платформа відома завдяки своїм інструментам, бібліотекам та API для розгортання й використання моделей машинного навчання у зручний спосіб.

Основою Hugging Face є бібліотека Transformers, яка містить велику кількість готових до використання моделей для задач обробки мови, таких як GPT, BERT, T5 та інші. Ці моделі є відкритими та оптимізованими для різноманітних завдань: генерація тексту, класифікація, аналіз настроїв, переклад, а також розпізнавання мовлення. Hugging Face дозволяє швидко інтегрувати потужні алгоритми у свої проєкти завдяки простому та зручному API, що підходить як для початківців, так і для досвідчених інженерів.

Ще однією перевагою Hugging Face є Model Hub – це репозиторій, де зберігаються тисячі моделей машинного навчання, які можна завантажити, адаптувати під свої потреби чи одразу використовувати онлайн через API. Користувачі можуть завантажувати власні моделі або працювати з уже готовими рішеннями, що спрощує розробку інтелектуальних систем та

економить час на навчання моделей з нуля. Hugging Face активно підтримує спільноту, де розробники діляться досвідом та готовими рішеннями для розв'язання завдань.

Для інтеграції в реальні продукти Hugging Face пропонує Inference API, що дозволяє отримати доступ до моделей через хмарний сервіс. Це ідеальний інструмент для розробників, яким потрібно швидко впровадити штучний інтелект у вебдодатки, мобільні платформи чи SaaS-рішення. Завдяки API, запити обробляються онлайн, без потреби у значних обчислювальних ресурсах на стороні користувача. Це робить Hugging Face важливим інструментом для створення інтелектуальних, аналітичних та швидких рішень у сфері вебдизайну, обробки даних і взаємодії з користувачами.

Загалом Hugging Face є потужним сервісом, що надає розробникам доступ до сучасних моделей штучного інтелекту, спрощує роботу з NLP і допомагає швидко інтегрувати інноваційні рішення. Завдяки відкритості, доступності та підтримці спільноти платформа продовжує стимулювати розвиток технологій і допомагає розробляти ефективні системи для різних індустрій.

2.2 Вибір фреймворків

У процесі розробки веб-додатків, особливо в контексті лаконічного вебдизайну та використання інтелектуальних технологій, важливим етапом є вибір фреймворків. Фреймворки забезпечують основу для ефективного створення структурованого, продуктивного та адаптивного вебрішення. Вони допомагають розробникам зосередитися на логіці та функціоналі, зменшуючи необхідність писати код з нуля. У цьому підпункті буде проаналізовано кілька сучасних фреймворків, які підходять для реалізації мінімалістичного підходу у вебдизайні.

2.2.1 React

React – це популярна JavaScript-бібліотека для створення інтерфейсів користувача, розроблена Facebook. React орієнтований на компонентний підхід, що дозволяє будувати вебсторінки як набір незалежних компонентів. Це особливо корисно для мінімалістичного дизайну, де структура інтерфейсу має бути чистою та зрозумілою. React забезпечує високу продуктивність завдяки використанню віртуального DOM, що дозволяє ефективно оновлювати елементи сторінки без перезавантаження.

Приклад використання:

```
import React from "react";

function Header() {
  return (
    <header className="bg-gray-100 p-4">
      <h1 className="text-2xl font-bold">Welcome to
Minimalist UI</h1>
    </header>
  );
}

export default Header;
```

Цей компонент створює заголовок з мінімалістичним дизайном, використовуючи класи Tailwind CSS.

У межах проєкту React можна використовувати для створення інтерфейсу з динамічним рендерингом контенту, який генерується через API Hugging Face. Серед ключових переваг React – його простота, надійність і підтримка великою спільнотою. Додаткові інструменти, такі як React Router (для навігації) та React Hooks, дозволяють покращити функціональність програми та зберегти її код зрозумілим.

2.2.2 Vue.js

Vue.js – ще один сучасний фреймворк, який підходить для розробки інтерфейсів з акцентом на простоту та мінімалізм. Vue.js поєднує переваги React і Angular, пропонуючи легкий синтаксис і високу продуктивність. Цей фреймворк ідеально підходить для швидкого створення прототипів вебзастосунків, оскільки має низький поріг входу.

Приклад використання:

```
<template>
  <div class="bg-gray-100 p-4">
    <h1 class="text-2xl font-bold">{{ title }}</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
      title: "Welcome to Minimalist UI",
    };
  },
};
</script>
```

У цьому прикладі Vue.js використовується для динамічного оновлення тексту заголовка за допомогою двостороннього зв'язування даних.

Для реалізації лаконічного підходу у вебдизайні Vue.js можна використовувати для побудови простих, але функціональних інтерфейсів. Двостороннє зв'язування даних та компонентна архітектура дозволяють швидко оновлювати UI на основі API-відповідей. Крім того, Vue легко інтегрується у вже наявні проєкти, що робить його зручним для поступового впровадження.

2.2.3 Tailwind CSS

Tailwind CSS є CSS-фреймворком, орієнтованим на utility-first підхід, що дозволяє створювати адаптивний дизайн без написання користувацького CSS. Це ідеальний інструмент для реалізації мінімалістичного вебдизайну, оскільки він надає готові класи для швидкого стилізування компонентів. Tailwind дозволяє чітко контролювати відступи, розміри шрифтів та палітру кольорів, що відповідає принципам лаконічності та чистоти інтерфейсу.

Приклад використання:

```
<div class="bg-white max-w-md mx-auto p-6 rounded-lg
shadow-lg">
  <h1 class="text-2xl font-bold text-gray-900">Minimalist
Card</h1>
  <p class="text-gray-600 mt-2">This is a simple card
layout using Tailwind CSS.</p>
</div>
```

Цей приклад демонструє створення мінімалістичної картки з використанням утиліт Tailwind CSS для відступів, кольорів і стилів.

У цьому проєкті Tailwind CSS може бути використаний для створення чистого та структурованого UI з чіткими відступами (за допомогою 8-point grid system), а також для забезпечення відповідності WCAG стандартам доступності. Готові класи дозволяють уникнути перевантаження кастомним CSS і покращують продуктивність розробки.

2.1.4 Next.js

Next.js – фреймворк для React, що додає можливість серверного рендерингу (SSR) та генерації статичних сторінок (SSG). Це дозволяє створювати продуктивні, оптимізовані та швидкі вебзастосунки. Next.js забезпечує краще SEO, завдяки швидкому завантаженню контенту та можливості lazy-loading для ресурсів.

Приклад використання:

```
import React from "react";

export async function getStaticProps() {
  return {
    props: { message: "Welcome to Minimalist UI with
Next.js" },
  };
}

function Home({ message }) {
  return (
    <div className="p-4">
      <h1 className="text-2xl font-bold">{message}</h1>
    </div>
  );
}

export default Home;
```

Цей приклад демонструє генерацію статичної сторінки, яка отримує дані під час побудови, забезпечуючи швидке завантаження контенту.

В контексті мінімалістичного вебдизайну Next.js допомагає оптимізувати швидкодійність сторінок і спрощує роботу з API. Для проєкту, який використовує Hugging Face API, Next.js може стати ефективним рішенням для інтеграції динамічних даних та їх виведення на сторінки без втрати продуктивності.

З огляду на вимоги до магістерської роботи та концепцію лаконічного підходу у вебдизайні, React у поєднанні з Tailwind CSS є найкращим вибором. React забезпечує компонентний підхід та інтеграцію API Hugging Face, тоді як Tailwind CSS допомагає створити чистий, мінімалістичний дизайн без зайвих стилів. У разі потреби в серверному рендерингу, використання Next.js буде додатковою перевагою для оптимізації продуктивності.

Таким чином, комбінація цих технологій дозволить ефективно реалізувати вимоги до проєкту, зберігаючи продуктивність, доступність та лінійність дизайну.

2.3 Вибір IDE

Вибір інтегрованого середовища розробки (IDE) є важливим етапом у розробці вебзастосунків, оскільки від цього залежить зручність, ефективність роботи та якість кінцевого продукту. На сонові пошуку та порівняння конкурентних переваг таких засобів як JetBrains WebStorm, Mickrosoft Vs Code та ряду інших засобів було прийняте рішення обрати StackBlitz, як найбільш відповідну меті та задачам роботи.

StackBlitz — це онлайн-платформа для розробки, яка дозволяє створювати та запускати вебзастосунки прямо в браузері без необхідності встановлювати додаткове програмне забезпечення. Вона підтримує популярні веб-технології, такі як JavaScript, TypeScript, Angular, React, Vue та інші. StackBlitz пропонує інтегроване середовище для розробки з автоматичним оновленням змін у реальному часі та можливістю запускати серверні та клієнтські проєкти прямо в браузері, що робить її чудовим інструментом для швидкої розробки та тестування вебзастосунків.

Особливістю StackBlitz є його здатність працювати з проектами на основі Node.js та інших сучасних фреймворків, без потреби локальної установки або налаштування середовища. Платформа підтримує інтеграцію з GitHub для синхронізації коду та забезпечує вбудований редактор з функціями автозавершення, перевірки коду, а також можливістю швидкої публікації проєктів. Вона активно використовує WebContainers — технологію, яка дозволяє запускати повноцінні сервери та розробницькі середовища безпосередньо в браузері, що забезпечує високу продуктивність і зручність.

StackBlitz є ідеальним вибором для розробки лаконічних вебзастосунків, таких як описаний у магістерській роботі, завдяки своїй здатності працювати з передовими вебтехнологіями, доступності онлайн та простоті використання для створення та тестування інтерфейсів без необхідності у складних налаштуваннях середовища.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Розроблена комплексна інформаційна модель (див. рис 3.1) слугує фундаментальною основою для створення системи автоматизації розробки мінімалістичних вебінтерфейсів. Унікальність даної моделі полягає в її орієнтації на принципи лаконічного вебдизайну та глибокій інтеграції всіх компонентів системи для досягнення оптимального результату.

Архітектура моделі базується на п'яти ключових компонентах, кожен з яких відіграє важливу роль у загальному процесі автоматизації. Першим фундаментальним елементом виступає профіль користувача, який являє собою комплексний набір даних для визначення цільової аудиторії. Система збирає та аналізує не лише базові демографічні характеристики, такі як вік, геолокація та мовні преференції, але й враховує більш глибинні психографічні особливості користувачів. Особлива увага приділяється аналізу поведінкових патернів та очікувань від інтерфейсу, що дозволяє створювати максимально персоналізовані рішення. Важливим аспектом є також врахування специфіки бізнесу замовника, включаючи галузеву приналежність, масштаб діяльності та особливості цільової аудиторії.

Другий компонент моделі представлений системою вимог до дизайну, яка забезпечує тонке налаштування як візуальних, так і функціональних аспектів вебінтерфейсу. У рамках цього компоненту реалізовано комплексний підхід до формування кольорової палітри, що включає роботу з основними та акцентними кольорами, створення гармонійних градієнтів та налаштування світлових схем. Типографічна система забезпечує професійний підхід до роботи зі шрифтами, включаючи підбір шрифтових пар, налаштування масштабування та оптимізацію інтерліняжу для максимальної читабельності контенту.

Центральним елементом моделі виступає генератор макетів – високотехнологічний компонент, що використовує передові алгоритми машинного навчання для оптимізації композиції та забезпечення візуальної гармонії. Цей модуль відповідає за створення адаптивних макетів, які автоматично пристосовуються до різних пристроїв та розмірів екранів. Особлива увага приділяється оптимізації продуктивності згенерованого коду та забезпеченню відповідності сучасним стандартам доступності.

Четвертий компонент – система інтерактивності. Цей модуль дозволяє миттєво візуалізувати будь-які зміни в дизайні.

Завершальним елементом моделі є модуль експорту, який відповідає за генерацію готового до використання коду. Цей компонент не просто створює базовий HTML/CSS код, а забезпечує його оптимізацію відповідно до сучасних стандартів веброзробки. Додатково генеруються готові компоненти для популярних фреймворків, що значно спрощує процес інтеграції створеного інтерфейсу в існуючі проекти.

Процес роботи системи реалізується через послідовність взаємопов'язаних етапів, починаючи від збору та валідації даних і закінчуючи експортом готового продукту. На етапі збору даних система не лише акумулює інформацію про вимоги користувача, але й проводить аналіз конкурентного середовища та формує базові дизайн-патерни. Наступний етап включає комплексний аналіз параметрів з використанням евристичних алгоритмів та систем валідації.

Особливу увагу приділено етапу генерації макету, який базується на принципах atomic design та включає створення адаптивної сітки, налаштування типографіки та впровадження інтерактивних елементів. Тестування створеного інтерфейсу проводиться на різних пристроях з перевіркою доступності та швидкодії. Фінальний етап експорту супроводжується створенням детальної документації та рекомендацій щодо подальшої підтримки проекту.

Така глибока інтеграція всіх компонентів інформаційної моделі забезпечує створення якісних мінімалістичних вебінтерфейсів, які відповідають сучасним вимогам до дизайну та функціональності. Модель враховує як технічні аспекти розробки, так і естетичні вимоги до візуального оформлення, що дозволяє досягати оптимального балансу між формою та функціональністю створюваних інтерфейсів.

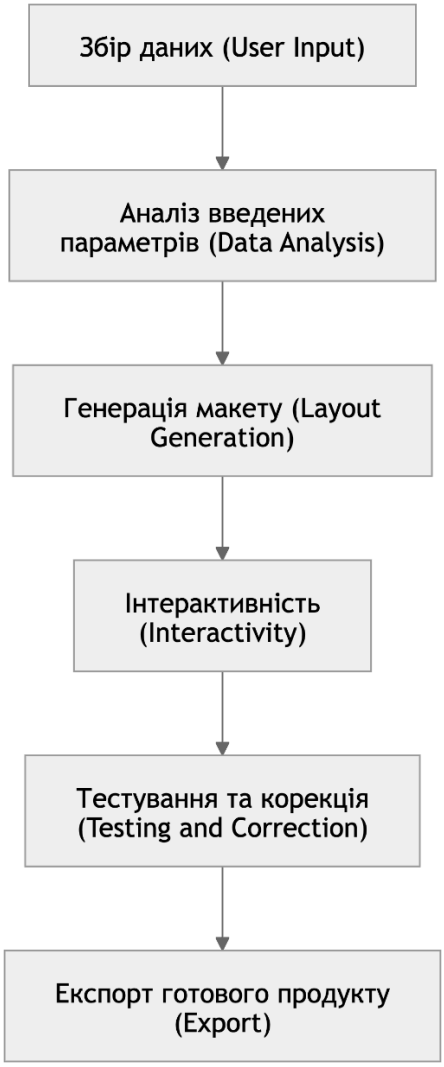

Рисунок 3.1 – Діаграма інформаційної моделі

Рисунок 3.2 – Use Case діаграма

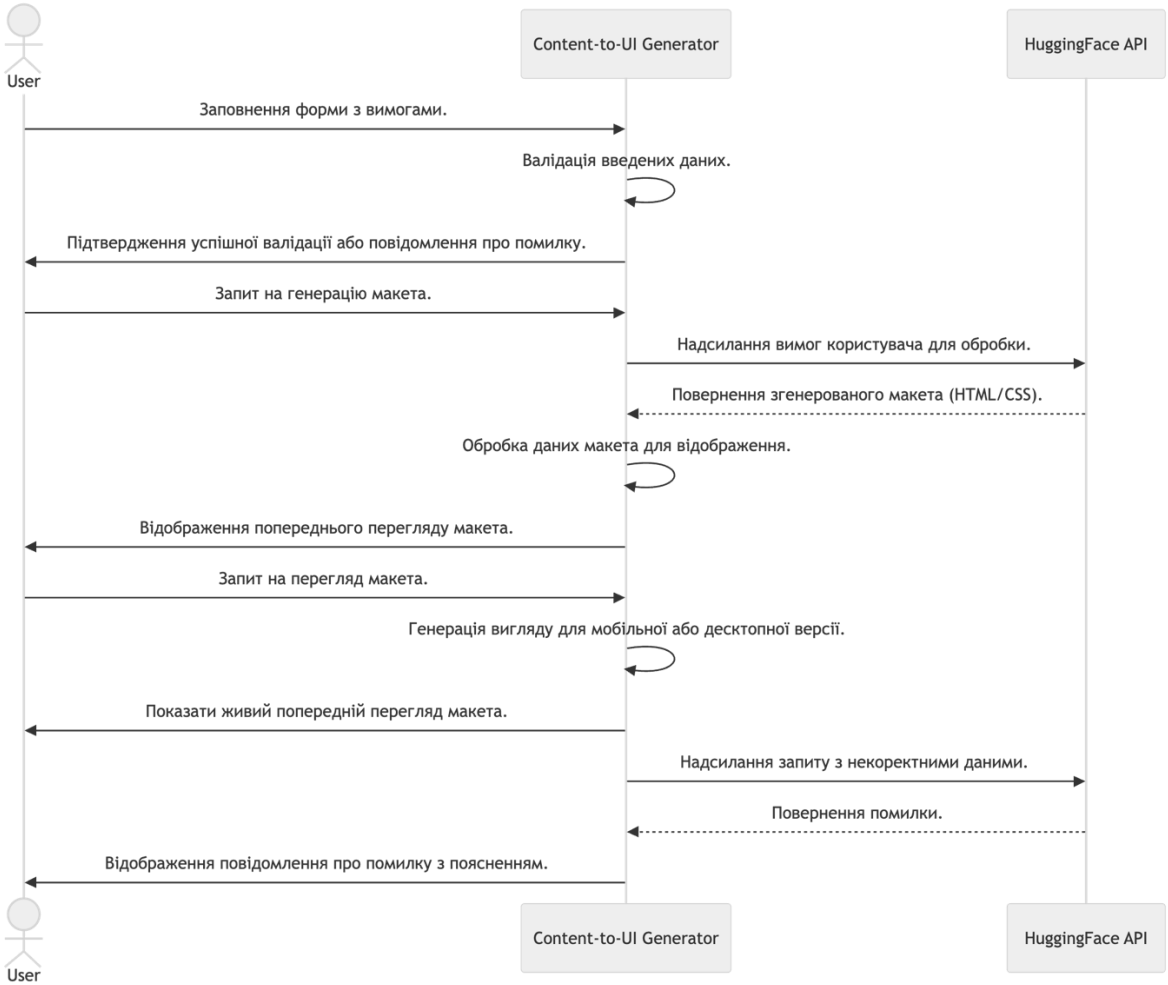

Рисунок 3.3 – Sequence UML діаграма

Інформаційна модель забезпечує логічну організацію компонентів та описує потоки даних між ними, що дозволяє мінімізувати час розробки та спрощує підтримку системи.

3.2 Програмна реалізація

На рисунку нижче зображено структуру файлів проєкту (Рисунок 3.4).

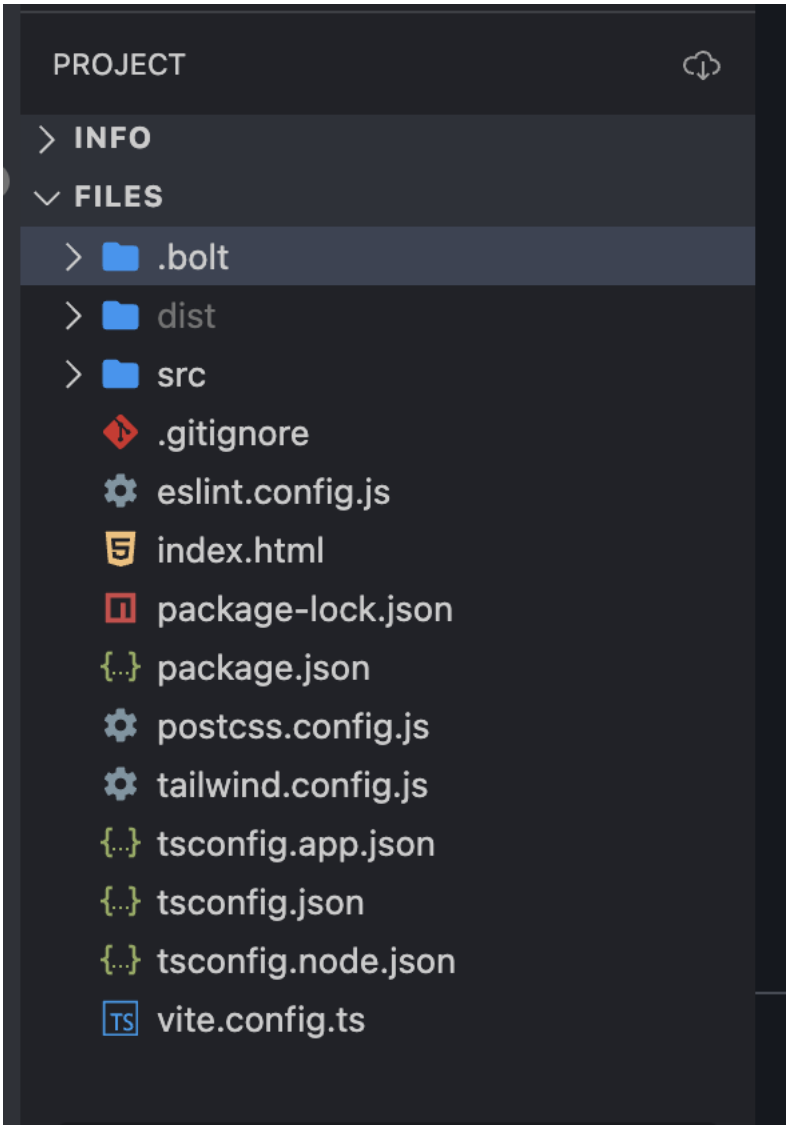

Рисунок 3.4 – Файлова структура проєкту

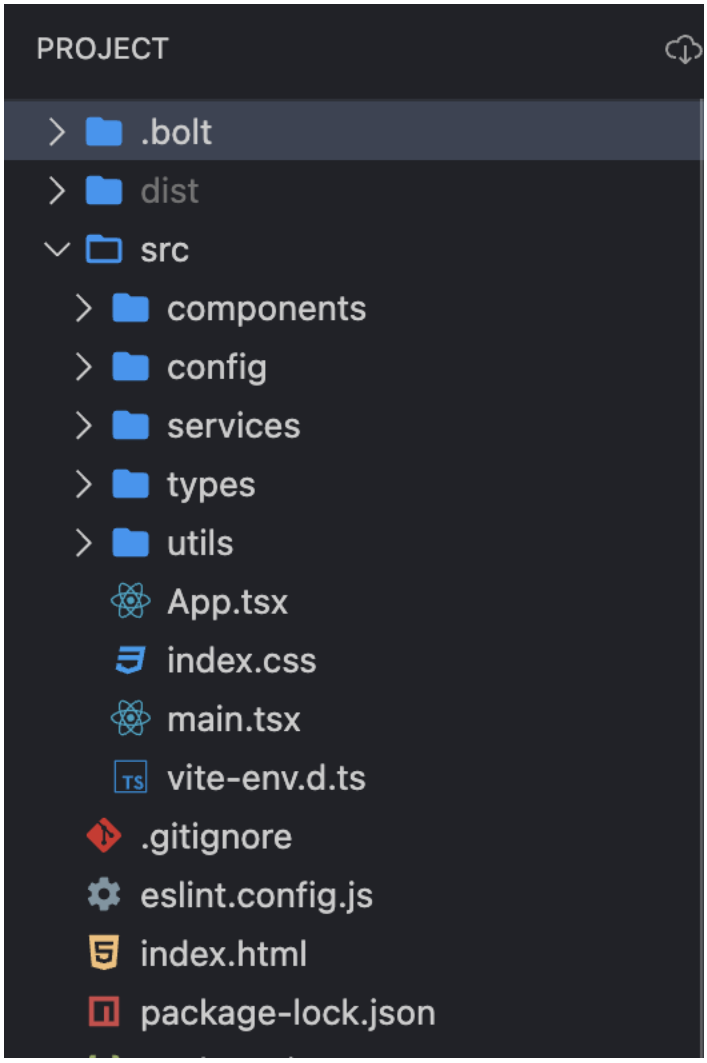

Рисунок 3.5 – Файлова структура папки src

Основним файлом додатку є App.tsx. В ньому визначений компонент App. Повний код App.js наведено в “Додаток А”.

Опис компонента App.tsx. Компонент App.tsx є головною частиною користувацького інтерфейсу веб-додатку Content-to-UI Generator, який відповідає за організацію основних етапів генерації макетів. Це компонента на основі React, яка управляє відображенням форм, попереднього перегляду макетів, взаємодією з користувачем та обробкою даних.

Основні частини компонента:

1. Стан додатку (state): layout: зберігає поточний згенерований макет (якщо він є). loading: фіксує стан завантаження під час генерації

макету. error: зберігає повідомлення про помилку, якщо вона виникає під час генерації макету. На рисунку 3.6 зображена ініціалізація стану.

```
export function App() {
  const [layout, setLayout] = React.useState<GeneratedLayout | null>(null);
  const [loading, setLoading] = React.useState(false);
  const [error, setError] = React.useState<string | null>(null);
```

Рисунок 3.6 – Ініціалізація стану

2. Метод handleSubmit (рис. 3.7). Цей метод обробляє подачу форми. Після того, як користувач надає свої вимоги (через InputForm), дані передаються до сервісу для генерації макету. Після успішної генерації макету: Зберігається в IndexedDB через layoutDB.saveLayout. Оновлюється стан layout, щоб відобразити новий макет. У разі помилки виводиться повідомлення про помилку.

```
const handleSubmit = async (requirements: UserRequirements) => {
  setLoading(true);
  setError(null);

  try {
    const generatedLayout = await generateLayout(requirements);
    await layoutDB.saveLayout(requirements, generatedLayout);
    setLayout(generatedLayout);
  } catch (err) {
    setError((err as Error).message);
  } finally {
    setLoading(false);
  }
};
```

Рисунок 3.7 – Зображення методу handleSubmit

3. Метод handleSelectLayout (рис. 3.8). Цей метод дозволяє вибирати збережені макети з історії. Якщо користувач вибирає попередньо збережений макет, він відображається на екрані.

Рисунок 3.8 – Зображення методу handleSelectLayout

4. Рендеринг компонентів. InputForm: компонент для введення вимог користувача, де здійснюється перевірка даних та передача на сервер для генерації макету. RecentLayouts: Компонент для відображення раніше збережених макетів. Користувач може вибрати один із попередніх макетів для редагування або перегляду. Preview: Компонент для попереднього перегляду згенерованого макету. Включає можливість перегляду в мобільному та десктопному вигляді.

5. UI структура. Шапка (header): відображає назву додатку та посилання на репозиторій GitHub. Основний контент (main): Містить форму введення, попередній перегляд макету та список попередніх макетів. Футер (footer): Інформація про створення додатку.

6. Керування завантаженням і помилками: якщо відбувається завантаження (під час генерації макету), користувач бачить анімацію завантаження. Якщо виникає помилка (наприклад, проблема при виклику API), виводиться відповідне повідомлення про помилку.

7. Інтерфейс та стиль: використовується Tailwind CSS для стилізації компонентів. Кожен компонент стилізовано для забезпечення чистого та привабливого дизайну, включаючи адаптивний дизайн для різних пристроїв.

Структура компонентів та їх взаємодія:

1. InputForm: Збирає вимоги користувача та передає їх у handleSubmit для генерації макету.

2. RecentLayouts: Відображає список попередніх макетів і дозволяє вибрати один для повторного використання.

3. Preview: Виводить згенерований макет або показує індикатор завантаження під час генерації.

Приклад роботи компонента: при натисканні кнопки “Generate Layout” в компоненті InputForm користувач вводить необхідні дані (наприклад, тип вебсайту, вимоги до дизайну). Після того як дані передаються у метод handleSubmit, відправляється запит до серверу для генерації макету. Отриманий результат виводиться в Preview компоненті для попереднього перегляду, а також зберігається в базі даних для подальшого використання.

Таким чином, App.tsx управляє основними аспектами взаємодії з користувачем, включаючи введення вимог, генерування макетів, їх перегляд та збереження історії макетів для повторного використання.

3.2 Тестування

Ручне тестування є важливим етапом процесу забезпечення якості програмного продукту, оскільки дозволяє перевірити поведінку додатку в реальних умовах.

Перевірка заповнення всіх полів. Розпочнемо з головної сторінки. Заповнюємо усі поля в формі: цільова аудиторія, мета вебсайту, розділи контенту, кольори бренду та натискаємо кнопку “Генерувати макет”. (Рис. 3.9):

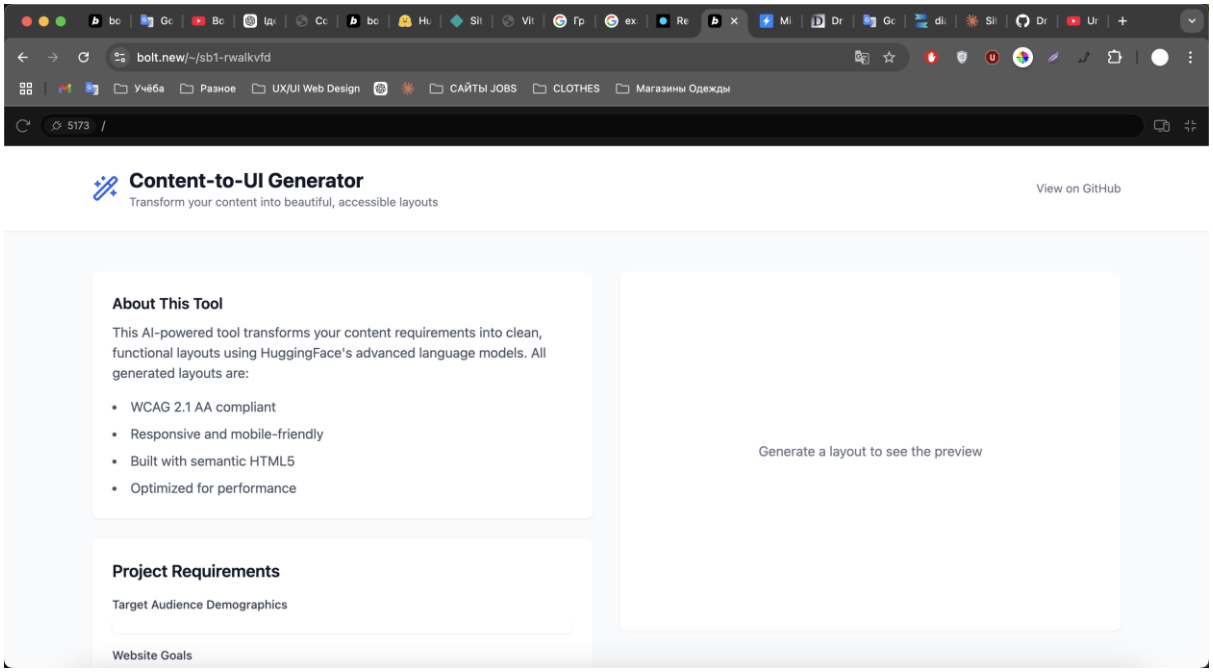

Рисунок 3.9 – Головна сторінка для внесення даних

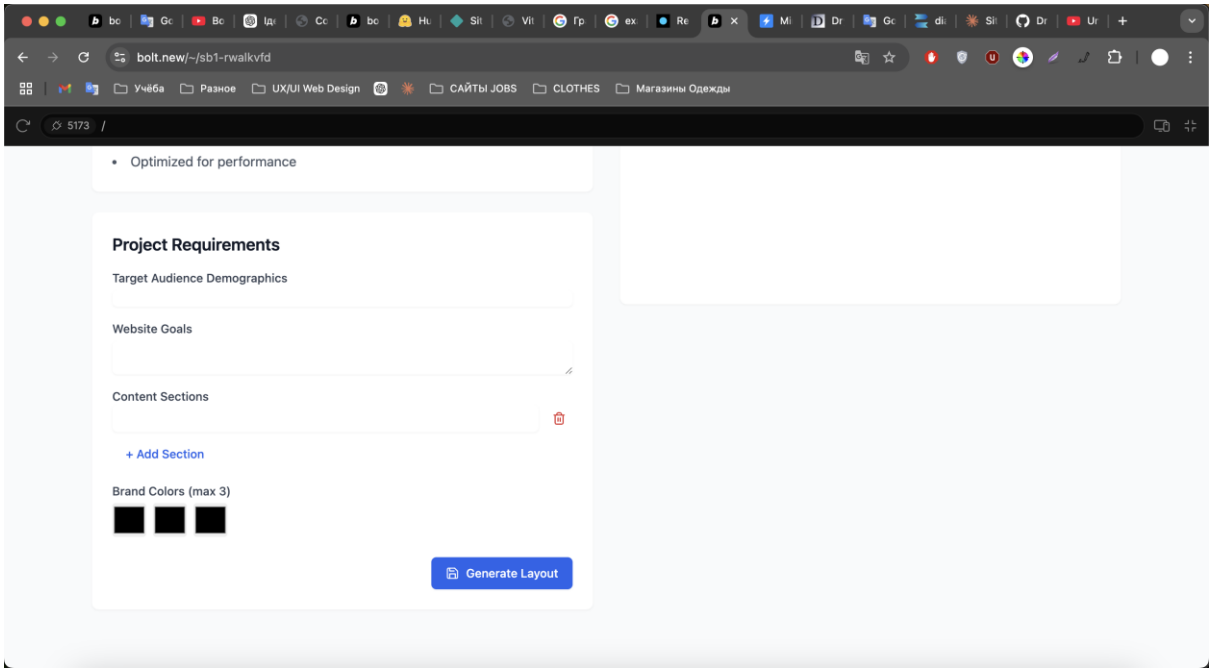

Рисунок 3.10 – Поля для вводу даних

При натискані кнопки генерувати макет запит з створенням базового макету виконаний успішно, про що інформує превью на рисунку 3.11.

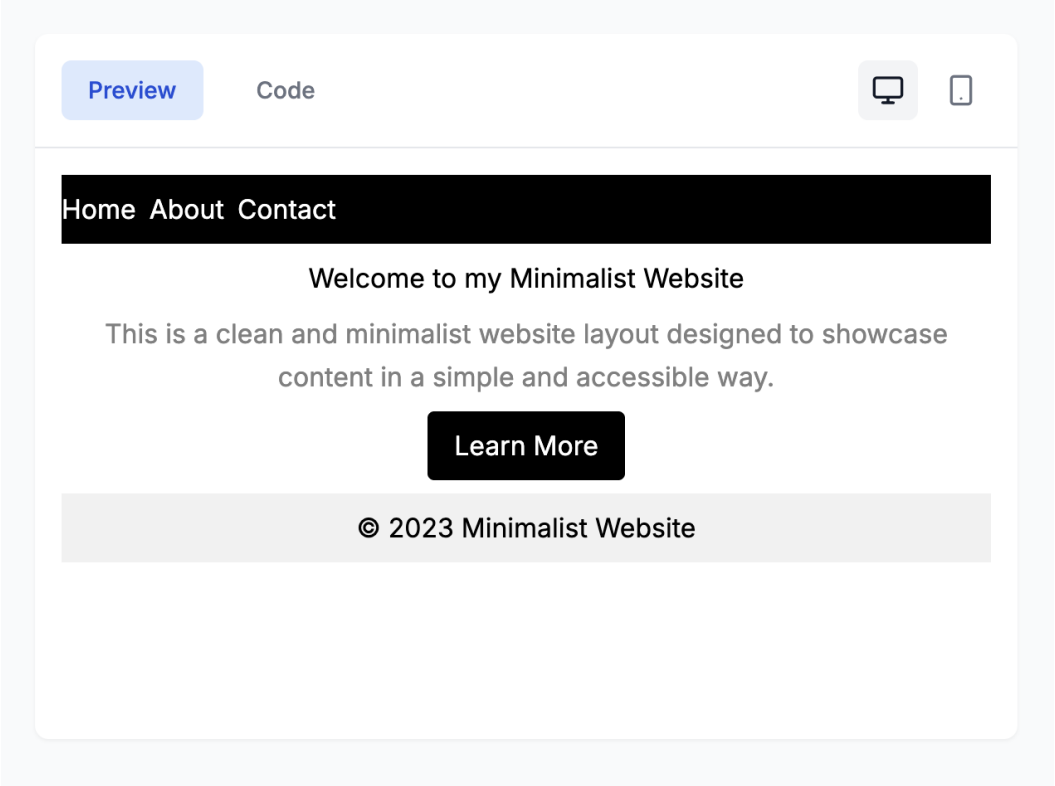

Рисунок 3.11 – Превью сгенерованого макету

Теж саме з макетом для мобільної версії. (Рис. 3.12):

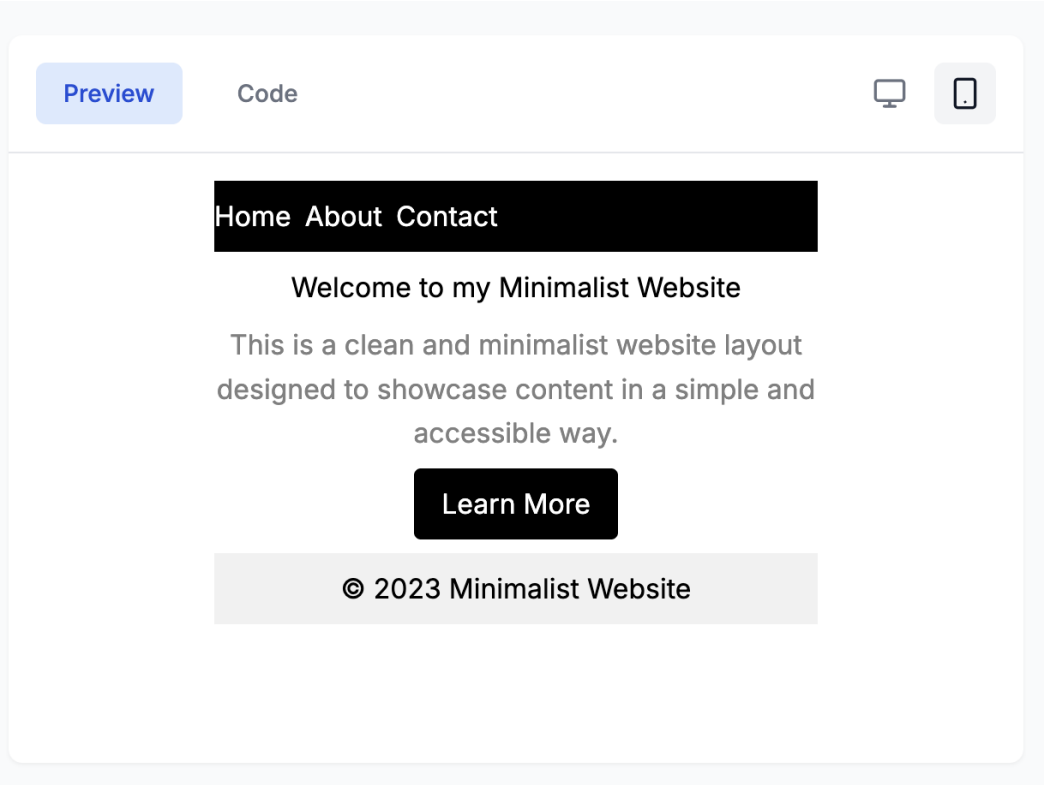

Рисунок 3.12 – Превью для мобільної версії

А також в доповнення вкладка коду на рисунку 3.13.

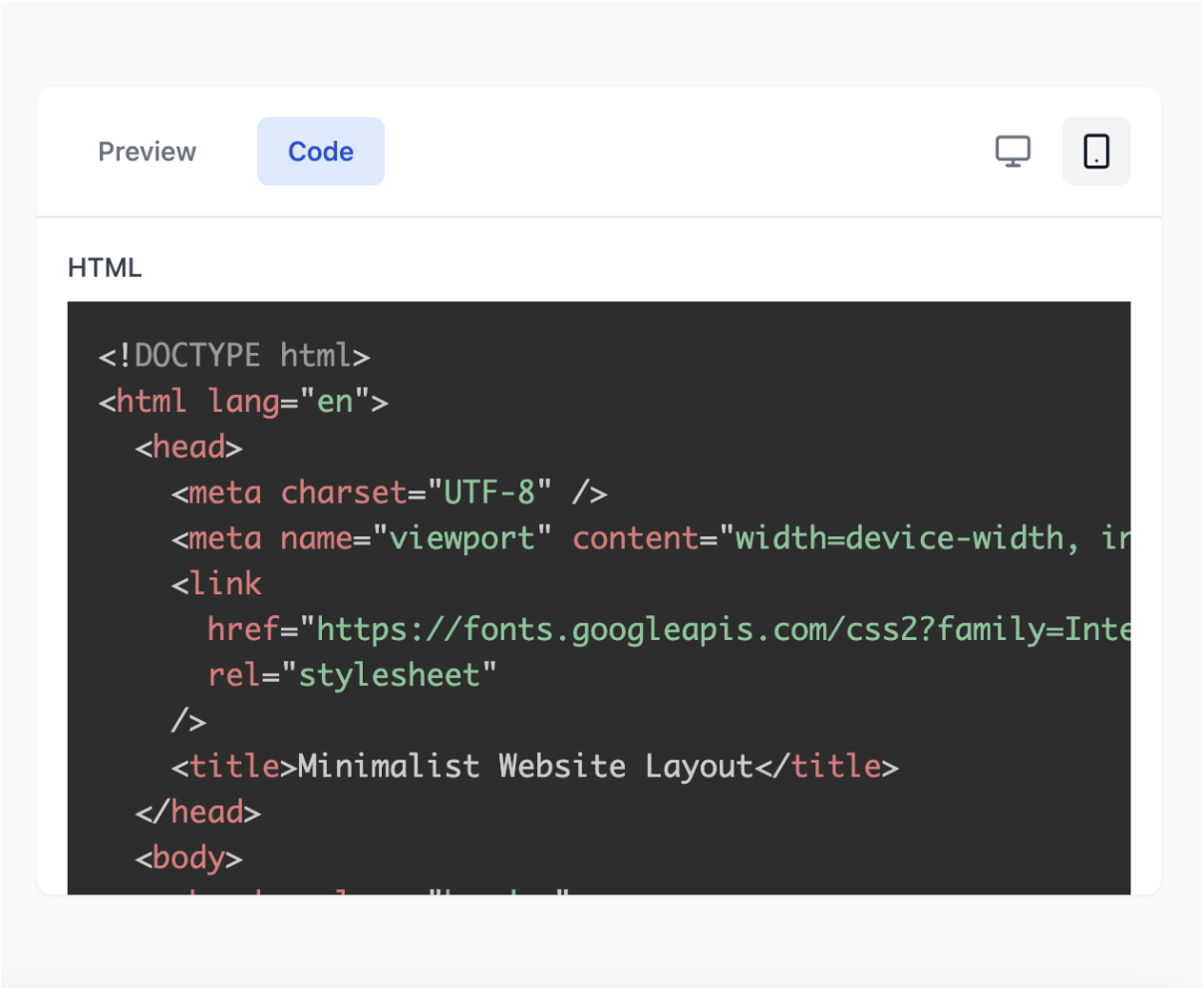

Рисунок 3.13 – Code для копіювання

Тепер давайте перейдемо до перевірки некоректного вводу (порожні поля). Відкриваємо веб-додаток та залишаємо поля порожніми, після цього натискаємо кнопну генерувати макет.

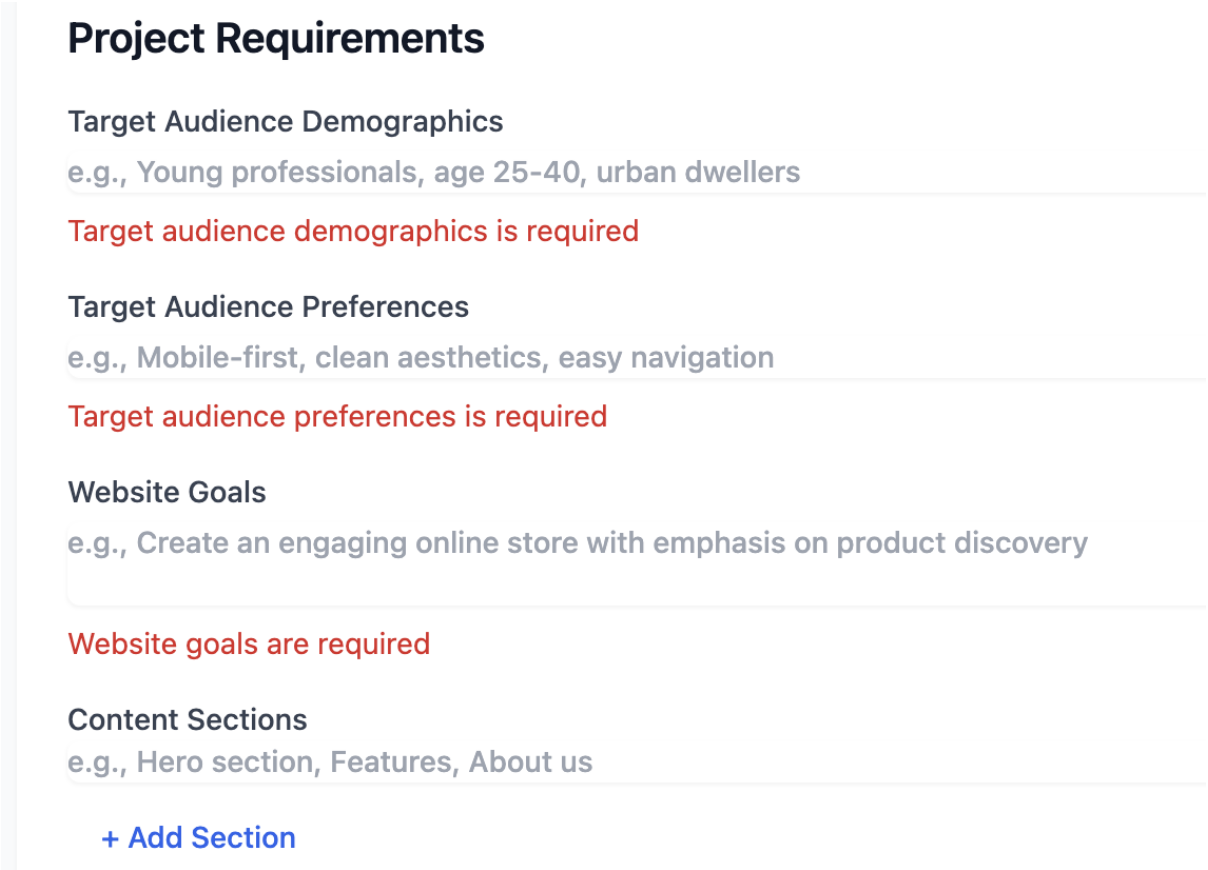

Рисунок 3.14 – Повідомлення про помилку

Як бачимо, після того як ми натиснули кнопку з`явилося повідомлення про помилку (Рис. 3.14). Тобто користувач не може продовжити генерацію макету без заповнених обов’язкових полів.

Далі, заповнюємо форму вимог користувача (Рис. 3.15). Натискаємо кнопку генерувати макет та дивимося чи з`явився перегляд в компоненті Preview. Після натискання кнопки згенерований макет відображається в реальному часі. Макет коректно відображається на мобільному та десктопному вигляді. Код HTML і CSS присутній.

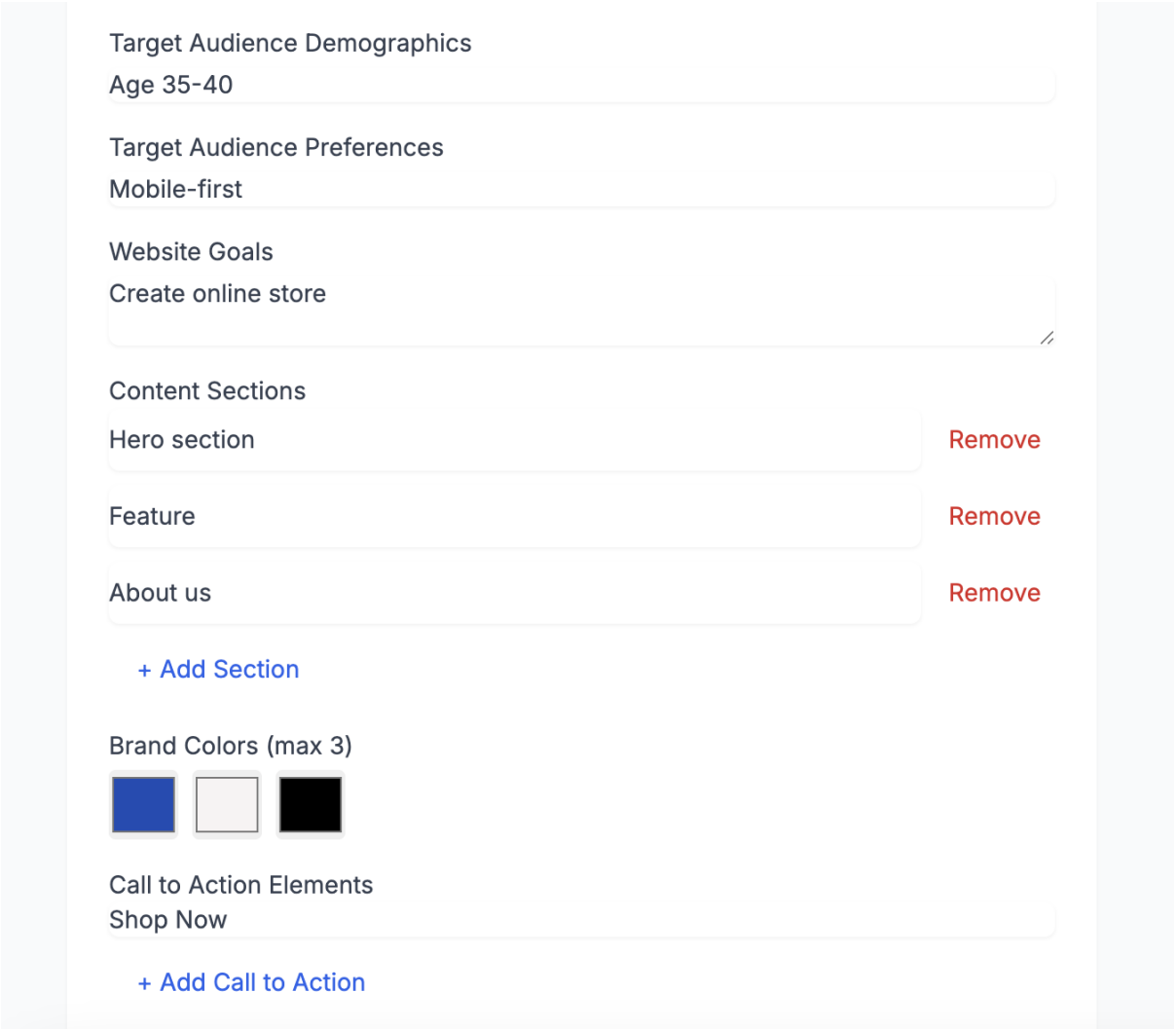

Рисунок 3.15 – Форма вимог користувача

ВИСНОВКИ

Кваліфікаціна робота присвячена зменьшенню інформаного шуму, автоматизації побудови простих та функціональних веб ресурсів із урахуванням вимог до їх доступності. В рамках дослідження інструментів, засобів та дизайнерських підходів було проведено ґрунтовний аналіз сучасних технологій веброзробки та концепцій мінімалістичного дизайну. Зокрема, було досліджено такі інструменти як React, Tailwind CSS, TypeScript та сервіс Hugging Face для інтеграції AI-можливостей. Проаналізовано існуючі аналоги (Wix ADI, Squarespace, Webflow) та визначено їх обмеження щодо реалізації лаконічного підходу у вебдизайні. Це дозволило обрати оптимальний стек технологій для реалізації проєкту.

Було успішно спроектовано комплексну інформаційну модель системи генерації шаблонів, яка включає п'ять ключових компонентів: профіль користувача, вимоги до дизайну, генератор макетів, систему інтерактивності та модуль експорту. Розроблена модель враховує принципи мінімалізму, адаптивності та доступності, що забезпечує створення якісних вебінтерфейсів відповідно до сучасних стандартів.

На основі спроектованої моделі було реалізовано повнофункціональний вебсервіс з використанням сучасних технологій веброзробки. Розроблений додаток надає користувачам можливість:

- Вводити вимоги через зручний інтерфейс.
- Отримувати згенеровані макети.
- Переглядати результат у мобільній та десктопній версіях.
- Експортувати готовий HTML/CSS код.

Проведено комплексне тестування розробленого рішення, яке підтвердило його працездатність та відповідність поставленим вимогам. Зокрема, було протестовано:

- Коректність роботи форми введення даних та валідації полів.

- Функціонування генерації макетів на основі користувацьких вимог.
- Адаптивність створених інтерфейсів.
- Можливість експорту та повторного використання макетів.
- Обробку помилок та граничних випадків.

Таким чином, розроблена інформаційна технологія успішно вирішує поставлені задачі та може бути використана для автоматизації процесу створення мінімалістичних вебінтерфейсів, що відповідають сучасним стандартам вебдизайну.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HuggingFace API Documentation. [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs. – Дата звернення: 27.09.2024.
2. IndexedDB API Documentation. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. – Дата звернення: 19.09.2024.
3. React – Офіційна документація. [Електронний ресурс]. – Режим доступу: https://react.dev/. – Дата звернення: 02.10.2024.
4. Tailwind CSS – Документація. [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs. – Дата звернення: 15.10.2024.
5. Web Accessibility Standards (WCAG 2.1). [Електронний ресурс]. – Режим доступу: https://www.w3.org/WAI/standards-guidelines/wcag/. – Дата звернення: 25.09.2024.
6. JavaScript Guide. Mozilla Developer Network (MDN). [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/Web/JavaScript/Guide. – Дата звернення: 04.10.2024.
7. SCSS Documentation. [Електронний ресурс]. – Режим доступу: https://sass-lang.com/documentation. – Дата звернення: 30.11.2024.
8. Валідація контенту: принципи та інструменти. [Електронний ресурс]. – Режим доступу: https://web.dev/validate-content/. – Дата звернення: 22.10.2024.
9. TypeScript Handbook. [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/handbook/intro.html. – Дата звернення: 19.11.2024.
10. CSS Grid Layout Documentation. Mozilla Developer Network (MDN). [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout. – Дата звернення: 14.10.2024.

11. A Guide to Flexbox. [Електронний ресурс]. – Режим доступу: https://css-tricks.com/snippets/css/a-guide-to-flexbox/. – Дата звернення: 03.11.2024.
12. React Query Documentation. [Електронний ресурс]. – Режим доступу: https://tanstack.com/query/v4/docs/overview. – Дата звернення: 06.10.2024.
13. React Testing Library Documentation. [Електронний ресурс]. – Режим доступу: https://testing-library.com/docs/react-testing-library/intro/. – Дата звернення: 27.11.2024.
14. Ефективна організація даних у IndexedDB. [Електронний ресурс]. – Режим доступу: https://www.smashingmagazine.com/2020/01/indexeddb-introduction/. – Дата звернення: 12.10.2024.
15. The Modern JavaScript Tutorial. [Електронний ресурс]. – Режим доступу: https://javascript.info/. – Дата звернення: 30.09.2024.
16. Основи адаптивного веб-дизайну. [Електронний ресурс]. – Режим доступу: https://web.dev/responsive-web-design-basics/. – Дата звернення: 09.11.2024.
17. Next.js – Офіційний гід. [Електронний ресурс]. – Режим доступу: https://nextjs.org/docs/getting-started. – Дата звернення: 26.11.2024.
18. React Hook Form Documentation. [Електронний ресурс]. – Режим доступу: https://react-hook-form.com/get-started/. – Дата звернення: 01.11.2024.
19. Вступ до веб-доступності (Introduction to Web Accessibility). [Електронний ресурс]. – Режим доступу: https://www.w3.org/WAI/fundamentals/accessibility-intro/. – Дата звернення: 18.11.2024.
20. Principles of Clean Code by Robert C. Martin. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/clean-code/9780136083238/. – Дата звернення: 07.11.2024.

ДОДАТОК А

Код файлу src/App.tsx

```
import React from 'react';
import { InputForm } from './components/InputForm';
import { Preview } from './components/Preview';
import { RecentLayouts } from './components/RecentLayouts';
import type { GeneratedLayout, UserRequirements } from './types';
import { generateLayout } from './services/layout/generator';
import { layoutDB } from './services/db';
import { Wand2 } from 'lucide-react';

export function App() {
  const [layout, setLayout] = React.useState<GeneratedLayout | null>(null);
  const [loading, setLoading] = React.useState(false);
  const [error, setError] = React.useState<string | null>(null);

  const handleSubmit = async (requirements: UserRequirements) => {
    setLoading(true);
    setError(null);

    try {
      const generatedLayout = await generateLayout(requirements);
      await layoutDB.saveLayout(requirements, generatedLayout);
      setLayout(generatedLayout);
    } catch (err) {
      setError((err as Error).message);
    } finally {
      setLoading(false);
    }
  };

  const handleSelectLayout = async (layoutId: string) => {
    const savedLayout = await layoutDB.getLayout(layoutId);
    if (savedLayout) {
      setLayout(savedLayout.layout);
    }
  };

  return (
    <div className="min-h-screen bg-gray-50">
      <header className="bg-white shadow-sm">
```

```
      <div className="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">
        <div className="flex items-center justify-between">
          <div className="flex items-center space-x-3">
            <Wand2 className="w-8 h-8 text-blue-600" />
            <div>
              <h1 className="text-2xl font-bold text-gray-900">Content-to-UI
Generator</h1>
              <p className="text-sm text-gray-500">Transform your content into
beautiful, accessible layouts</p>
            </div>
          </div>
          <a
            href="https://github.com/yourusername/content-to-ui-generator"
            target="_blank"
            rel="noopener noreferrer"
            className="text-sm text-gray-500 hover:text-gray-700"
          >
            View on GitHub
          </a>
        </div>
      </div>
    </header>

    <main className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
      <div className="px-4 py-6 sm:px-0">
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">
          <div className="space-y-4">
            <div className="bg-white p-6 rounded-lg shadow-sm mb-6">
              <h2 className="text-lg font-semibold text-gray-900 mb-2">About
This Tool</h2>
              <p className="text-gray-600 mb-4">
                This AI-powered tool transforms your content requirements into
clean, functional layouts
                using HuggingFace's advanced language models. All generated
layouts are:
              </p>
              <ul className="list-disc list-inside text-gray-600 space-y-2">
                <li>WCAG 2.1 AA compliant</li>
                <li>Responsive and mobile-friendly</li>
                <li>Built with semantic HTML5</li>
                <li>Optimized for performance</li>
              </ul>
            </div>
```

```
            <RecentLayouts onSelect={handleSelectLayout} />
            <InputForm onSubmit={handleSubmit} />
            {error && (
              <div className="p-4 bg-red-50 border border-red-200 rounded-md">
                <p className="text-sm text-red-600">{error}</p>
              </div>
            )}
          </div>
          <div className="bg-white shadow-sm rounded-lg overflow-hidden h-
[calc(100vh-12rem)]">
            {loading ? (
              <div className="h-full flex items-center justify-center">
                <div className="animate-spin rounded-full h-8 w-8 border-b-2
border-blue-600"></div>
              </div>
            ) : (
              <Preview layout={layout} />
            )}
          </div>
        </div>
      </div>
    </main>

    <footer className="bg-white border-t border-gray-200 mt-12">
      <div className="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">
      </div>
    </footer>
  </div>
 );
}
```

ДОДАТОК Б

Код файлу src/Components/FormField.tsx

```
import React from 'react';

interface FormFieldProps {
  label: string;
  error?: string;
  children: React.ReactNode;
}

export function FormField({ label, error, children }: FormFieldProps) {
  return (
    <div className="space-y-2">
      <label className="block text-sm font-medium text-gray-700">
        {label}
        {children}
      </label>
      {error && (
        <p className="text-sm text-red-600">{error}</p>
      )}
    </div>
  );
}
```

Код файлу src/Components/InputForm.tsx

```
import React from 'react';
import { Save } from 'lucide-react';
import { FormField } from './FormField';
import { validateRequirements } from '../utils/validation';
import type { UserRequirements } from '../types';
```

```
interface InputFormProps {
  onSubmit: (requirements: UserRequirements) => void;
}

export function InputForm({ onSubmit }: InputFormProps) {
  const [requirements, setRequirements] = React.useState<UserRequirements>({
    targetAudience: {
      demographics: '',
      preferences: '',
    },
    websiteGoals: '',
    contentSections: [''],
    brandColors: ['#000000'],
    typography: {
      headingFont: 'Inter',
      bodyFont: 'Inter',
    },
    callToAction: [''],
  });

  const [errors, setErrors] = React.useState<Record<string, string>>({});

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();

    const validation = validateRequirements(requirements);

    if (!validation.isValid) {
      const errorMap: Record<string, string> = {};
      validation.errors.forEach(error => {
        // Map errors to specific fields
        if (error.includes('demographics')) errorMap.demographics = error;
        if (error.includes('preferences')) errorMap.preferences = error;
```

```
      if (error.includes('goals')) errorMap.goals = error;
      if (error.includes('content section')) errorMap.contentSections = error;
      if (error.includes('brand color')) errorMap.brandColors = error;
      if (error.includes('heading font')) errorMap.headingFont = error;
      if (error.includes('body font')) errorMap.bodyFont = error;
      if (error.includes('call-to-action')) errorMap.callToAction = error;
    });
    setErrors(errorMap);
    return;
  }

  setErrors({});
  onSubmit(requirements);
};

const addSection = () => {
  if (requirements.contentSections.length < 5) {
    setRequirements(prev => ({
      ...prev,
      contentSections: [...prev.contentSections, ''],
    }));
  }
};

const removeSection = (index: number) => {
  setRequirements(prev => ({
    ...prev,
    contentSections: prev.contentSections.filter((_, i) => i !== index),
  }));
};

return (
```

```
   <form onSubmit={handleSubmit} className="space-y-6 bg-white p-6
rounded-lg shadow-sm">
     <div className="space-y-4">
       <h2 className="text-xl font-semibold text-gray-900">Project
Requirements</h2>

       <FormField
         label="Target Audience Demographics"
         error={errors.demographics}
       >
         <input
           type="text"
           className="mt-1 block w-full rounded-md border-gray-300 shadow-sm
focus:border-blue-500 focus:ring-blue-500"
           value={requirements.targetAudience.demographics}
           onChange={e => setRequirements(prev => ({
             ...prev,
             targetAudience: { ...prev.targetAudience, demographics: e.target.value
},
           }))}
           placeholder="e.g., Young professionals, age 25-40, urban dwellers"
         />
       </FormField>

       <FormField
         label="Target Audience Preferences"
         error={errors.preferences}
       >
         <input
           type="text"
           className="mt-1 block w-full rounded-md border-gray-300 shadow-sm
focus:border-blue-500 focus:ring-blue-500"
           value={requirements.targetAudience.preferences}
```

```
          onChange={e => setRequirements(prev => ({
            ...prev,
            targetAudience: { ...prev.targetAudience, preferences: e.target.value },
          }))}
          placeholder="e.g., Mobile-first, clean aesthetics, easy navigation"
        />
      </FormField>

      <FormField
        label="Website Goals"
        error={errors.goals}
      >
        <textarea
          className="mt-1 block w-full rounded-md border-gray-300 shadow-sm
focus:border-blue-500 focus:ring-blue-500"
          value={requirements.websiteGoals}
          onChange={e => setRequirements(prev => ({ ...prev, websiteGoals:
e.target.value }))}
          placeholder="e.g., Create an engaging online store with emphasis on
product discovery"
        />
      </FormField>

      <FormField
        label="Content Sections"
        error={errors.contentSections}
      >
        <div className="space-y-2">
          {requirements.contentSections.map((section, index) => (
            <div key={index} className="flex gap-2">
              <input
                type="text"
```

```
              className="flex-1 rounded-md border-gray-300 shadow-sm
focus:border-blue-500 focus:ring-blue-500"
              value={section}
              placeholder="e.g., Hero section, Features, About us"
              onChange={e => {
                const newSections = [...requirements.contentSections];
                newSections[index] = e.target.value;
                setRequirements(prev => ({ ...prev, contentSections: newSections
}));
              }}
            />
            {requirements.contentSections.length > 1 && (
              <button
                type="button"
                onClick={() => removeSection(index)}
                className="p-2 text-red-600 hover:text-red-800"
              >
                Remove
              </button>
            )}
          </div>
        ))}
        {requirements.contentSections.length < 5 && (
          <button
            type="button"
            onClick={addSection}
            className="mt-2 px-4 py-2 text-sm font-medium text-blue-600
hover:text-blue-800"
          >
            + Add Section
          </button>
        )}
      </div>
```

```
      </FormField>

      <FormField
        label="Brand Colors (max 3)"
        error={errors.brandColors}
      >
        <div className="flex gap-2 mt-1">
          {requirements.brandColors.map((color, index) => (
            <input
              key={index}
              type="color"
              className="h-10 w-10 rounded cursor-pointer"
              value={color}
              onChange={e => {
                const newColors = [...requirements.brandColors];
                newColors[index] = e.target.value;
                setRequirements(prev => ({ ...prev, brandColors: newColors }));
              }}
            />
          ))}
          {requirements.brandColors.length < 3 && (
            <button
              type="button"
              onClick={() => setRequirements(prev => ({
                ...prev,
                brandColors: [...prev.brandColors, '#000000'],
              }))}
              className="h-10 w-10 rounded border-2 border-dashed border-gray-
300 flex items-center justify-center text-gray-400 hover:text-gray-500"
            >
              +
            </button>
          )}
```

```
        </div>
      </FormField>

      <FormField
        label="Call to Action Elements"
        error={errors.callToAction}
      >
        <div className="space-y-2">
          {requirements.callToAction.map((cta, index) => (
            <div key={index} className="flex gap-2">
              <input
                type="text"
                className="flex-1 rounded-md border-gray-300 shadow-sm
focus:border-blue-500 focus:ring-blue-500"
                value={cta}
                placeholder="e.g., Shop Now, Contact Us, Learn More"
                onChange={e => {
                  const newCTAs = [...requirements.callToAction];
                  newCTAs[index] = e.target.value;
                  setRequirements(prev => ({ ...prev, callToAction: newCTAs }));
                }}
              />
              {requirements.callToAction.length > 1 && (
                <button
                  type="button"
                  onClick={() => {
                    const newCTAs = requirements.callToAction.filter((_, i) => i !==
index);
                    setRequirements(prev => ({ ...prev, callToAction: newCTAs }));
                  }}
                  className="p-2 text-red-600 hover:text-red-800"
                >
                  Remove
```

```
                </button>
              )}
            </div>
          ))}
          {requirements.callToAction.length < 3 && (
            <button
              type="button"
              onClick={() => setRequirements(prev => ({
                ...prev,
                callToAction: [...prev.callToAction, ''],
              }))}
              className="mt-2 px-4 py-2 text-sm font-medium text-blue-600
hover:text-blue-800"
            >
              + Add Call to Action
            </button>
          )}
        </div>
      </FormField>
    </div>

    <div className="flex justify-end gap-4">
      <button
        type="submit"
        className="inline-flex items-center px-4 py-2 border border-transparent
text-sm font-medium rounded-md shadow-sm text-white bg-blue-600 hover:bg-
blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-
500"
      >
        <Save className="w-4 h-4 mr-2" />
        Generate Layout
      </button>
    </div>
```

```
    </form>
  );
}
```

Код файлу src/Components/Preview.tsx

```
import React from 'react';
import type { GeneratedLayout } from '../types';
import { PreviewHeader } from './preview/PreviewHeader';
import { CodeView } from './preview/CodeView';
import { LayoutPreview } from './preview/LayoutPreview';
import { ErrorState } from './preview/ErrorState';
import { LoadingState } from './preview/LoadingState';

interface PreviewProps {
  layout: GeneratedLayout | null;
  error?: string | null;
  loading?: boolean;
}

export function Preview({ layout, error, loading }: PreviewProps) {
  const [viewMode, setViewMode] = React.useState<'desktop' |
'mobile'>('desktop');
  const [activeTab, setActiveTab] = React.useState<'preview' |
'code'>('preview');

  if (loading) {
    return <LoadingState />;
  }

  if (error) {
    return <ErrorState message={error} />;
  }
```

```
  if (!layout) {
    return (
      <div className="h-full flex items-center justify-center text-gray-500">
        Generate a layout to see the preview
      </div>
    );
  }

  return (
    <div className="h-full flex flex-col">
      <PreviewHeader
        viewMode={viewMode}
        activeTab={activeTab}
        onViewModeChange={setViewMode}
        onTabChange={setActiveTab}
      />

      <div className="flex-1 overflow-auto">
        {activeTab === 'preview' ? (
          <LayoutPreview layout={layout} viewMode={viewMode} />
        ) : (
          <CodeView layout={layout} />
        )}
      </div>
    </div>
  );
}
```

Код файлу src/Components/RecentLayouts.tsx

```
import React from 'react';
import { layoutDB } from '../services/db';
import { Clock, Trash2 } from 'lucide-react';
```

```
interface RecentLayoutsProps {
  onSelect: (layoutId: string) => void;
}

export function RecentLayouts({ onSelect }: RecentLayoutsProps) {
  const [layouts, setLayouts] = React.useState<Awaited<ReturnType<typeof
layoutDB.getRecentLayouts>>>([]);

  const loadLayouts = React.useCallback(async () => {
    const recentLayouts = await layoutDB.getRecentLayouts();
    setLayouts(recentLayouts);
  }, []);

  React.useEffect(() => {
    loadLayouts();
  }, [loadLayouts]);

  const handleDelete = async (id: string, e: React.MouseEvent) => {
    e.stopPropagation();
    await layoutDB.deleteLayout(id);
    await loadLayouts();
  };

  if (layouts.length === 0) {
    return null;
  }

  return (
    <div className="bg-white p-6 rounded-lg shadow-sm mb-6">
      <div className="flex items-center space-x-2 mb-4">
        <Clock className="w-4 h-4 text-gray-500" />
        <h2 className="text-lg font-semibold text-gray-900">Recent
Layouts</h2>
```

```
      </div>
      <div className="space-y-2">
        {layouts.map((item) => (
          <div
            key={item.id}
            onClick={() => onSelect(item.id)}
            className="flex items-center justify-between p-3 rounded-md border
border-gray-200 hover:border-blue-500 cursor-pointer"
          >
            <div>
              <p className="text-sm font-medium text-gray-900">
                {item.requirements.websiteGoals.slice(0, 50)}...
              </p>
              <p className="text-xs text-gray-500">
                {new Date(item.createdAt).toLocaleDateString()}
              </p>
            </div>
            <button
              onClick={(e) => handleDelete(item.id, e)}
              className="p-1 text-gray-400 hover:text-red-500"
            >
              <Trash2 className="w-4 h-4" />
            </button>
          </div>
        ))}
      </div>
    </div>
  );
}
```

ДОДАТОК В

Код файлу src/Components/Preview/ErrorState.tsx

```
import React from 'react';
import { AlertCircle } from 'lucide-react';

interface ErrorStateProps {
  message: string;
}

export function ErrorState({ message }: ErrorStateProps) {
  return (
    <div className="h-full flex flex-col items-center justify-center text-red-600
p-4">
      <AlertCircle className="w-8 h-8 mb-2" />
      <p className="text-center font-medium">{message}</p>
      <p className="text-sm text-red-500 mt-1">Please try again or contact
support if the issue persists.</p>
    </div>
  );
}
```

Код файлу src/Components/Preview/LayoutPreview.tsx

```
import React from 'react';
import type { GeneratedLayout } from '../../types';

interface LayoutPreviewProps {
  layout: GeneratedLayout;
  viewMode: 'desktop' | 'mobile';
}

export function LayoutPreview({ layout, viewMode }: LayoutPreviewProps) {
  return (
    <div
      className={`mx-auto p-4 ${
```

```
        viewMode === 'mobile' ? 'max-w-sm' : 'max-w-6xl'
      }`}
    >
      <div
        className="preview-container"
        dangerouslySetInnerHTML={{ __html: layout.html }}
      />
      <style>{layout.css}</style>
    </div>
  );
}
```

Код файлу src/Components/Preview/LoadingState.tsx

```
import React from 'react';

export function LoadingState() {
  return (
    <div className="h-full flex items-center justify-center">
      <div className="flex flex-col items-center">
        <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-
blue-600 mb-2"></div>
        <p className="text-sm text-gray-500">Generating layout...</p>
      </div>
    </div>
  );
}
```

Код файлу src/Components/Preview/PreviewHeader.tsx

```
import React from 'react';
import { Monitor, Smartphone } from 'lucide-react';

interface PreviewHeaderProps {
  viewMode: 'desktop' | 'mobile';
  activeTab: 'preview' | 'code';
  onViewModeChange: (mode: 'desktop' | 'mobile') => void;
```

```
  onTabChange: (tab: 'preview' | 'code') => void;
}

export function PreviewHeader({
  viewMode,
  activeTab,
  onViewModeChange,
  onTabChange
}: PreviewHeaderProps) {
  return (
    <div className="border-b border-gray-200">
      <div className="flex justify-between items-center p-4">
        <div className="flex space-x-4">
          <button
            onClick={() => onTabChange('preview')}
            className={`px-4 py-2 text-sm font-medium rounded-md ${
              activeTab === 'preview'
                ? 'bg-blue-100 text-blue-700'
                : 'text-gray-500 hover:text-gray-700'
            }`}
          >
            Preview
          </button>
          <button
            onClick={() => onTabChange('code')}
            className={`px-4 py-2 text-sm font-medium rounded-md ${
              activeTab === 'code'
                ? 'bg-blue-100 text-blue-700'
                : 'text-gray-500 hover:text-gray-700'
            }`}
          >
            Code
          </button>
        </div>
        <div className="flex space-x-2">
          <button
            onClick={() => onViewModeChange('desktop')}
            className={`p-2 rounded-md ${
              viewMode === 'desktop'
                ? 'bg-gray-100 text-gray-900'
                : 'text-gray-500 hover:text-gray-700'
            }`}
            aria-label="Desktop view"
```

```
          >
            <Monitor className="w-5 h-5" />
          </button>
          <button
            onClick={() => onViewModeChange('mobile')}
            className={`p-2 rounded-md ${
              viewMode === 'mobile'
                ? 'bg-gray-100 text-gray-900'
                : 'text-gray-500 hover:text-gray-700'
            }`}
            aria-label="Mobile view"
          >
            <Smartphone className="w-5 h-5" />
          </button>
        </div>
      </div>
    </div>
  );
}
```

ДОДАТОК Г

Код файлу src/main.tsx

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import { App } from './App';
import './index.css';

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>

);
```