ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти Бакалавр

на тему: «Вебсервіс для підготовки до співбесіди в ІТ-компанію за допомогою штучного інтелекту»

Виконав: здобувач вищої освіти
за освітньо-професійною
програмою Інформаційні
управляючі системи спеціальності
126 Інформаційні системи та
технології
ступеня вищої освіти Бакалавр
групи 126ІСТбд41
Михно Т.О.
Керівник: Слюсар В.І.
Рецензент: Брикун О.М.

Полтава – 2024 року

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується необхідністю реалізації структурованої та якісної підготовки кандидатів для роботи в IT-галузі в умовах зростання конкуренції. Багато з претендентів відчувають труднощі в аналізі своїх знань, визначенні слабких місць і практиці з типовими питаннями, що задаються на співбесідах. Це може призвести до невдачі навіть кваліфікованих спеціалістів на етапі відбору. Вебсервіс, що використовує штучний інтелект для підготовки до співбесід, може значно підвищити ефективність цього процесу. За допомогою інноваційних технологій, таких як машинне навчання та обробка природної мови, можна створити індивідуалізовані тренувальні програми, які допоможуть кандидатам покращити свої навички та підготуватися до співбесід у реальних умовах. Вебсервіс, що використовує штучний інтелект, може аналізувати індивідуальні потреби та надавати персоналізовані рекомендації.

При цьому, штучний інтелект може швидко адаптуватися до нових вимог і трендів, а також забезпечити інтерактивні тренування, включаючи симуляції співбесід, що дозволяє кандидатам практикувати свої відповіді в умовах, наближених до реальних. В свою чергу, вебсервіс може бути доступним у будь-який час та з будь-якого місця, що забезпечує зручність для кандидатів, які можуть підготуватися до співбесіди у зручний для них час. Все це свідчить про актуальність теми роботи.

Метою кваліфікаційної роботи є удосконалення технології вебдодатку для підвищення ефективності підготовки кандидатів на працевлаштування в IT-компанії за рахунок використання штучного інтелекту.

Завданнями кваліфікаційної роботи є:

- аналіз рекомендацій щодо оптимізації роботи вебдодатку;
- визначення особливостей використання бібліотеки React;
- розробка вебсервісу для підготовки до співбесіди в IT-компанію за допомогою штучного інтелекту.

Об'єктом дослідження є процес підготовки кандидатів до співбесіди у сфері інформаційних технологій через вебсервіс.

Предметом дослідження є ефективність та адаптивність методів штучного інтелекту в контексті підготовки до співбесіди в IT-компанії.

Методами дослідження в рамках визначення інструментарію для веброзробки, і економічного обґрунтування прийнятих рішень використовувався аналітичний метод досліджень, а для веброзробки за допомогою бібліотек React JS – моделювання.

Інформаційна база кваліфікаційної роботи сформована з ресурсів, що містять інформацію про інструментарій для розробки вебдодатків та користувацького інтерфейсу.

Практична значущість роботи полягає у розробці вебдодатку для підготовки до співбесіди в IT-компанію – може бути використані для подальших досліджень за даною тематикою та при проектуванні нейропомічників.

Апробація результатів відбувалася в рамках науково-практичної конференції за підсумками проходження виробничої практики здобувачів вищої освіти спеціальності 126 Інформаційні системи та технології (17 жовтня 2023 р., м. Полтава), XIX щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету (14 травня 2024 р., м. Полтава).

За результатами досліджень здійснено 2 публікації тез доповідей.

Структура кваліфікаційної роботи логічно пов'язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 59 сторінок формату А4. Вона містить 34 рисунки і 2 таблиці.

РОЗДІЛ 1
АНАЛІЗ ІНСТРУМЕНТАРІЮ ДЛЯ РОЗРОБКИ ВЕБДОДАТКІВ

1.1 Підходи до розробки вебдодатків

У світі сучасних технологій вебдодатки відіграють ключову роль, забезпечуючи доступ до інформації мережу Інтернет. Вони являють собою складні програмні системи, що поєднують у собі серверну та клієнтську частини, працюючи в тандемі для забезпечення взаємодії з користувачем.

Серверна частина вебдодатку, що виконується на віддаленому сервері, є мозковим центром системи. Вона відповідає на запити, обробляє їх, забезпечує виконання бізнес-логіки, взаємодію з базами даних та інші операції, що вимагають обчислювальних ресурсів та доступу до даних. Серверна частина забезпечує надійність, безпеку та масштабованість вебдодатку, дозволяючи йому обслуговувати багато користувачів одночасно [1].

Клієнтська частина, яка виконується у браузері користувача, відповідає за візуальне представлення інформації та взаємодію з користувачем. Вона отримує дані від сервера, обробляє їх та відображає у зручному для користувача форматі. Клієнтська частина також відповідає за обробку дій користувача, таких як кліки, введення тексту та інші форми взаємодії, та надсилання відповідних запитів на сервер.

Ключовою перевагою вебдодатків є міжплатформовість. Оскільки вони працюють у браузері, користувачі можуть користуватись вебдодатком з будь-якого девайсу, будь-якої операційної системи. Це робить вебдодатки універсальним та зручним інструментом для користування [2].

Сучасні вебдодатки використовують різноманітні технології для створення інтерфейсів, забезпечення обробки даних [3]. Поєднання потужних серверних технологій, як Node.js, з гнучкими клієнтськими бібліотеками, як наприклад React, дає можливість розробляти вебдодатки, що задовольняють найвищі вимоги сучасного користувача.

Підходи до розробки вебдодатків:

Клієнт-серверний підхід – є основоположною моделлю для побудови вебдодатків, яка забезпечує чіткий розподіл функціональності між двома основними компонентами: клієнтом (фронтендом) та сервером (бекендом). Цей розподіл дозволяє оптимізувати роботу додатку, підвищити його масштабованість та спростити підтримку [4].

Клієнт (Frontend) – обличчя вебдодатку клієнтська частина, яку часто називають фронтендом, відповідає за візуальне представлення вебдодатку та взаємодію з користувачем. Клієнтська частина включає в себе все, що користувач бачить та з чим взаємодіє у своєму браузері: інтерфейс користувача, кнопки, форми, анімації та інший контент [5]. Клієнтська частина також відповідає за обробку дій користувача, таких як кліки, введення тексту, прокручування сторінки тощо, та надсилання відповідних запитів на сервер.

Для створення клієнтської частини вебдодатку використовуються : HTML (структура сторінки), CSS (стилізація) та JavaScript (логіка та інтерактивність), також використовуються більш сучасні фреймворки та бібліотеки, наприклад React, Angular, значно спрощують та прискорюють розробку фронтенду, надаючи готові компоненти, інструменти для управління станом та інші корисні функції.

Серверна частина, яку також називають бекендом, працює на віддаленому сервері та виконує основну логіку вебдодатку. Бекенд отримує запити з клієнта, обробляє їх, взаємодіє з базами даних, файловою системою та іншими ресурсами, виконує необхідні обчислення та повертає результати клієнту у вигляді даних або HTML-сторінок.

Для розробки бекенду використовуються різноманітні мови програмування, кожна мова має особливості [6]. Node.js, наприклад, завдяки своїй неблокуючій архітектурі та використанню JavaScript, є кращим варіантом для створення швидкого та масштабованого бекенду, особливо для вебдодатків що працюють з даними в реальному часі.

Фронтенд та серверна частини вебдодатку обмінюються даними,

використовуючи протокол HTTP (Hypertext Transfer Protocol). Клієнт відправляє запит по HTTP на бекенд сервер, вказуючи, яку дію він хоче виконати (наприклад, отримати сторінку, відправити дані форми або завантажити файл). Сервер обробляє запит, виконує необхідні дії та повертає відповідь клієнту, дані у форматі JSON, XML або HTML-сторінку для відображення у браузері [7].

Переваги клієнт-серверного підходу:

– чіткий розподіл відповідальності: кожен компонент виконує свої функції, що спрощує розробку;

– масштабованість: можна легко збільшувати потужність сервера або додавати нові сервери для обробки більшої кількості запитів;

– гнучкість: можна використовувати різні технології для розробки клієнтської та серверної частини додатку;

– безпека: сервер може виконувати перевірку даних та авторизацію користувачів.

Клієнт-серверний підхід є основою більшості сучасних вебдодатків. Він дозволяє створювати масштабовані системи, які ефективно працюють та надають користувачам багато функціональних можливостей.

Односторінковий (Single-PageApplication,SPA) підхід – це сучасна парадигма веброзробки, яка революціонізувала взаємодію з вебдодатками. На відміну від традиційних багатосторінкових застосунків, де кожна дія користувача викликає перезавантаження сторінки, SPA підхід робить оновлення контенту на сторінці без оновлення самої сторінки.

У SPA підході HTML, CSS та JavaScript, завантажуються один раз при першому відвідуванні сторінки. Після цього, при переході між різними розділами або виконанням дій користувача, замість повного перезавантаження сторінки, відбувається динамічне оновлення лише необхідних частин контенту. Це досягається за допомогою JavaScript, який взаємодіє з сервером через API (Application Programming Interface) для отримання та оновлення даних [8].

До переваг використання SPA можна виділити:

– швидкість та відгук: SPA забезпечують більш швидку роботу вебдодатку, оскільки немає необхідності чекати на повне перезавантаження сторінки при кожній дії;

– навантаження на сервер: оскільки SPA завантажують всі необхідні ресурси один раз, це зменшує обсяг переданих даних. Це дозволяє серверу працювати ефективніше та обслуговувати більшу кількість користувачів;

Недоліками використання SPA можна вважати:

– початкове завантаження: початкове завантаження SPA може займати більше часу, оскільки всі необхідні ресурси завантажуються одночасно. Для вирішення цього слід використовувати такі техніки, як ледаче завантаження (lazy loading) та розбиття коду на менші частини (code splitting);

– SEO: оскільки контент SPA генерується динамічно на стороні клієнта, це може ускладнити індексацію пошуковими системами. Для покращення SEO можна зробити серверний рендеринг (SSR) або попередній рендеринг.

React – являє собою один з провідних інструментів, який активно використовується для розробки односторінкових додатків. Його компонентна архітектура, віртуальний DOM та декларативний підхід ідеально підходять для побудови динамічних та інтерактивних інтерфейсів [9].

Односторінкові застосунки (SPA) стали стандартом для створення сучасних вебдодатків, забезпечуючи високу швидкість, інтерактивність та покращений користувацький досвід. React, з його потужними можливостями та великою екосистемою, є ідеальним інструментом для розробки SPA будь-якої складності.

1.2 Середовища розробки для створення вебдодатку

У світі веброзробки існує безліч кодових редакторів, всі вони мають як переваги так і недоліки. Вибір найкращого інструменту залежить від потреб розробника, його вподобань, особливостей проекту та технологій, що використовуються.

Серед найпопулярніших кодових редакторів для веброзробки можна виділити такі:

- Visual Studio Code: Універсальний інструмент від Microsoft, який підтримує широкий спектр мов програмування та пропонує величезну бібліотеку розширень для індивідуального налаштування.
- Sublime Text: Легкий та швидкий редактор, відомий своєю простотою та широким набором функцій для підвищення продуктивності розробника.
- Atom: Гнучкий редактор, який дозволяє легко налаштовувати його за допомогою розширень та тем, адаптуючи до власних потреб.
- Visual Studio: Потужне інтегроване середовище розробки (IDE) від Microsoft, що надає набір інструментів для розробки програмного забезпечення.
- Brackets: Спеціалізований редактор для веброзробників, що фокусується на HTML, CSS та JavaScript, з вбудованою функцією live preview для миттєвого перегляду змін.
- PhpStorm: Професійний редактор, створений спеціально для розробки на PHP, але також підтримує інші мови та пропонує інструменти для веброзробки.

Вибір кодового редактору залежить від потреб розробника. Одні редактори пропонують ширший вибір розширень та плагінів, інші відрізняються швидкістю та легкістю. При виборі варто враховувати мови програмування на яких буде написано додаток, оскільки кодові редактори можуть мати кращу підтримку якоїсь конкретної технології.

1.3 Основні компоненти вебсторінки

Вебсторінка – це не просто набір тексту та зображень, а складна структура, що складається з різноманітних компонентів, які відповідають за створення цілісного та функціонального інтерфейсу користувача [10]. Розглянемо основні будівельні блоки, що формують вебсторінку:

1. Заголовок (Header): це частина вебсторінки, зазвичай знаходиться найвище за всі інші елементи, яка зазвичай містить логотип сайту, назву, меню навігації, що допомагає користувачу зорієнтуватися та знайти потрібну інформацію. Заголовок є важливим елементом брендингу та першого враження про вебсайт.

2. Навігація (Navigation): це система меню, посилань, які допомагають користувачеві переміщуватися між різними розділами вебсайту.

3. Підвал (Footer): це частина вебсторінки, яка зазвичай розташована знизу під всім іншим контентом та містить контактну інформацію, соціальні мережі, карту сайту, політику конфіденційності та інші додаткові відомості.

4. Бічна панель (Sidebar): це додаткова область на сторінці, яка зазвичай розташована збоку від основного контенту. Бічна панель може містити різноманітну інформацію, таку як реклама, віджети соціальних мереж, архів статей, список популярних публікацій тощо.

5. Форми (Forms): це інтерактивні елементи, для вводу даних користувачами та відправлення їх на сервер. Форми використовуються для реєстрації, авторизації, оформлення замовлень, зворотного зв'язку та інших цілей.

6. Медіа-елементи (Media): це зображення, відео, аудіо які використовуються для ілюстрації контенту, створення атмосфери та залучення уваги користувачів.

Всі ці елементи важливі у створенні успішного вебсайту. Вони працюють разом, як частини цілісного механізму, забезпечуючи привабливість дизайну вебдодатку та зручність користування.

1.4 Принципи позиціонування елементів вебдодатку

Позиціонування елементів є ключовим аспектом вебдизайну та розробки, що визначає розташування та візуальне представлення елементів на сторінці вебдодатку [11].

Правильне позиціонування надає користувачам зручність, естетичну привабливість та адаптивність дизайну для екранів з різними розмірами.

Основні принципи позиціонування:

1. Нормальний потік документа (Normal Flow): це стандартний спосіб розташування елементів у HTML-документі. Елементи блочного рівня (block-level elements) розташовуються один під одним вертикально, займаючи всю доступну ширину, тоді як елементи рядкового рівня (inline-level elements) розташовуються в рядку, займаючи лише необхідний простір [12].

2. Позиціонування відносно потоку (Relative Positioning): цей тип позиціонування дозволяє зміщувати елемент від початкового положення у нормальному потоці документа. Для цього використовується властивість CSS position: relative та властивості зміщення top,right,bottom та left.

3. Позиціонування поза потоком (Absolute Positioning): цей тип позиціонування виводить елемент з потоку документа та дозволяє розміщувати його у будь-де на сторінці відносно найближчого батьківського елемента з позиціонуванням relative, absolute або fixed. Для цього використовується властивість CSS position: absolute [13].

4. Фіксоване позиціонування (Fixed Positioning): цей тип позиціонування схожий на абсолютне, але елемент буде фіксуватися відносно вікна браузера, а не батьківського елемента. Це означає, що елемент залишається на своєму місці навіть при прокручуванні сторінки. Для цього використовується властивість CSS position: fixed.

5. Плаваючі елементи (Floating Elements): плаваючі елементи виводяться з нормального потоку документа та розміщуються зліва або справа від контейнера.

6. Flexbox: це інструмент для гнучких та адаптивних макетів. Flexbox дозволяє легко вирівнювати елементи, розподіляти простір між ними та змінювати їх порядок [14].

7. Grid Layout: це ще один потужний інструмент, що надає більше можливостей для контролю розміщення елементів, ніж Flexbox.

Важливі аспекти позиціонування:

– контекст накладання (Stacking Context): це порядок, в якому елементи перекривають один одного при накладанні [15]. Контекст накладання визначається властивістю CSS z-index;

– відступи (Margins) та поля (Padding): це простір навколо елемента, який використовується для візуального розділення елементів та створення відступів;

– адаптивність: важливо враховувати розміри екранів, використовуючи медіа-запити.

Розуміння основних принципів позиціонування елементів є важливим для створення гармонійного, зручного та адаптивного дизайну вебдодатку.

1.5 Аналіз рекомендацій щодо оптимізації роботи вебдодатку

Оптимізація роботи вебдодатку є критично важливим аспектом для забезпечення його успіху. Швидкий та ефективний застосунок сприяє кращій індексації пошуковими системами та підвищенню конверсії. Ось деякі ключові рекомендації для оптимізації вебдодатку:

1. Оптимізація зображень. Використання інструментів для стиснення зображень без втрати якості (наприклад, TinyPNG, ImageOptim), щоб зменшити їх розмір та прискорити завантаження сторінки. Застосування сучасних форматів зображень, такі як WebP або AVIF, які забезпечують краще стиснення порівняно з традиційними форматами JPEG та PNG. Завантаження зображення тоді, коли зображення з'являються у полі зору користувача, щоб прискорити початкове завантаження сторінки.

2. Оптимізація коду. Зменшення розміру CSS та JavaScript файлів, видаливши зайві пробіли, коментарі до коду, непотрібні символи. Об'єднання кілька файлів для зменшення кількість запитів на бекенд вебдодатку [16]. Аналізувати та оптимізувати запити до бази даних, щоб зменшити час їх виконання та навантаження на сервер.

3. Використання CDN. Розподіляти статичні ресурси (зображення, CSS, JavaScript) по мережі серверів по всьому світу, щоб прискорити їх доставку.

4. Оптимізація рендерингу. Використання Server-Side Rendering (SSR) рендеринг сторінки на сервері дозволяє прискорити початкове завантаження та покращити SEO, або Client-Side Rendering (CSR) рендеринг сторінки на стороні клієнта дозволяє розробляти інтерактивні та динамічні вебдодатки. Оптимізація роботи – вимагає постійної уваги та вдосконалення [17].

Використовуючи перераховані методики можна значно покращити продуктивність додатку та забезпечити позитивний досвід користувача.

Таким чином, у першому розділі визначено, що розробка сучасних вебдодатків – це складний, багатогранний процес що вимагає глибокого розуміння архітектури, компонентного підходу, односторінкових застосунків та інших технологій [18]. Від вибору правильних інструментів та підходів залежить успіх проекту, його продуктивність та зручність.

РОЗДІЛ 2
ІНТЕГРАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА ВЕБСЕРВІСУ

2.1 Особливості використання бібліотеки React

React – це передова JavaScript бібліотека розроблена Facebook для створення високопродуктивних, інтерактивних та масштабованих користувацьких інтерфейсів [19]. Вона надає розробникам потужний інструментарій для побудови складних вебдодатків, які можуть динамічно оновлюватися без необхідності перезавантаження всієї сторінки.

Основна філософія React полягає в компонентному підході. Кожен окремий компонент містить логіку, стан та вигляд що стосується саме його, це робить розробку вебдодатку легшим та зрозумілішим. Завдяки декларативному стилю написання коду, розробники описують бажаний стан інтерфейсу, а React автоматично оновлює його при зміні даних [20].

React використовує Virtual DOM. Він оновлює необхідні частини віртуального DOM при зміні даних, а потім ефективно синхронізує його з реальним DOM, мінімізуючи кількість маніпуляцій з браузером та значно покращуючи продуктивність додатків.

React активно розвивається та має величезну спільноту, що сприяє постійному вдосконаленню бібліотеки, створення нових інструментів та розширення екосистеми [21].

При тому, що React має певний поріг входження для новачків, його переваги, такі як висока продуктивність, компонентний підхід, декларативний стиль та велика спільнота, створюють умови для зручного створення сучасних, масштабованих та зручних у використанні вебдодатків.

Бібліотека виникла у 2011 р. всередині Facebook, коли команда розробників, включаючи Джордана Волке, зіткнулася з недостатньою масштабованістю та ефективністю існуючих інструментів для побудови користувацького інтерфейсу.

Внутрішній проект Facebook під назвою XHP виявився недостатньо гнучким та продуктивним для задоволення потреб компанії, що швидко зростала. Це спонукало команду до пошуку нових рішень, і результатом їхніх зусиль став React [22].

React, представлений публіці у 2013 р., швидко здобув визнання серед розробників завдяки своїй продуктивності, гнучкості та інтуїтивно зрозумілому синтаксису. Активне використання Facebook у таких своїх продуктах, як Facebook та Instagram, значно сприяло розвитку та популяризації React.

У 2015 р. React представив революційні концепції функціональних компонентів та хуків (hooks), які спростили створення та управління компонентами, надаючи їм доступ до стану, методів життєвого циклу та інших функцій, раніше доступних лише класовим компонентам.

2017 р. ознаменувався випуском React 16, який приніс значні покращення, включаючи підтримку фрагментів, порталів, оптимізовану обробку помилок та новий алгоритм реконциліації (Fiber), що забезпечив плавнішу та швидшу роботу додатків.

Еволюція React продовжується з кожною новою версією. React 17, випущений у 2020 р., зосередився на стабільності, тоді як React 18, представлений у 2022 р., приніс інноваційні можливості, такі як Server Components, автоматичне розпаралелювання рендерингу та покращене керування пам'яттю [23]. Сьогодні React це найпопулярніша та впливовіша бібліотека для розробки вебінтерфейсів. Він продовжує розвиватися та вдосконалюватися, залишаючись на передовій технологій фронтенд розробки.

У React все обертається навколо компонентів. Компонент – це незалежний, багаторазовий блок коду, який описує частину користувацького інтерфейсу. Він може бути простим, як кнопка або поле введення, або складним, як ціла сторінка або навіть цілий додаток.

Основні принципи компонентної моделі:

1. Інкапсуляція: кожен компонент в React функціонує як окремий модуль, володіючи власною логікою, станом та зовнішнім виглядом [24]. Така

автономність спрощує розробку, оскільки зміни в одному компоненті не зачіпають інші, забезпечуючи стабільність та передбачуваність роботи додатку.

2. Композиція: компоненти можуть об'єднуватись та вкладатись один в одного, утворюючи ієрархічну структуру.

3. Повторне використання: оскільки компоненти є незалежними, їх можна легко використовувати повторно в різних частинах додатку. Це значно прискорює розробку та зменшує дублювання коду.

4. Стан (State): компоненти мають внутрішній стан, що визначає їх поведінку та вигляд [25]. При зміні стану компонент автоматично оновляється, відображаючи нові дані.

5. Властивості (Props): компоненти отримувють дані ззовні через властивості (props). Це дозволяє налаштовувати поведінку та вигляд компонентів залежно від контексту їх використання.

React пропонує два основних типи компонентів:

– функціональні компоненти: це прості JavaScript функції, які приймають властивості (props) як аргумент та повертають елемент React, що описує вигляд компонента. Вони є більш лаконічними та легкими для розуміння;

– класові компоненти: це класи ES6, які розширюють клас Component від React. Вони мають більш складну структуру та використовують методи життєвого циклу, обробники подій та інші функції.

Переваги компонентної моделі:

– модульність: код розбивається на логічні блоки, що полегшує підтримку вебдодатку;

– повторне використання: компоненти можна використовувати повторно, що економить час та зусилля;

– тестування: компоненти можна легко тестувати окремо, що підвищує якість коду;

– гнучкість: компоненти можна легко змінювати та налаштовувати.

Компонентна модель – це основа React, яка робить його потужним та гнучким інструментом для створення вебдодатків. Розуміння цієї концепції є ключем до успішного використання React.

Virtual DOM – це ключова інновація React, яка значно покращує продуктивність та ефективність оновлення користувацького інтерфейсу. Він являє собою легковагову копію реального DOM, яка зберігається в пам'яті JavaScript [26]. Віртуальний DOM – це деревоподібна структура, де кожен вузол відповідає елементу реального DOM, але замість безпосередньо маніпулювати браузерними елементами, React працює з цим віртуальним представленням.

Коли відбуваються зміни в стані компонента або його властивостях, React не оновлює відразу реальний DOM. Замість цього, він створює нову версію віртуального DOM, що відображає бажаний стан інтерфейсу. Потім React запускає процес реконциліації, під час якого він порівнює нову версію віртуального DOM зі старою та визначає мінімально необхідні зміни, які потрібно внести в реальний DOM, щоб привести його у відповідність до нового стану [27].

Цей процес реконциліації є надзвичайно ефективним, оскільки React використовує розумні алгоритми для визначення найкоротшого шляху оновлення. Замість повторного рендеру всього дерева DOM, він оновлює лише ті елементи, які дійсно змінилися. Це дозволяє уникнути зайвих операцій з браузером, таких як перемальовування та перекомпонування елементів, що значно покращує продуктивність додатків, особливо при великій кількості елементів на сторінці.

Віртуальний DOM також спрощує процес розробки, оскільки розробникам не потрібно поглиблюватись в деталі маніпуляцій з DOM. Вони можуть зосередитися на описі бажаного стану інтерфейсу в декларативному стилі, а React автоматично подбає про його оновлення.

Крім того, віртуальний DOM дозволяє React реалізувати Server-Side Rendering (SSR) та рендеринг на мобільних пристроях за допомогою React Native. При SSR React може рендерити компоненти на сервері та відправляти вже готовий HTML клієнту, що прискорює первинне завантаження сторінки та покращує SEO.

Загалом, віртуальний DOM – це ключова технологія, яка робить React таким потужним та ефективним інструментом для створення сучасних

вебдодатків. Це дозволяє отримати високу продуктивність та відкриває нові можливості для рендерингу на різних платформах.

Односпрямований потік даних (Unidirectional Data Flow) – це ще одна ключова концепція React, яка забезпечує передбачуваність та керованість стану додатків. Вона визначає, як дані передаються та оновлюються між компонентами, створюючи чітку ієрархію та мінімізуючи можливість виникнення побічних ефектів.

У React дані в компоненті передаються від батьківського до дочірнього через властивості (props). Дочірній компонент не може безпосередньо змінити отримані властивості, що робить їх незмінними (immutable). Якщо дочірній компонент потребує змінити дані, він повідомляє батьківському компоненту, який, у свою чергу, оновить свій стан та передасть нові властивості дочірньому компоненту.

Такий підхід створює односпрямований потік даних, де зміни завжди ініціюються зверху вниз по ієрархії компонентів [28]. Це робить стан додатку більш передбачуваним та легким для розуміння, оскільки завжди зрозуміло, звідки надходять дані, коли і чому вони оновлюються, тощо.

Односпрямований потік даних також спрощує налагодження додатків. Якщо виникає проблема, пов'язана з даними, можна легко відстежити шлях їх передачі та визначити, який компонент відповідальний за їх оновлення.

Переваги односпрямованого потоку даних:

– передбачуваність: стан додатку стає більш передбачуваним, оскільки зміни завжди ініціюються зверху вниз по ієрархії компонентів;

– керованість: легше відстежувати та контролювати зміни даних, оскільки вони завжди проходять через батьківські компоненти;

– налагодження: простіше знаходити та виправляти помилки з даними, оскільки шлях їх передачі є чітко визначеним;

– масштабованість: односпрямований потік даних добре масштабується при збільшенні складності додатків.

Реалізація односпрямованого потоку даних:

Для реалізації односпрямованого потоку даних у React використовуються

такі інструменти та підходи:

– Props: передача даних в компоненті з батьківського до дочірнього;

– State: зберігання стану компонента та його оновлення через метод setState();

– Callbacks: передача функцій зворотного виклику з батьківського до дочірнього компонента для оновлення стану батьківського компонента;

– Context API: глобальне управління станом для передачі даних між компонентами, які не мають прямого зв'язку батько-дитина;

– бібліотеки управління станом (Redux, MobX): для управління складним станом додатків та забезпечення централізованого сховища даних.

Односпрямований потік даних є фундаментальним принципом React, який забезпечує передбачуваність, керованість та масштабованість додатків.

React вражає своєю гнучкістю та відкритістю, що проявляється у його здатності до інтеграції з широким спектром відомих та корисних бібліотек. Це дозволяє розробникам розширювати функціональність React та створювати потужні та комплексні вебдодатки, використовуючи найкращі інструменти та практики [29].

React сам по собі не має вбудованих механізмів для управління складним станом додатків. Однак, він легко інтегрується з бібліотеками управління станом, такими як Redux та MobX. Redux пропонує централізоване сховище стану та передбачуваний спосіб його оновлення через дії (actions) та редуктори (reducers). MobX, з іншого боку, надає більш реактивний підхід, дозволяючи оголошувати спостережувані дані та автоматично оновлювати компоненти при їх зміні.

Для створення односторінкових додатків з навігацією між різними маршрутами (сторінками) React легко інтегрується з бібліотеками маршрутизації, такими як React Router. Він дозволяє визначати маршрути, пов'язувати їх з компонентами та реалізовувати різні стратегії навігації, такі як переходи з анімацією або захист маршрутів.

Для отримання даних з сервера та виконання запитів React використовує додаткові бібліотеки, наприклад Axios. Axios – це популярна бібліотека, яка спрощує виконання HTTP-запитів та обробку відповідей сервера. Fetch API – це

вбудований у браузер інтерфейс для роботи з мережевими запитами.

React надає інструменти для тестування компонентів, наприклад React Testing Library (RTL) та Jest. RTL дозволяє тестувати компоненти як би це потенційно міг зробити користувач. Jest –надає багато можливостей для написання юніт-тестів, інтеграційних тестів та інших видів тестів.

Для прискорення розробки та забезпечення єдиного стилю інтерфейсу React може використовувати готові UI-бібліотеки та дизайн-системи, такі як Material UI, Ant Design або Semantic UI. Ці бібліотеки надають набір готових компонентів, які відповідають певним дизайн-принципам та мають узгоджений вигляд [30].

Крім перерахованих вище, існує багато бібліотек що використовуються з React для вирішення багатьох можливих завдань. Наприклад, Formik для зручної розробки форм, React Hook Form для валідації форм, React Intl для інтернаціоналізації, React Query для кешування даних та ін.

Переваги інтеграції з бібліотеками:

– розширення функціональності: бібліотеки дозволяють додавати нові можливості до React додатків без необхідності писати заново вже готовий функціонал;

– прискорення розробки: готові компоненти та інструменти з бібліотек скорочують час розробки;

– підвищення якості коду: бібліотеки часто містять добре протестований та оптимізований код, що покращує якість додатків;

– спрощення підтримки: використання популярних бібліотек полегшує підтримку коду та оновлення додатків.

Інтеграція з відомими бібліотеками є однією з переваг React, яка робить його універсальним інструментом для створення різноманітних вебдодатків.

2.2 Виконання JavaScript в середовищі NodeJS

Node.js – це справжній прорив у світі веброзробки, який переніс JavaScript з браузера на сервер. Він відкрив нові горизонти для створення швидких, масштабованих та ефективних серверних додатків, використовуючи знайомий та універсальний JavaScript.

Node.js побудований на рушії V8 JavaScript Engine від Google, тому він успадкував його високу продуктивність та оптимізації. Але головна його особливість полягає в асинхронній, неблокуючій моделі введення-виведення, яка дозволяє обробляти багато запитів одночасно без блокування виконання. Це досягається завдяки подійного циклу (event loop) та колбеків (callbacks), які дозволяють виконувати операції введення-виведення у фоновому режимі, не зупиняючи основний потік виконання [31].

Node.js з'явився на світ у 2009 р. завдяки Райану Далу, який прагнув створити платформу для побудови масштабованих мережевих додатків за допомогою JavaScript. Він використав рушій V8 JavaScript Engine від Google, який відомий своєю високою продуктивністю виконання коду, завдяки інноваційній неблокуючій моделі, введення-виведення та можливості використовувати JavaScript на стороні вебсервера.

NPM (Node Package Manager) – це найбільший у світі реєстр програмного забезпечення, який містить сотні тисяч модулів для Node.js, що покривають практично всі аспекти веброзробки. Ця величезна екосистема є результатом активної спільноти яка постійно створюює нові інструменти [32].

Переваги Node.js:

– висока продуктивність: завдяки неблокуючій моделі та використанню рушія V8, Node.js забезпечує високу швидкість обробки запитів;

– легкість розробки: використання JavaScript як єдиної мови програмування для клієнтської частини та серверної частини, спрощує розробку та дозволяє використовувати одні й ті ж знання та навички;

– масштабованість: Node.js масштабується, додаючи нові сервери для обробки більшої кількості запитів;

– велика екосистема: NPM надає доступ до величезної кількості готових бібліотек, що прискорюють розробку.

Node.js підходить для мережевих додатків реального часу, таких як чати, онлайн ігри, системи співпраці та інші додатки, які вимагають швидкої обробки великої кількості одночасних з'єднань. Він також широко використовується для створення API, мікросервісів, серверів для вебсокетів та інших серверних компонентів.

Найпопулярнішим фреймворком є Express.js, це мінімалістичний фреймворк для створення вебсерверів та API, NestJS – фреймворк, що базується на TypeScript та використовує архітектуру, натхненну Angular, Koa.js – наступне покоління фреймворку від команди Express, та інші.

Node.js також активно використовується для створення серверних додатків з GraphQL – мови запитів та середовища виконання для API, яка дозволяє визначати, які дані потрібно отримати в одному запиті.

Node.js не підходить для додатків, які вимагають інтенсивних обчислень на процесорі, оскільки JavaScript є однопотоковою мовою. Однак, для більшості вебдодатків, які в основному займаються введенням-виведенням (робота з мережею, базами даних), Node.js є чудовим вибором.

Node.js продовжує розвиватися та вдосконалюватися. З'являються нові версії з новим функціоналом та покращеннями продуктивності. Екосистема Node.js також постійно розширюється, з'являються нові модулі, фреймворки та інструменти. Це робить Node.js одним з найперспективніших та затребуваних інструментів у світі веброзробки.

У світі веброзробки створення повноцінних та функціональних додатків вимагає гармонійної взаємодії між двома ключовими компонентами: фронтендом (client-side) та бекендом (server-side). Фронтенд відповідає за візуальну частину додатку, з якою взаємодіє користувач, тоді як бекенд забезпечує обробку даних, логіку та взаємодію з сервером. Для досягнення цієї

синергії, сучасні технології, такі як React та Node.js, пропонують потужний та ефективний тандем.

React надає розробникам інструменти для побудови інтерактивних та динамічних фронтендів. Він дозволяє розробляти невеликі за обсягом компоненти, які відображають дані, реагують на дії користувача та оновлюють інтерфейс без необхідності перезавантаження сторінки [33].

Node.js, у свою чергу, розкриває можливості JavaScript на стороні сервера, забезпечуючи обробку запитів від клієнта, взаємодію з даними, виконання бізнес-логіки та інші серверні операції.

Інтеграція React та Node.js дозволяє робити повноцінні вебдодатки, де React відповідає за візуальне представлення та взаємодію з користувачем, а Node.js забезпечує логіку та обробку даних на сервері.

Один з найпоширеніших підходів – це використання REST API (Representational State Transfer Application Programming Interface). Node.js створює сервер, який надає REST API, а React робить запити до цього API для отримання та відправки даних. Цей підхід дозволяє чітко розділити відповідальність між серверною та клієнтською частиною додатку, забезпечуючи гнучкість та масштабованість [34].

Для додатків реального часу, які вимагають постійного обміну даними в реальному часі можна використовувати WebSockets. Node.js може створювати WebSocket сервер, а React – підключатися до нього та обробляти отримані повідомлення.

Для покращення SEO слід використовувати Server-Side Rendering (SSR), коли Node.js рендерить компоненти React на сервері та відправляє вже готовий HTML клієнту. Це дозволяє прискорити первинне завантаження сторінки та покращити індексацію пошуковими системами.

Інтеграція React та Node.js забезпечують розробку потужних та масштабованих вебдодатків з інтерактивним інтерфейсом та складною серверною логікою. Цей підхід відкриває широкі можливості для розробників та дозволяє реалізовувати найсміливіші ідеї у світі веброзробки.

2.3 Використання штучного інтелекту у веброзробці

Штучний інтелект (ШІ) стрімко змінює ландшафт веброзробки, пришвидшуючи створення інтерактивних, персоналізованих та інтелектуальних додатків. Інтеграція ШІ у вебдодатки дозволяє розробникам вирішувати складні завдання, автоматизувати процеси та надавати користувачам унікальний досвід взаємодії.

Зростаюча популярність ШІ у веброзробці обумовлена кількома факторами:

– технологічний прогрес: розвиток машинного навчання, обробки природної мови, ШІ зробив їх доступнішими та ефективнішими для використання у вебдодатках;

– збільшення обчислювальних потужностей: сучасні хмарні платформи та потужні процесори дозволяють виконувати складні обчислення ШІ безпосередньо у браузері або на сервері;

– попит на персоналізацію: користувачі очікують від вебдодатків персоналізованого досвіду, а ШІ дозволяє аналізувати поведінку користувачів та надавати їм релевантний контент та рекомендації;

– автоматизація рутинних завдань: ШІ автоматизує одноманітні завдання, такі як обробка даних, генерація контенту, модерація коментарів та інші, що звільняє час розробників для вирішення більш творчих завдань.

Одним з яскравих прикладів використання ШІ у веброзробці є інтеграція API GPT (Generative Pre-trained Transformer) для генерації питань до користувача. GPT – це потужна модель обробки природної мови, яка здатна генерувати тексти, схожі на людські, відповідати на питання, перекладати мови, тощо.

Інтеграція GPT API у вебдодаток дозволяє створювати інтерактивні опитування, вікторини, чат-ботів, де питання генеруються автоматично на основі контексту та попередніх відповідей користувача.

Інші приклади використання ШІ у веброзробці:

– чат-боти: ШІ-чат-боти можуть відповідати на питання користувачів,

надавати підтримку, приймати замовлення, замінюючи або доповнюючи роботу операторів;

– персоналізація контенту: ШІ може аналізувати поведінку користувачів та надавати їм персоналізовані рекомендації щодо товарів, послуг, контенту та інших аспектів;

– аналіз даних: завдяки здатності обробляти величезні масиви інформації, штучний інтелект виявляє приховані взаємозв'язки та тренди, надаючи цінні поради для прийняття стратегічних рішень та оптимізації бізнес-процесів.

ШІ продовжує розвиватися, вплив на веброзробку буде тільки зростати, може змінити спосіб створення вебдодатків, автоматизуючи рутинні завдання та дозволяючи розробникам зосередитися на творчих аспектах.

У другому розділі проаналізовано вибір інструментів для розробки вебдодатку. React та Node.js – це потужний симбіоз, що дозволяє створювати сучасні, високопродуктивні та масштабовані вебдодатки. React, з його компонентним підходом, віртуальним DOM та односпрямованим потоком даних, забезпечує зручність розробки складних інтерфейсів, тоді як Node.js надає гнучкість та ефективність для обробки серверної логіки. Їхня спільна робота відкриває нові горизонти у веброзробці, дозволяючи створювати додатки, що відповідають найвищим вимогам сучасного користувача.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ ДЛЯ ПІДГОТОВКИ ДО СПІВБЕСІДИ В ІТ-КОМПАНІЮ

3.1 Розробка клієнтської частини вебдодатку

Для швидкого старту розробки клієнтської частини було обрано інструмент Vite. Це сучасний інструмент збірки, який значно перевершує Create React App (CRA) за швидкістю завдяки використанню нативних ES-модулів у браузері під час розробки та оптимізованій збірці (рис. 3.1). Vite дозволяє миттєво запускати сервер розробки, оминаючи попередню збірку, що значно прискорює початок розробки проекту [35]. Крім того, Vite підтримує гаряче перезавантаження (HMR), що забезпечує миттєве оновлення змін у коді без необхідності перезавантажувати всю сторінку.

```
PS C:\Users\mixno\OneDrive\Робочий стіл\PDAU> npm create vite
√ Project name: ... DIPLOM
√ Package name: ... diplom
? Select a framework: » - Use arrow-keys. Return to submit.
>   Vanilla
    Vue
    React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

Рисунок 3.1 – Створення проекту за допомогою Vite

Структура файлів у вебдодатках React гнучка і може бути адаптована під потреби проекту та стиль роботи команди розробників. Незважаючи на це, основні елементи та загальна організація зазвичай зберігають певну подібність (рис. 3.2).

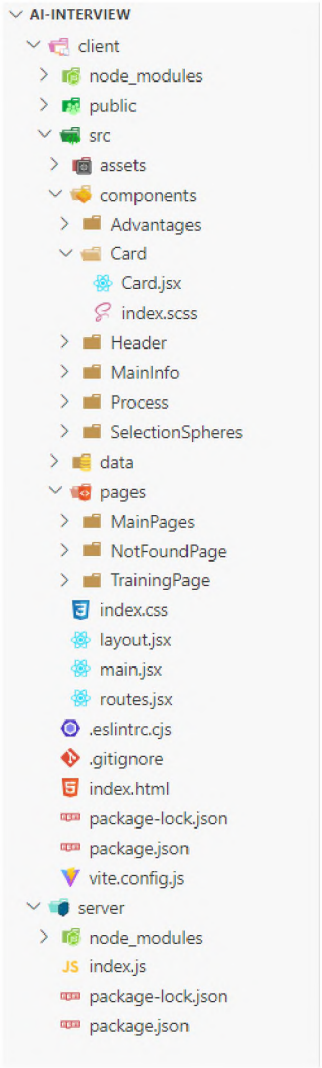

Рисунок 3.2 – Файлова структоура в проекті

Основні компоненти структури є:

– public/: ця директорія слугує сховищем для статичних ресурсів, таких як головна сторінка index.html, іконка favicon.ico, зображення, шрифти та інші файли, які браузер має безпосередньо завантажувати та відображати;

– src/: це серце React, тут знаходиться весь вихідний код. Тут будуть створюватись компоненти, логіка, стилі та інші файли, необхідні для роботи додатку;

– main.jsx (або main.tsx): цей файл служить точкою входу для додатку. Він зазвичай містить імпорт необхідних бібліотек, створення кореневого компонента та його рендеринг в DOM;

– components/: у цій директорії будуть зберігатись багаторазові компоненти вебдодатку. Кожен компонент, як правило, має власний файл, який містить його логіку, структуру, стилі;

– pages/: якщо додаток має кілька сторінок або маршрутів, ця директорія може бути корисною для організації компонентів, відповідальних за відображення та функціональність кожної сторінки;

– assets/: тут зберігаються різноманітні ресурси, зображення, відео, аудіофайли, шрифти та інші;

– utils/: ця директорія часто використовується для зберігання допоміжних функцій, утиліт та інших модулів, які не є компонентами, але використовуються в різних частинах вебдодатку;

– styles/: місцем для зберігання глобальних стилів;

– tests/: у цій директорії можуть зберігатись тести для додатку, якщо використовувати в проекті інструменти тестування;

– config/: тут можна зберігати конфігураційні файли, які містять налаштування додатку.

Ця структура є лише відправною точкою, її можна вільно адаптувати під потреби проекту.

Структура клієнтської частини вебдодатку складається з декількох ключових компонентів, кожен з яких відповідає за певну функціональність.

Header це невід'ємна частина будь-якого вебдодатку, забезпечуючи навігацію та відображення важливої інформації (рис. 3.3 – 3.4).

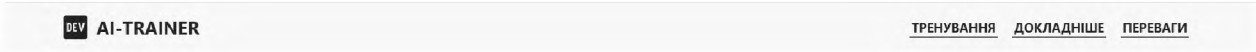

Рисунок 3.3 – Вигляд Header на сайті

```jsx
export function Header() {

    return (
        <header className={style.header}>
            <div className='container'>
                <div className={style.header__wrapper}>
                    <div className={style.header__logo}>
                        <FaDev size={'30px'}/>
                        <h2>
                            <Link to="/">AI-Trainer</Link>
                        </h2>
                    </div>
                    <div className={style.header__links}>
                        <Link className={style.header__item} to="/training">Тренування</Link>
                        <Link className={style.header__item} to="/info">Докладніше</Link>
                        <Link className={style.header__item} to="/advantages">Переваги</Link>
                    </div>
                </div>
            </div>
        </header>
    );
}
```

Рисунок 3.4 – Код реалізації компоненту Header

У даному проекті для реалізації навігації в Header використовується бібліотека React Router DOM, що дозволяє створювати декларативні маршрути та посилання між різними сторінками застосунку, забезпечуючи зручний та інтуїтивно зрозумілий спосіб переміщення користувача по сайту (рис. 3.5).

```jsx
export const router = createBrowserRouter([
    {
        path: '/',
        element: <Layout/>,
        children: [
            {
                path: '/',
                element: <MainPage/>
            },
            {
                path: '*',
                element: <NotFoundPage/>
            },
            {
                path: '/info',
                element: <Process/>
            },
            {
                path: '/advantages',
                element: <Advantages/>
            },
            {
                path: '/training',
                element: <SelectionSpheres/>
            },
            {
                path: '/training/:id',
                element: <TrainingCard/>
            },
        ]
    }
])
```

Рисунок 3.5 – Код реалізації роутингу в додатку

«Головна сторінка» слугує вхідною точкою для користувачів вебдодатку. Вона надає загальну інформацію про призначення платформи, її основні функції та можливості (рис. 3.6). Головна сторінка також містить елементи навігації, що дозволяють користувачам перейти до інших розділів застосунку, таких як тренування, профіль, статистика тощо.

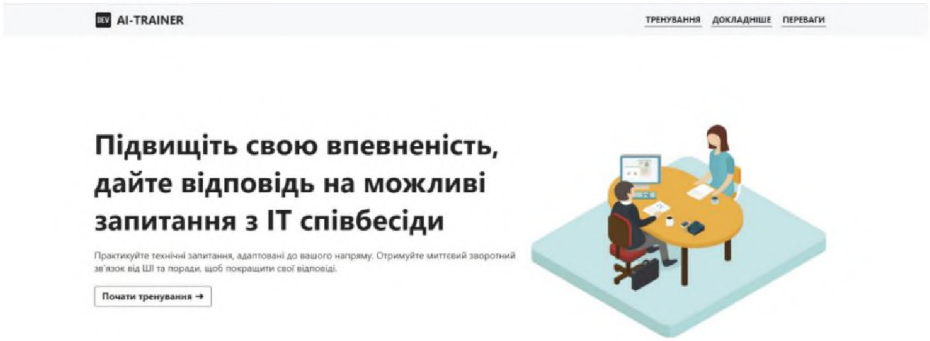

Рисунок 3.6 – Вигляд «Головної сторінки» сайту

Реалізація головної сторінки виконана за допомогою компонента MainInfo, який відповідає за відображення контенту та логіку цієї сторінки (рис. 3.7).

```
export function MainInfo() {

    return (
        <main className={style.info}>
            <div className={style.info__wrap}>
                <div className={style.info__text}>
                    <h1 className={style.info__header}>Підвищіть свою впевненість, дайте відповідь на можливі запитання з IT співбесіди</h1>
                    <p className={style.info__description}>Практикуйте технічні запитання, адаптовані до вашого напряму. Отримуйте миттєвий зв
                    <button className={style.info__button}>
                        <Link to={'/training'}>Почати тренування  &#10140;</Link>
                    </button>
                </div>
                    <img src="/public/main.png" alt="" className={style.info__img}/>
            </div>
        </main>
    )
}
```

Рисунок 3.7 – Код реалізації компоненту MainInfo

Сторінка «Докладніше» призначена для ознайомлення користувача з процесом використання платформи для підготовки до співбесід. Вона детально розкриває кроки, які необхідно виконати для ефективної підготовки, та демонструє переваги використання ШІ в цьому процесі (рис. 3.8).

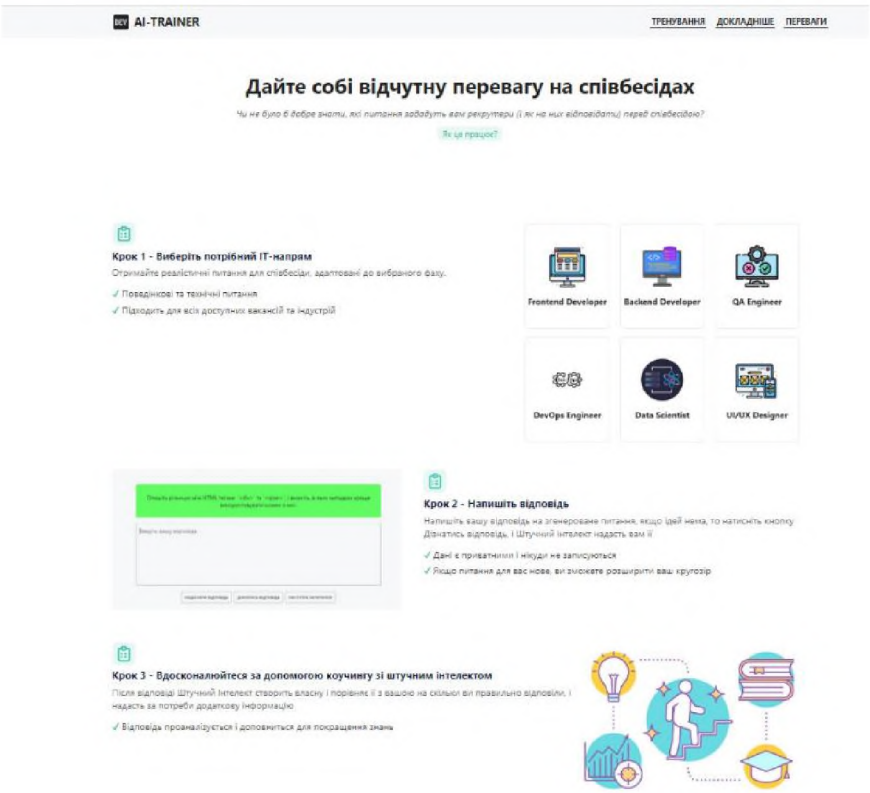

Рисунок 3.8 – Вигляд сторінки «Докладніше»

Сторінка складається з трьох основних блоків, кожен з яких описує окремий крок процесу:

Крок 1 – вибір IT-напряму: користувач обирає сферу діяльності, в якій він хоче пройти співбесіду;

Крок 2 – написання відповіді: користувач відповідає на згенероване питання або отримує підказку від штучного інтелекту;

Крок 3 – вдосконалення за допомогою навчання зі штучним інтелектом: Штучний інтелект аналізує відповідь користувача, порівнює її з правильною та надає рекомендації щодо покращення.

Реалізація сторінки докладніше виконана за допомогою компонента Process, який відповідає за відображення контенту та логіку цієї сторінки (рис. 3.9 – 3.10).

```
export function Process() {
    return (
        <div className={style.process}>
            <div className={style.process__header}>
                <div className={style.process__title}>Дайте собі відчутну перевагу на співбесідах</div>
                <div className={style.process__subtitle}>Чи не було б добре знати, які
                    питання зададуть вам рекрутери (і як на них відповідати) перед співбесідою?
                </div>
                <div className={style.process__label}>Як це працює?</div>

            </div>

            <div className={style.process__wrap}>
                <div className={style.process__step}>
                    <TbClipboardList className={style.process__step_icon}/>
                    <div className={style.process__step_header}>Крок 1 - Виберіть потрібний IT-напрям</div>
                    <div className={style.process__step_description}>Отримайте реалістичні питання для співбесіди, адаптовані до вибраного фаху.</div>
                    <div className={style.process__step_li}>
                        <span>&#10004;</span>Поведінкові та технічні питання
                    </div>
                    <div className={style.process__step_li}>
                        <span>&#10004;</span>Підходить для всіх доступних вакансій та індустрій
                    </div>
                </div>
                <div className={style.process__preview}>
                    <img src="/public/sphere.png" width={'500px'}/>
                </div>
            </div>

            <div className={style.process__wrap}>

                <div className={style.process__preview}>
                    <img src="/public/question.jpg" width={'500px'}/>
                </div>
                <div className={style.process__step}>
                    <TbClipboardList className={style.process__step_icon}/>
                    <div className={style.process__step_header}>Крок 2 - Напишіть відповідь </div>
                    <div className={style.process__step_description}>Напишіть вашу відповідь на згенероване питання, якщо ідей нема, то натисніть кнопку Дізн
                    <div className={style.process__step_li}>
                        <span>&#10004;</span>Дані є приватними і нікуди не записуються
                    </div>
                    <div className={style.process__step_li}>
                        <span>&#10004;</span>Якщо питання для вас нове, ви зможете розширити ваш кругозір
                    </div>
                </div>
```

Рисунок 3.9 – Код реалізації сторінки «Докладніше»

```
            <div className={style.process__wrap}>
                <div className={style.process__step}>
                    <TbClipboardList className={style.process__step_icon}/>
                    <div className={style.process__step_header}>Крок 3 - Вдосконалюйтеся за допомогою коучингу зі штучним інтелектом</div>
                    <div className={style.process__step_description}>Після відповіді Штучний Інтелект створить власну і порівняє її з вашою на скільки ви правильно
                    <div className={style.process__step_li}><span>&#10004;
                        </span>Відповідь проаналізується і доповниться для покращення знань
                    </div>
                </div>
                <div className={style.process__preview}>
                    <img src="/public/step3.jpg" width={'400px'}/>
                </div>
            </div>
        </div>
    );
}
```

Рисунок 3.10 – Код реалізації сторінки «Докладніше»

Сторінка «Переваги» призначена для демонстрації користувачам переваг використання платформи для підготовки до співбесід. Вона порівнює стан кандидата до та після тренувань, підкреслюючи позитивний вплив використання сервісу на успішність співбесід (рис. 3.11). Реалізація сторінки «Переваги» виконана в компоненті Advantages (рис. 3.12).

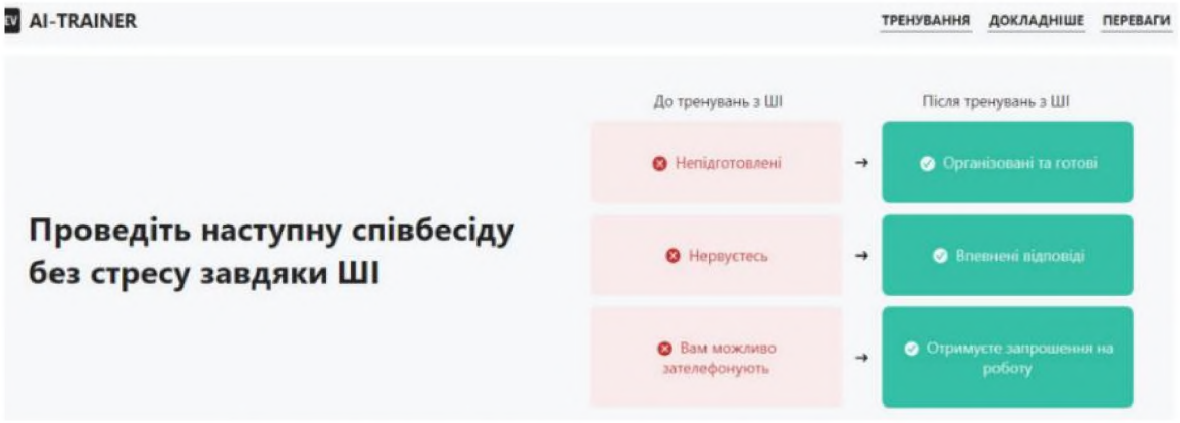

Рисунок 3.11 – Вигляд сторінки «Переваги»

```
export function Advantages() {

    return (
        <>
        <div className={style.advantages}>
                <div className={style.advantages__wrap}>
                    <div className={style.advantages__title}>Проведіть наступну співбесіду без стресу завдяки ШІ</div>
                    <div className={style.advantages__content}>
                        <div className={style.advantages__row}>
                            <div className={style.advantages__element}>До тренувань з ШІ</div>
                            <div className={style.advantages__element}>Після тренувань з ШІ</div>
                        </div>

                        <div className={style.advantages__row}>
                            <div className={`${style.advantages__element} ${style.negative}`}>
                                <IoMdCloseCircle className={style.advantages__svg} size={'1.3em'}/>
                                Непідготовлені
                            </div>
                            <div className={style.advantages__element}>&#10140;</div>
                            <div className={`${style.advantages__element} ${style.positive}`}>
                                <IoCheckmarkCircleSharp className={style.advantages__svg} size={'1.3em'}/>
                                Організовані та готові
                            </div>
                        </div>
                        <div className={style.advantages__row}>
                            <div className={`${style.advantages__element} ${style.negative}`}>
                                <IoMdCloseCircle className={style.advantages__svg} size={'1.3em'}/>
                                Нервуєтесь
                            </div>
                            <div className={style.advantages__element}>&#10140;</div>
                            <div className={`${style.advantages__element} ${style.positive}`}>
                                <IoCheckmarkCircleSharp className={style.advantages__svg} size={'1.3em'}/>
                                Впевнені відповіді
                            </div>
                        </div>
                        <div className={style.advantages__row}>
                            <div className={`${style.advantages__element} ${style.negative}`}>
                                <IoMdCloseCircle className={style.advantages__svg} size={'1.3em'}/>
                                Вам можливо зателефонують
                            </div>
                            <div className={style.advantages__element}>&#10140;</div>
                            <div className={`${style.advantages__element} ${style.positive}`}>
                                <IoCheckmarkCircleSharp className={style.advantages__svg} size={'1.3em'}/>
                                Отримуєте запрошення на роботу
                            </div>
```

Рисунок 3.12 – Код реалізації сторінки «Переваги»

Сторінка «Вибір сфери» є важливим етапом підготовки до співбесіди, оскільки дозволяє користувачу зосередитися на конкретній галузі знань.

На цій сторінці представлено набір карток, кожна відповідає певній IT-спеціальності (рис. 3.13).

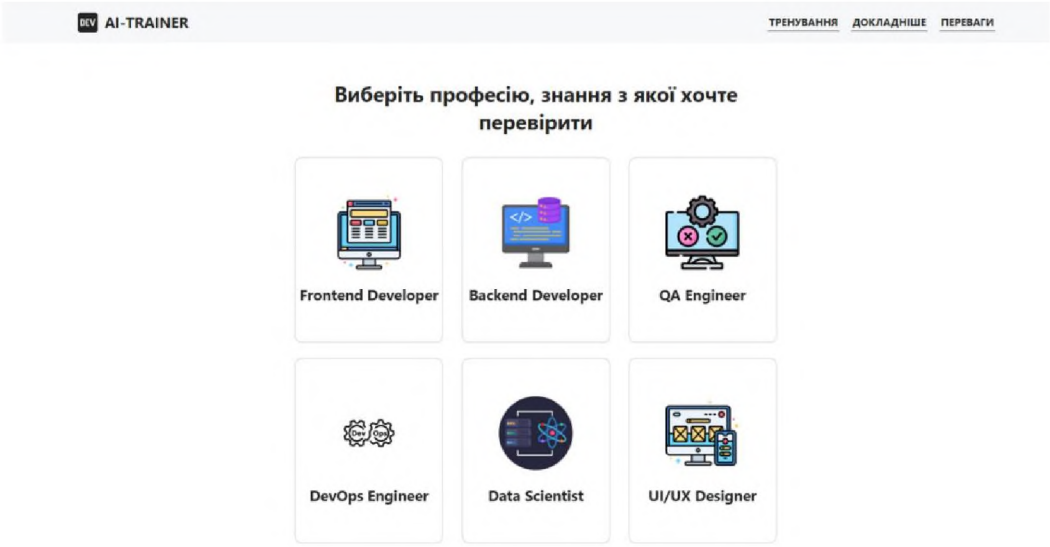

Рисунок 3.13 – Вигляд сторінки «Вибір сфери»

Користувач при наведенні курсору може ознайомитися з коротким описом кожної спеціальності та вибрати ту, яка його цікавить (рис. 3.14).

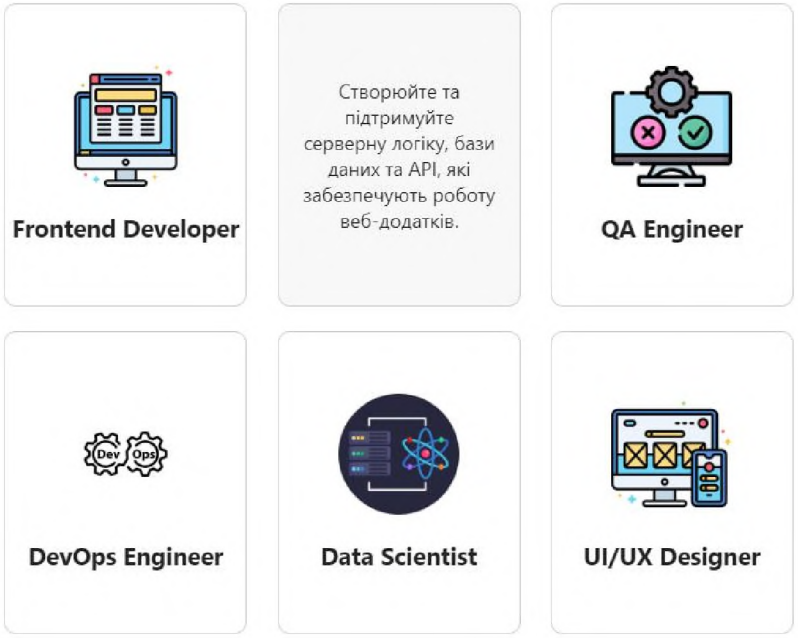

Рисунок 3.14 – Вигляд перегляду інформації при наведенні на картку

Основний код для відображення сторінки знаходиться в компоненті SelectionSpheres. В свою чергу цей компонент перебирає завдяки методу map всі

об’єкти що містяться в файлі data.json, що є локальною базою даних для зберігання карток спеціальностей (Рис) та відрисовує для кожного з них компонент Card (рис. 3.15 – 3.17).

```jsx
export function SelectionSpheres() {

    return (
        <div className={style.selection}>
                <div className={style.selection__container}>
                <div className={style.selection__title}>
                    Виберіть професію, знання з якої хочте перевірити
                </div>

                {
                  data.map((card) => (
                    <Link  key={card.id} to={`/training/${card.id}`}>
                        <Card
                            logo={card.logo}
                            title={card.title}
                            description={card.description}/>
                    </Link>
                  ))
                }
            </div>
        </div>
    )
}
```

Рисунок 3.15 – Код компоненту SelectionSpheres


```json
[
  {
    "id": "frontend-developer",
    "logo": "public/front.png",
    "title": "Frontend Developer",
    "description": "Створюйте візуально привабливі та інтерактивні користувацькі інтерфейси для веб-сайтів і веб-додатків."
  },
  {
    "id": "backend-developer",
    "logo": "public/back.png",
    "title": "Backend Developer",
    "description": "Створюйте та підтримуйте серверну логіку, бази даних та API, які забезпечують роботу веб-додатків."
  },
  {
    "id": "qa-engineer",
    "logo": "public/qa.png",
    "title": "QA Engineer",
    "description": "Забезпечення якості програмних продуктів шляхом розробки та виконання тестів, виявлення та повідомлення про помилки."
  },
  {
    "id": "devops-engineer",
    "logo": "public/devops.jpg",
    "title": "DevOps Engineer",
    "description": "Подолання розриву між розробкою та експлуатацією, автоматизація доставки програмного забезпечення та управління інфраструктурою."
  },
  {
    "id": "data-scientist",
    "logo": "public/data.png",
    "title": "Data Scientist",
    "description": "Отримуйте знання та інсайти з даних за допомогою статистичного аналізу, машинного навчання та візуалізації даних."
  },
  {
    "id": "ui-ux-designer",
    "logo": "public/design.png",
    "title": "UI/UX Designer",
    "description": "Створюйте інтуїтивно зрозумілі, візуально привабливі та зручні користувацькі інтерфейси та користувацький досвід."
  }
]
```


Рисунок 3.16 – Вигляд файду data.json

```jsx
export function Card({logo, title, description}) {
    return (
        <div className="card">
            <div className="card-inner">
                <div className="card-front">
                    <img src={logo} alt={title} className="card-logo"/>
                    <h3 className="card-title">{title}</h3>
                </div>
                <div className="card-back">
                    <p className="card-description">{description}</p>
                </div>
            </div>
        </div>
    );
}
```

Рисунок 3.17 – Код компоненту Card

Компонент TrainingCard відповідає за відображення та логіку сторінки тренування, де користувач може отримувати згенеровані запитання, відповідати на них та отримувати фідбек від штучного інтелекту.

Спочатку користувачеві відображається лише кнопка «Почати тренування» (рис. 3.18). При натисканні на цю кнопку відбуваються наступні дії:

1. Компонент переходить у стан очікування (isLoading встановлюється в true), і на екрані з'являється індикатор завантаження.
2. Відправляється запит на бекенд для отримання першого запитання.
3. Після отримання запитання воно відображається на екрані, а також з'являються елементи управління: поле для вводу відповіді та кнопки.

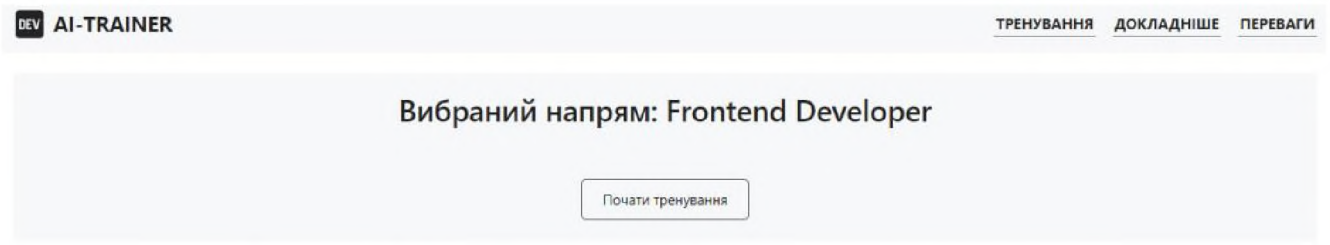

Рисунок 3.18 – Вигляд сторінки початку тренування

Кнопка «Надіслати відповідь» при натисканні на цю кнопку відповідь користувача відправляється на бекенд для порівняння з правильною відповіддю, штучний інтелект аналізує відповідь користувача та формує фідбек, який відображається на екрані. Фідбек може містити оцінку відповіді за шкалою від 1

до 10, пояснення правильної відповіді, а також рекомендації щодо покращення відповіді користувача (рис. 3.19).

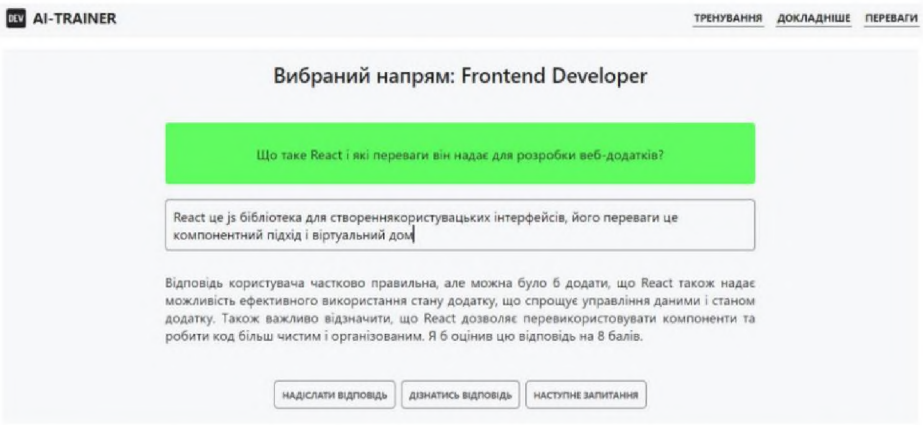

Рисунок 3.19 – Вигляд компонента тренування при надсиланні власної відповіді

Кнопка «Дізнатись відповідь» при натисканні на цю кнопку на бекенд відправляється запит на отримання правильної відповіді до поточного запитання, отримана відповідь відображається у полі вводу, що дозволяє користувачу побачити правильний варіант (рис. 3.20).

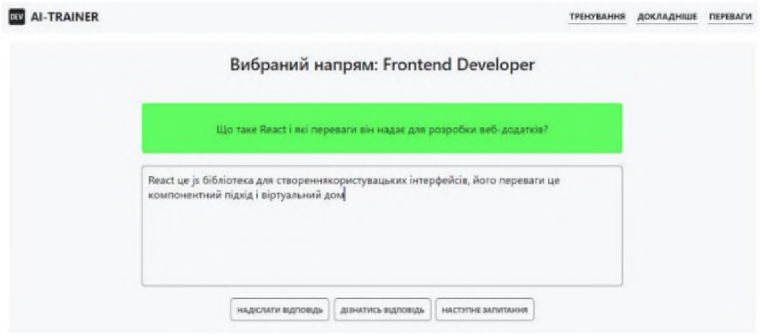

Рисунок 3.20 – Вигляд компонента тренування при натисканні кнопки «Дізнатись відповідь»

Кнопка «Наступне запитання» при натисканні на цю кнопку відбувається перехід до наступного запитання, поле вводу та фідбек очищуються, і процес повторюється з новим запитанням. За всю логіку роботи форми тренування відповідає компонент TrainingCard (рис. 3. 21).

```
export function TrainingCard() {
  const [selectedCard, setSelectedCard] = useState(null);
  const [showQuestion, setShowQuestion] = useState(false);
  const [question, setQuestion] = useState(null);
  const [answer, setAnswer] = useState('');
  const [feedback, setFeedback] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const { id } = useParams();

  useEffect(() => {
    const card = data.find(card => card.id === id);
    setSelectedCard(card);
  }, [id]);

  const fetchQuestion = async () => {
    setIsLoading(true);
    try {
      const response = await axios.post('http://localhost:3001/get-question', {
        prompt: `Напиши одне з найбільш поширених на IT співбесіді технічне запитання на позицію ${selectedCard.title}`
      });
      setQuestion(response.data.question);
    } catch (error) {
      console.error('Помилка при отриманні запитання:', error);
      setError('Не вдалося отримати запитання. Спробуйте пізніше.');
    } finally {
      setIsLoading(false);
    }
  };

  const handleAnswerSubmit = async () => {
    setIsLoading(true);
    try {
      const response = await axios.post('http://localhost:3001/compare-answers', {
        question,
        userAnswer: answer,
      });
      setFeedback(response.data.feedback);
    } catch (error) {
      console.error('Помилка при відправці відповіді:', error);
      setError('Не вдалося відправити відповідь. Спробуйте пізніше.');
    } finally {
```

Рисунок 3.21 – Фрагмент коду компонента TrainingCard

Сторінка 404 – це стандартна сторінка, яка відображається, коли користувач намагається перейти на неіснуючу сторінку сайту. Вона повідомляє користувача про те, що сторінка не знайдена, та пропонує повернутися на головну сторінку (рис. 3. 22).

Рисунок 3.22 – Вигляд сторінки яка відображається коли вказаний невідомий маршрут

Сторінка помилки 404 реалізована в типовому компоненті NotFoundPage, який забезпечує стандартний сценарій.

3.2 Розробка серверної частини вебдодатку

Серверна частина застосунку, розроблена на Node.js з використанням фреймворку Express.js, забезпечує основний функціонал платформи для підготовки до співбесід. Node.js обрано завдяки його високій продуктивності та здатності ефективно обробляти численні одночасні запити, а також завдяки багатьом бібліотекам. Express.js спрощує створення маршрутів, обробку запитів та відповідей, що робить розробку серверної логіки більш зручною та ефективною.

Також для реалізації взаємодії із Штучним Інтелектом потрібно отримати API ключ для можливості робити запити до ChatGPT через розроблений вебдодаток (рис. 3.23).

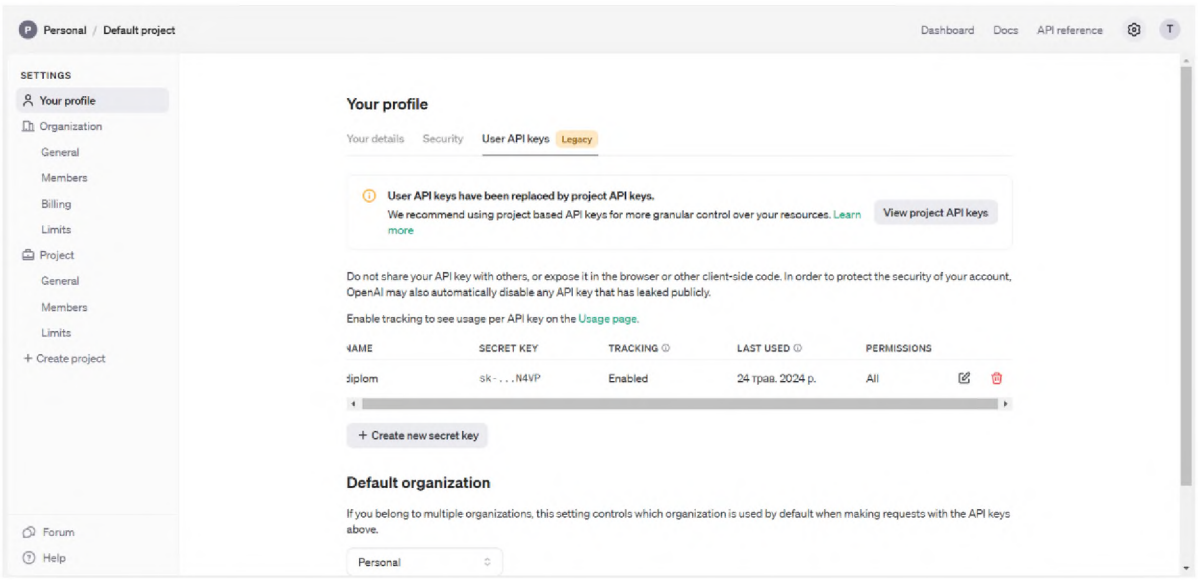

Рисунок 3.23 – Сторінка отримання ключа для доступудо GPT API

У коді серверної частини спочатку імпортуються необхідні залежності:

– cors для налаштування Cross-Origin Resource Sharing, що забезпечує безпечний обмін даними;

– express як основний фреймворк для створення вебсервера;

– OpenAI, офіційна бібліотека для взаємодії з API OpenAI, включаючи модель GPT-3.5 (рис. 3.24).

```
import cors from 'cors';
import express from 'express';
import OpenAI from 'openai';
```

Рисунок 3.24 – Імпорт потрібних модулів для роботи

Далі відбувається ініціалізація сервера: створюється екземпляр Express-застосунку (app), встановлюється порт для прослуховування (3001) та ініціалізується змінна apiKey з вашим секретним ключем доступу до OpenAI API. Використання app.use(express.json()) дозволяє серверу коректно обробляти вхідні дані у форматі JSON (рис. 3.25).

```
const app = express();
const port =  3001;
const apiKey = 'sk-IQPuSz6a0RkOMld3gJ4CT3BlbkFJzAxpK7

app.use(express.json());
app.use(cors())

const openai = new OpenAI({
  apiKey: apiKey,
});
```

Рисунок 3.25 – Підключення CORS та ініціалізація порту та ключа

Наступним кроком створюється клієнт OpenAI (openai) з обліковими даними, що дає змогу взаємодіяти з API OpenAI та використовувати його потужні можливості для генерації відповідей. Сервер визначає три основні маршрути для обробки запитів від клієнта (рис. 3.26).

```
app.post('/get-question', async (req, res) => {
  try {
    const { prompt } = req.body;

    const chatCompletion = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [{ role: "user", content: prompt }],
    });

    const question = chatCompletion.choices[0].message.content;

    res.json({ question });
  } catch (error) {
    console.error('Error getting question:', error);
    if (error.response) {
      console.error(error.response.status, error.response.data);
      res.status(error.response.status).json(error.response.data);
    } else {
      res.status(500).json({ error: 'An error occurred during question generation' });
    }
  }
});
```

Рисунок 3.26 – Код маршруту для отримання запитання з серверу

Маршрут POST /get-question приймає промпт від клієнта, який містить опис категорії запитань, і повертає згенероване GPT-3.5 запитання у форматі JSON. Маршрут POST /get-answer приймає текст запитання та повертає коротку та точну відповідь, згенеровану GPT-3.5, також у форматі JSON (рис. 3.27).

```
app.post('/get-answer', async (req, res) => {
  try {
    const { question } = req.body;

    const completion = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        { role: "user", content: `Дай коротку та точну відповідь на це запитання. ${question}` },
      ],
    });

    const answer = completion.choices[0].message.content;

    res.json({ answer });
  } catch (error) {
    console.error('Помилка при отриманні відповіді:', error);
    res.status(500).json({ error: 'Помилка при отриманні відповіді' });
  }
});
```

Рисунок 3.27 – Код маршруту для отримання правильної відповіді з серверу

Маршрут POST /compare-answers приймає текст запитання та відповідь користувача, порівнює їх з правильною відповіддю, отриманою від GPT, та повертає фідбек у вигляді оцінки та рекомендацій щодо покращення відповіді (рис. 3.28).

```
app.post('/compare-answers', async (req, res) => {
  try {
    const { question, userAnswer } = req.body;

    const correctAnswerCompletion = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        { role: "user", content: `Дай коротку та точну відповідь на це запитання. ${question}` },
      ],
    });
    const correctAnswer = correctAnswerCompletion.choices[0].message.content;

    const feedbackCompletion = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "user",
          content: `Запитання: ${question}. Порівняй правильну відповідь: ${correctAnswer} та відповідь що вказав користувач ${userAnswer} та оціни на с
        },
      ],
    });


    const feedback = feedbackCompletion.choices[0].message.content;

    console.log(feedback)

    res.json({ feedback });
  } catch (error) {
    console.error('Помилка при порівнянні відповідей:', error);
    res.status(500).json({ error: 'Помилка при порівнянні відповідей' });
  }
});
```

Рисунок 3.28 – Код маршруту для порівняння правильної відповіді і від користувача

Кожен з цих маршрутів містить обробку можливих помилок, у разі помилки, сервер повертає відповідний код статусу та повідомлення про помилку.

Нарешті, сервер запускається на вказаному порту (3001) та виводить повідомлення в консоль про успішний запуск (рис. 3.29).

```
app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

Рисунок 3.29 – Код запуску роботи сервера

Ця конфігурація дозволяє фронтенду взаємодіяти з бекендом, відправляючи запити на ці маршрути та отримуючи відповіді у форматі JSON, що забезпечує повноцінну роботу вебдодатку.

3.3 Економічне обґрунтування

Розробка дипломного проекту складається з наступних етапів:

– проектування – постановка задачі, розробка технічного завдання та розробка алгоритмів вирішення (18 % від загального часу);

– впровадження – безпосереднє впровадження спроектованої системи (61 % від загального часу);

– тестування – виявлення дефектів функцій, логіки та форми реалізації з подальшим усуненням (11 % від загального часу);

– завершально-підсумкові контрольні роботи (10 % загального часу).

– Діаграма розподілу часу розробки вебсайту для сервісного центру у відсотковому відношенні представлена згідно з рис. 3.30.

Загальний термін розробки вебдодатку для підготовки до співбесіди в IT компанію – два місяці.

Рисунок 3.30 – Діаграма розподілу часу на розробку вебдодатку

Розрахуємо всі можливі витрати, що слід враховувати при розробці вебдодатку. Вартісна оцінка включає:

– ресурси та комплектуючі (М);

– електроенергія (Е);

– основна заробітна плата розробників (Зо);

– додаткова заробітна плата розробників (ЗД);

– відрахування до фонду соціального захисту населення та на обов'язкове страхування(Ссн);

– амортизація обладнання та програмного забезпечення (А);

– накладні витрати (РН).

Вартість матеріалів (М), необхідних розробки програмного забезпечення представлена відповідно до табл. 3.1.

Загальна вартість ресурсів, необхідні вебдодатку 2200 грн:

$$M = s_1 + s_2 + s_3 + s_4 = 700 + 600 + 800 + 100 = 2200 \text{ грн}, \quad (3.1)$$

де s_1 – вартість хостингу;

s_2 – вартість домену;

s_3 – вартість використаного інтернету;

s_4 – вартість реєстрації домену.

Витрати на електроенергію (Е), необхідні розробки програмного забезпечення склали 193 грн. За місяць витрати електроенергії становили 12,5 кВт. За два місяці витрати склали 2×12,5 = 25 кВт. Базовий тариф становить 7,7 грн за 1 кВт/рік.

$$E = K_e \times T_p, E = 7,7 \times 192,7 = 193, \quad (3.2)$$

де K_e – вартість одного кВт /год, грн;

T_p – кількість кВт/год.

Таблиця 3.1 – Вартість ресурсів, необхідних розробки вебдодатків

Найменування ресурсів	Кількість, шт.	Ціна, грн	Сума, грн
Хостинг	1	700	700
Домен	1	600	600
Інтернет	1	800	800
Реєстрація домену	1	100	100
Разом	×	×	2200

Основною статтею витрат за створення вебдодатку для підготовки до співбесіди в ІТ компанію є заробітна плата ($З_о$) розробника. Основна заробітна плата розраховується за формулою:

$$З_о = ТС_і \times К_і \times n \times t = 7952 \times 1,2 \times 1 \times 2 = 19085 \text{ грн}, \quad (3.3)$$

де $ТС_і$ – тарифна ставка фахівця і-го розряду, грн;

$К_і$ – коефіцієнт преміювання ($К_і$ = 1,2);

n – кількість виконавців, які брали участь у розробці програмного забезпечення, чол. (одна людина);

t – кількість часу, витрачене розробку інтернет магазину сільськогосподарських товарів, місяців (два місяці).

Тарифна ставка і-го розряду ($ТС_і$) розраховується за такою формулою:

$$ТС_і = ТС_1 \times ТК = 2800 \times 2,84 = 7952 \text{ грн}, \quad (3.4)$$

де $ТС_1$ – діюча тарифна ставка 1-го розряду;

ТК – тарифний коефіцієнт згідно з ЄТС становить 2,84.

Додаткова заробітна плата ($З_д$) розраховується за формулою:

$$З_д = \frac{З_о \times Нд}{100\%} = \frac{19085{,}00 \times 20\%}{100\%} = 3816{,}96 грн = 3817 грн. \quad (3.5)$$

де $З_д$ – додаткова заробітна плата виконавців на конкретне програмне забезпечення, грн;

Нд – норматив додаткової заробітної плати (20 %) становить:

$$З_д = \frac{19085{,}00 \times 20\%}{100\%} = 3817 грн. \quad (3.6)$$

Розраховуємо суму відрахувань на соціальні потреби ($З_{сз}$):

$$З_{сз} = \frac{(З_д + З_о) \times Н_{сз}}{100\%} = \frac{(19085+3817) \times 35\%}{100\%} = 8016 грн, \quad (3.7)$$

де $Н_{сз}$ – норматив відрахувань до Фонду соціального захисту населення та відрахувань на обов'язкове страхування (35 %).

Вартість основних засобів (С) та нематеріальних активів, що використовуються в процесі розробки:

– основні засоби – комп’ютер Dell Inspirion 7577 (52000 грн);

– нематеріальні активи – Microsoft Office 2010 Professional (21005 грн) та Adobe Photoshop (25900 грн).

Розрахуємо вартість основних засобів та нематеріальних активів:

$$З = 52000 + 21005 + 25900 = 98905 грн. \quad (3.8)$$

Норма амортизації для лінійного способу нарахування обчислюється за такою формулою:

$$Н_а = \frac{1}{Т_С} \times 100\% = \frac{100\%}{5} = 20\%, \quad (3.9)$$

де $Т_с$ – термін експлуатації обладнання.

Розрахуємо амортизаційні відрахування за п'ять років.

Розрахувати амортизацію за рік (А):

$$А = \frac{С \times Н_а}{100\%} = \frac{98905 \times 20\%}{100\%} = 19781 грн. \quad (3.10)$$

Розрахуємо амортизаційні відрахування за місяць ($А_1$):

$$А = \frac{С \times Н_а}{12} = \frac{19781}{12} = 1488{,}5 грн. \quad (3.11)$$

Розрахувати амортизацію за два місяці ($А_2$).

$$А_2=1488,5\times2=2977 \text{ грн.} \quad (3.12)$$

Накладні витрати ($Р_н$), які відносяться до розробленого вебдодатку за нормативом ($Н_р$) у відсотках до основної заробітної плати виконавця, визначатимуться за формулою:

$$Р_н=З_о\times\frac{Н_р}{100\%}=19085\times\frac{10\%}{100\%}=1909 \text{ грн,} \quad (3.13)$$

де $Р_н$ – накладні витрати на конкретне ПЗ, грн;

$Н_р$ – стандартні накладні витрати – 10 %.

Загальна вартість згідно кошторису (планова вартість (С)) на вебдодаток розраховується за формулою:

$$С=М+Е+З_о+З_д+З_{сз}+А+Р_н. \quad (3.14)$$

Результат розрахунків оформляється в табл. 3.2. Діаграма планової собівартості програмного забезпечення представлена на рис. 3.31.

Таблиця 3.2 – Розрахунок планової вартості вебдодатку

Стаття витрат	Витрати, грн
1. Матеріали та комплектуючі (М)	2200
2. Електроенергія (Е)	193
3. Основна заробітна плата виконавців ($З_о$)	19085
4. Додаткова заробітна плата виконавців ($З_д$)	3817
5. Відрахування на соціальні потреби ($З_{сз}$)	8016
6. Амортизація (А)	2977
7. Накладні витрати ($Р_н$)	1909
8. Загальна сума витрат за кошторисом (повна собівартість) (С)	38197

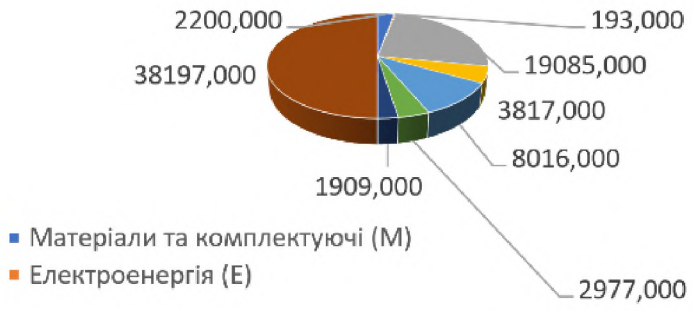

Рисунок 3.31 – Діаграма планової собівартості вебдодатку

Таким чином, було розроблено клієнтську та серверну частину вебдодатку та проведено економічне обґрунтування, в результаті якого було розраховано загальну вартість витрат на розробку.

ВИСНОВКИ

У результаті проведеного дослідження та розробки було успішно створено інтерактивний вебдодаток на базі React, призначений для підготовки користувачів до співбесід в IT-компаніях. Цей додаток є комплексним інструментом, що охоплює різноманітні аспекти підготовки, включаючи технічні знання, навички вирішення проблем та поведінкові аспекти.

У ході роботи було проведено глибокий аналіз сучасних технологій фронтенд-розробки, включаючи React та його екосистему, а також серверну платформу Node.js. Було детально розглянуто ключові особливості React, такі як компонентна архітектура, віртуальний DOM та односпрямований потік даних, які дозволили створити гнучкий, масштабований та високопродуктивний додаток.

Особлива увага була приділена інтеграції штучного інтелекту у вигляді GPT API для генерації питань до користувача. Це дозволило створити інтерактивний та адаптивний процес підготовки, що враховує індивідуальні потреби та рівень знань кожного користувача.

Вебдодаток реалізує широкий спектр функціональних можливостей, включаючи генерацію питань різної складності та тематики, відстеження прогресу користувача, надання зворотного зв'язку та рекомендацій щодо подальшого навчання. Завдяки використанню сучасних технологій та підходів, додаток забезпечує інтуїтивно зрозумілий та зручний інтерфейс, що сприяє ефективному навчанню та підготовці до співбесід.

Проведене тестування підтвердило високу ефективність та зручність використання розробленого вебдодатку. Користувачі відзначили його інтерактивність, адаптивність та корисність у підготовці до співбесід.

Розроблений вебдодаток має значний потенціал для подальшого розвитку та вдосконалення. Можливі напрямки розвитку включають розширення бази питань, додавання нових функцій, таких як симуляція співбесіди, інтеграція з іншими навчальними ресурсами та платформами, а також адаптація під різні IT-спеціалізації.

Успішна реалізація цього проекту підтверджує ефективність використання React та Node.js для створення складних та інтерактивних вебдодатків. Застосування штучного інтелекту у вигляді GPT API відкриває нові можливості для персоналізації та адаптації навчального процесу, що сприяє підвищенню ефективності підготовки до співбесід в IT-компаніях.

Таким чином, результатом роботи є вебсервіс для підготовки до співбесіди в IT-компанію. Він може бути використаний для подальших досліджень за даною тематикою та при проектуванні нейропомічників.