КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від ___ травня 2025 року, протокол № ____.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

ДИПЛОМНА РОБОТА МАГІСТРА
на тему:
«ІНТЕГРАЦІЯ ВЕБ-ВЕРСІЇ ШТУЧНОГО ІНТЕЛЕКТУ З ВИКОРИСТАННЯМ СТЕКУ MERN»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 – Електронні комунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Горський Олег Олексійович _____

Науковий керівник:
канд. фіз.-мат. наук, доцент
Кононов Михайло Володимирович _____

Рецензент:
завідувач кафедри комп'ютерних наук
Державного університету інформаційно-комунікаційних технологій,
доктор технічних наук, професор
Вишнівський Віктор Вікторович_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Олег Горський

Київ – 2025

РЕФЕРАТ

Дипломна робота: 69 ст., 20 рис., 1 табл., 20 джерел, 15 дод.

ШТУЧНИЙ ІНТЕЛЕКТ, ВЕБ-ВЕРСІЯ, MERN, СЕРВЕРНА ЧАСТИНА, ВІЗУАЛЬНА ЧАСТИНА.

Об’єкт дослідження – інтеграція рушія штучного інтелекту у веб-версію, що створена завдяки фреймворку React та інших супутніх елементів.

Мета роботи – інтеграція рушія штучного інтелекту у веб-версію чат-бота з використанням стеку технологій MERN.

У роботі під час створення проекту було використано фреймворки для проектування серверної частини та візуальної частини. Основною мовою програмування у даному проекті є TypeScript, що є супер-сетом мови програмування JavaScript.

В результаті було реалізовано функціонал авторизації, реєстрації, вихід користувача із системи, відправка повідомлень та їх видалення з листування у разі заповнення або втрати контексту листування між користувачем та штучним інтелектом.

Реалізація серверної частини та візуальної частини проекту було розбито на компоненти, що було здійснено вищезгаданому супер-сету TypeScript із додаванням візуальних та серверних плагінів.

Аналіз отриманих результатів свідчить про доцільність і ефективність подальшого використання штучного інтелекту як інтегрованого компонента у фул-стек розробці веб-застосунків, що відкриває нові перспективи для оптимізації розробницьких процесів.

ЗМІСТ

ВСТУП

У XXI столітті попит на пошук та швидку обробку інформації кожного дня зростає в експоненціальній прогресії. З появою штучного інтелекту та його подальшого розвитку – в суспільстві з’явився попит не тільки на пошук та швидку обробку бажаної інформації, але й на те як вона буде якісною у використанні в різних сферах людської діяльності – від допомоги саморозвитку людини до організації та створення надскладних проектів.

Наразі у світі є три «атланта» серед засобів штучного інтелекту – OpenAI, Gemini AI, DeepSeek, які власне створюють між собою конкуренцію та паралельно зумовлюють розвиток сфери нейромереж (поліпшення швидкості обробки інформації, компонування якомога якісної відповіді та тощо).

Метою даного дипломного проекту є побудова складного застосунку, що використовує інтегрований у нього засіб штучного інтелекту з використанням мови програмування TypeScript та фреймворків React.JS, Express.JS, системою керування бази даних MongoDB та серверного середовища Node.JS й рушія OpenAI – ChatGPT.

Задачі даного проекту наступні:

1. Створення фронт-енд частини проекту (візуалізація проекту).

2. Створення серверу, що стабільно функціонує та зберігає у собі інформацію отриману з боку клієнта.

3. Створення працездатної бек-енд частини проекту, яка дозволяє обробляти запити клієнта й виводити на екран бажану клієнтом інформацію.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. **Огляд сучасних веб-рішень на основі штучного інтелекту**

Розвиток штучного інтелекту дозволяє створювати безліч рішень для його використання з урахуванням заданої задачі та сфери, де власне буде застосуватись штучний інтелект – від віртуальних асистентів до генерації контенту. Разом з цим багато компаній включають використання штучного інтелекту у власне виробництво свого продукту, що зумовлює підвищення ефективності та зручність самого користування.

У даному підрозділі буде наведено приклади сучасних веб-рішень на основі штучного інтелекту та де, як вони використовуються.

Чат-боти та асистенти:

Чат-боти та асистенти є найпоширенішими веб-рішеннями штучного інтелекту, вони використовують нейронні мережі для розуміння отриманого від користувача тексту та подальшої генерації відповіді або імітації людського мислення.

1. ChatGPT (OpenAI) – найбільш поширений в світі чат-бот, який дозволяє якісно скомпонувати та надати користувачу якісну інформацію, яку запитував користувач. Підтримує діалог та в останніх версіях розгорнуто з поясненнями пише програмний код будь-якою мовою програмування.

2. Google Bard (Gemini AI) – на рівні з ChatGPT чат-бот, але вже інтегрований у середовище Google та на відміну від вищезгаданого ChatGPT – вміє аналізувати зображення й має доступ до актуальних даних у відкритому доступі.

3. Meta AI – вбудований штучний інтелект у застосунки Messenger та WhatsApp, що генерує відповіді на запити та допомагає у різних задачах.

При цьому усі перелічені приклади дуже сильно відрізняються між собою, наприклад ChatGPT має глибоке розуміння тексту, який отримує від користувача та може чітко й зрозуміло сформувати користувачу відповідь на задане запитання. У той же час Gemini AI є мультимодальним, що означає можливість аналізувати отриманий текст, аудіофайл, відеофайл або зображення одночасно [1].

Meta AI вже відрізняється від ChatGPT та Gemini AI, тим що вона вже запрограмована із великим масивом даних, які мають усі можливі джерела – від книжок до веб-сторінок [2], а також на відміну від вищезазначених аналогів, даний ШІ використовує підхід самонавчання, коли модель навчається передбачати наступне слово в послідовності тексту. Цей процес повторюється мільйони разів, що дозволяє моделі вивчати закономірності, зв'язки та контекст у мові.

Інтеграція ChatGPT у різні веб-додатки дозволяє значно покращити якість обробки запитів користувачів завдяки його здатності до глибокого розуміння природної мови, що підтверджує ефективність таких рішень у сучасних інформаційних системах [3].

Генерація контенту:

Сучасні моделі штучного інтелекту здатні створювати текстові, графічні та відеоматеріали, що у декілька разів спрощує роботу копірайтерів, дизайнерів та інших спеціалістів.

1. Dall-E – генератор зображень від OpenAI, що створює графічні зображення згідно користувацьких запитів.

2. Runaway ML – сервіс для створення відео на основі текстового опису та подальшого редагування кадрів за допомогою штучного інтелекту.

3. DeepL – перекладач, який використовує метод «deep learning» для точного розуміння контексту користувацького запиту.

Автоматизація та обробка даних:

Велика кількість сучасних веб-додатків також використовують штучний інтелект для автоматизації своєї роботи з даними та подальшої їхньої аналітики.

Інструменти штучного інтелекту дозволяють створювати ефективні веб-додатки, що можуть самостійно навчатись та приймати рішення на основі наявних даних.

1. Hugging Face – платформа для роботи з обробкою природньої мови та методикою «deep learning».

2. OpenCV – бібліотека для комп’ютерного зору, що використовується для аналізу зображення в реальному часі.

Дослідження показують, що використання генеративних моделей (DALL-E тощо) дозволяє автоматизувати створення контенту, що суттєво знижує навантаження на традиційних спеціалістів [4].

Пошукові системи та аналітика:

Пошукові системи значно покращились з моменту появи штучного інтелекту, адже це дозволяє надавати користувачеві найбільш релевантну та чітку відповідь. Разом з цим штучний інтелект у пошукових системах дозволяє якомога швидше отримувати відповіді, знаходити у відкритому доступі найбільш релевантний контент з подальшим аналізом великих обсягів даних.

1. Wolfram Alpha – розрахункова система з використанням ШІ (штучного інтелекту) для подальших математичних, аналітичних розрахунків, наукових відповідей тощо.

2. Google Search (Search Generative Experience – SGE) – нова пошукова технологія Google, що узагальнює відповіді та надає готові рішення користувачам у відповідь.

Вищенаведені приклади використання штучного інтелекту дозволяють скласти повну картину того, де та власне яким окремим задачам піддаються веб-рішення.

1.2. **Вибір технологічного стеку MERN для реалізації проекту**

Для подальшої розробки даного проекту необхідно обрати оптимальний стек технологій, який в першу чергу забезпечить продуктивність додатку та зручність під час розробки.

В нашому проекті було обрано стек MERN, що дозволяє створити повноцінний веб-додаток із надійним серверним та клієнтським функціоналом.

Стек MERN розшифровується наступним чином:

- M – MongoDB – нереляційна база даних, що дозволяє зберігати інформацію у вигляді JSON формату.
- E – Express.JS – веб-фреймворк для створення серверного середовища Node.JS.

- R – React.JS – бібліотека мови JavaScript для побудови динамічного інтерфейсу користувача.
- N – Node.JS – серверне середовище для виконання коду, який написаний на мові JavaScript.

Зазначений стек MERN є найбільш поширеним стеком технологій, який використовують для розробки веб-додатків на рівні з MEVN та MEAN, які відрізняються між собою візуальним фреймворком Vue.JS та Angular.

Даний стек також дозволяє проводити розробку набагато швидше та надає просте управління компонентами проекту, як й швидку обробку даних.

Як показують дослідження Yenduri et al. (2023) та Pendela et al. (2024), застосування стеку MERN забезпечує високу продуктивність і масштабованість веб-додатків завдяки повній сумісності JavaScript-компонентів і можливості швидкої інтеграції API [5].

Далі буде зазначено обґрунтування, чому саме цей стек технологій був обраний для даного проекту.

MongoDB є ідеальним вибором для веб-додатків, які працюють із великими обсягами динамічних даних, включаючи текстову інформацію, зображення та метадані.

- **Гнучкість структури даних** – на відміну від реляційних баз даних (SQL), MongoDB не потребує жорстких схем, що дозволяє легко змінювати структуру даних.
- **Масштабованість** – підтримує горизонтальне масштабування, що робить її ідеальним вибором для високонавантажених застосунків.
- **Інтеграція з JavaScript** – використання JSON-подібних документів спрощує взаємодію з іншими компонентами MERN-стеку.
- **Швидкість і продуктивність** – завдяки індексації та кешуванню, MongoDB швидко обробляє запити, що покращує продуктивність додатку.
- Express.js використовується для створення API та серверної логіки, що є важливою частиною веб-додатку.

- **Простота використання** – Express.js має мінімалістичну структуру, що дозволяє швидко налаштовувати бекенд.
- **Гнучкість** – легко інтегрується з різними модулями та бібліотеками (наприклад, для аутентифікації, логування, обробки файлів).
- **Ефективне управління запитами** – підтримує middleware для обробки HTTP-запитів, що полегшує реалізацію бізнес-логіки.
- **Розробка REST API** – дозволяє створювати ефективні API для взаємодії фронтенду з бекендом

Згідно з класичними дослідженнями Pendela та Sai (2024), використання REST-архітектури дозволяє розробити масштабовану і гнучку систему, що є особливо важливим для інтеграції різноманітних сервісів, таких як штучний інтелект, у веб-додатках [6].

React.js обрано як основну технологію для створення інтерфейсу користувача.

- **Компонентний підхід** – дозволяє розбивати додаток на модульні частини, що спрощує розробку та підтримку коду.
- **Віртуальний DOM** – підвищує продуктивність веб-інтерфейсу, мінімізуючи оновлення реального DOM.
- **Односторінкові застосунки (SPA)** – React.js ідеально підходить для створення швидких і динамічних веб-додатків.
- **Екосистема та підтримка** – велика кількість бібліотек (Redux, React Query) дозволяє гнучко налаштовувати управління станом додатку.
- **Node.js** використовується як основа серверної частини, забезпечуючи високу продуктивність і асинхронну обробку запитів.
- **Єдиний стек JavaScript** – дозволяє використовувати JavaScript як для фронтенду, так і для бекенду, що спрощує розробку.
- **Асинхронна архітектура** – використання неблокуючих операцій забезпечує високу швидкість обробки запитів.

- **Швидкодія** – Node.js працює на рушії V8, що робить його одним із найшвидших середовищ виконання JavaScript.
- **Масштабованість** – добре підходить для розподілених систем і мікросервісної архітектури.

Також варто розбрати, які мають переваги компоненти стеку MERN:

- **Повна сумісність компонентів** – MongoDB, Express.js, React.js і Node.js працюють разом без потреби в додаткових адаптаційних інструментах.
- **Єдиний стек мови програмування JavaScript** – дозволяє використовувати одну мову програмування для всіх частин проєкту.
- **Гнучкість та масштабованість** – стек легко адаптується до змін у структурі даних та росту кількості користувачів.
- **Простота інтеграції API** – бекенд легко взаємодіє з фронтендом через REST або GraphQL API.
- **Висока продуктивність** – асинхронна робота серверної частини та ефективне управління даними.
- **Популярність та підтримка спільноти** – велика кількість документації, форумів і бібліотек, що спрощує розробку та вирішення проблем.

Варто зазначити, що обраний стек є оптимальним вибором для реалізації додатка такого плану. Дослідження Bogner та Merkel (2022) свідчать, що використання TypeScript забезпечує кращу структурованість та зменшує ймовірність виникнення помилок у порівнянні з JavaScript, що є однією з ключових причин вибору даної мови для цього проекту [7].

1.3. **Постановка завдання та визначення вимог до веб-версії**

Інтеграція рушія штучного інтелекту у веб-версію вимагає чітко сформульованої та логічно обґрунтованої постановки задачі й визначення вимог до функціональності системи. Такий підхід допоможе створити зручний, продуктивний та ефективний веб-застосунок, що відповідатиме стандартам та потребам користувачів.

Основним завданням є розробка повноцінної платформи, що включатиме у собі:

- Інтерактивний веб-інтерфейс сторінки, який забезпечить швидкий та зручний доступ до можливостей штучного інтелекту.
- Механізм обробки запитів, що дозволятиме аналізувати введені користувачем дані та форматувати відповіді.
- Розгортання моделі штучного інтелекту на серверній частині та її безпосередня інтеграція із бекендом.
- Система збереження історії запитів та результатів взаємодії для подальшого покращення досвіду користувача.
- Оптимізація продуктивності та безпеки для стабільності роботи веб-додатка.

Цей функціонал буде забезпечувати користувачам можливість швидко отримувати відповіді від ШІ та взаємодіяти із системою у зручному веб-інтерфейсі та використовувати можливості штучного інтелекту у своїй діяльності, а саме просте листування із чат-ботом.

До функціональних вимог проекту треба зазначити, що вони будуть ділитись на чотири частини даного проекту – фронтенд, бекенд, база даних та сам рушій штучного інтелекту, що буде інтегрований у сам проект.

Вимоги до фронтенд частини:

- Реалізація зручного та доступного для користувача інтерактивного інтерфейсу в подальшій взаємодії з ШІ.
- Створення динамічних компонентів (поле введення, кнопки управління тощо).
- Адаптивність дизайну під різні пристрої (комп’ютери, планшети, мобільні телефони).
- Відображення історії запитів та відповідей в чаті.

Функціональні вимоги до бекенду:

- Розгортка серверної частини, що буде обробляти користувацькі запити.

- Інтеграція з моделлю штучного інтелекту для генерації відповідей.
- Організація передачі репрезентативного стану для подальшої взаємодії між клієнтом та сервером.
- Реалізація авторизації та реєстрації користувачів.
- Функціонал «виходу» користувача із системи.

Вимоги до функціоналу бази даних:

- Збереження історії користувацьких запитів.
- Збереження інформації при авторизації та реєстрації користувачів.

Вимоги до штучного інтелекту:

- Використання готової моделі (в розрахунку до цього проекту, буде використовуватись рушій ChatGPT).
- Генерація відповідей на основі текстових запитів від користувачів.
- Оптимізація продуктивності для миттєвої обробки запитів користувачів.

Разом з цим до проекту додаються нефункціональні вимоги, які визначатимуть якість роботи веб-застосунку, його безпеку та зручність використання користувачем.

Продуктивність веб-версії полягає в наступному:

- Час відповіді на запит користувача не повинен перевищувати 2-3 секунд.
- Балансування навантаження сервера при великій кількості запитів користувачів.
- Оптимізація роботи з базою даних для швидкого доступу до історії запитів користувача.

Безпека застосунку повинна відповідати таким вимогам:

- Захист API від неавторизованого доступу та можливих атак.
- Шифрування даних, що передаються між клієнтом та сервером (авторизація користувача).

Кросплатформеність має собою на увазі реалізацію питань працездатності застосунку у веб-браузерах для мобільних та десктопних пристроїв (персональні комп’ютери, ноутбуки).

Сам веб-застосунок повинен бути функціональним, безпечним, швидкодійним та продуктивним, при цьому забезпечуючи користувачів зручним та зрозумілим інтерфейсом із швидким доступом до штучного інтелекту. Завдяки використанню стеку MERN можна створити інтерактивний, безпечний застосунок із високим рівнем продуктивності, що буде відповідати сучасним вимогам.

2. ОБҐРУНТУВАННЯ АРХІТЕКТУРИ

2.1. Архітектурні підходи до розробки веб-додатків на основі MERN

Вибір правильної архітектури є головним фактором для створення продуктивної веб-версії штучного інтелекту. Стек MERN якраз дозволяє будувати розробниками повноцінні веб-застосунки використовуючи JavaScript або TypeScript на всіх рівнях та етапах розробки. У цьому підрозділі буде оглянуто архітектурні принципи, підходи до структурування коду та взаємодію між компонентами у проекті.

Варто зазначити, що проект ділиться на дві частини – бекенд (серверна частина) та фронтенд (візуальна частина) в стеку MERN. Але перед цим також треба уточнити певні загальні принципи архітектури проектів, які використовують MERN:

- Розділення проекту на дві критичні частини: сервер та клієнт, де за сервер відповідають Node.JS та Express.JS, а вже React.JS відповідає за клієнтський рівень. Це дозволяє розробнику окремо розвивати обидві частини продукту.
- Клієнт-серверна архітектура являє собою повну взаємодію між фронтендом та бекендом через застосування принципу REST API для отримання та відправки даних.
- MVC-підхід (Model-View-Controller) – цей підхід використовується у бекенд частині проекту, де воно розділяє на три частини:
- Model (база даних) – база даних, що керує структурою та логікою наявних даних.
- View (клієнтська частина) – відображення візуальної частини для користувача.
- Controller (серверна частина) – обробка запитів між фронтендом та базою даних.

Розібравшись із загальними принципами архітектури MERN, ми можемо перейти до огляду архітектурних підходів в розробці проекту. В даному

дипломному проекті застосовуватись один з двох архітектурних підходів: традиційна монолітна архітектура проекту або архітектура клієнт-сервер.

Відмінність між традиційною монолітною архітектурою та клієнт-серверною архітектурою полягає в тому, що в першому варіанті увесь застосунок працюватиме як єдиний механізм – клієнт, сервер та база даних. В той же час клієнт-серверна архітектура працює наступним чином для даного застосунку – фронтенд працює окремо від серверної частини через REST API, коли в цей же час бекенд оброблятиме запити отримуючи їх та записуючи в базу даних.

Також треба додати, що в цьому проекті із обраною архітектурою є повноцінна взаємодія між фронтендом та бекендом, яка виглядає наступним чином:

React.JS подає запит користувача через API та відповідає за відображення даних на екрані, коли в цей час Express.JS та Node.JS приймають запит від користувача й обробляють його з подальшим зверненням на базу даних. В свою чергу база даних зберігатиме дані такі, як дані користувача (логін, пароль), запити до штучного інтелекту, тощо.

REST API, який тут буде задіяний – його задачею буде забезпечення ефективного обміну даних між клієнтом та сервером.

2.2. Архітектурне проектування фронтенд-частини

Фронтенд відповідає у веб-додатку за відображення інформації та інтерфейсу на екрані пристрою, а також взаємодію між користувачем та системою із обміном даних. Використання фреймворку React.JS дозволяє нам створювати динамічні та водночас зручні у експлуатації додатки.

Саме архітектурне проектування фронтенду в цьому проекті діє за наступними принципами, яких потрібно притримуватись під час розробки:

- **SPA (Single Page Application)** – застосунок працює без перезавантаження сторінок, що забезпечує швидкість роботи та зручність для користувача.

Компонентний підхід – весь UI ділиться на незалежні, повторно використовувані компоненти, що полегшує підтримку та розвиток проекту.

- **Розподіл логіки за рівнями** – компоненти розділяються за їхньою роллю в застосунку (**контейнерні компоненти** взаємодіють з API, **презентаційні** відповідають за відображення даних).
- **Керування станом** – використовується React Context API, Redux або Zustand для глобального управління станом.
- **Маршрутизація** – використання React Router для динамічного керування сторінками без повного перезавантаження.
- **Компонентність** – розбиття проекту на декілька елементів для їхнього подальшого компонування в один файл.

На рисунку 2.1. зображено з яких елементів складається візуальна частина веб-версії та їх пов’язаність між собою в подальшому під час розробки.

Як бачимо, візуальна сторона проекту складається з трьох головних елементів та дотичних до них компонентів з функціоналом, який буде реалізований у серверній частині.

Сторінки реєстрації, головна сторінка та сторінка чату – є ключовими візуальними елементами для даного проекту.

На сторінці реєстрації користувач зможе зареєструвати свій обліковий запис та матиме змогу авторизуватись у систему, яка буде виглядати спочатку на головній сторінці, а потім буде переводити користувача на сторінку чату для подальшої взаємодії у вигляді відправки текстових запитів із використанням рушія штучного інтелекту.

Рисунок. 2.1 – Схема елементів клієнтської частини застосунка

На рисунку 2.2. наведено архітектурну складову фронт-енд частини проекту, який є найбільшим за обсягом та реалізацією, та складається з достатньої кількості компонентів – опис, яких буде наведено нижче.

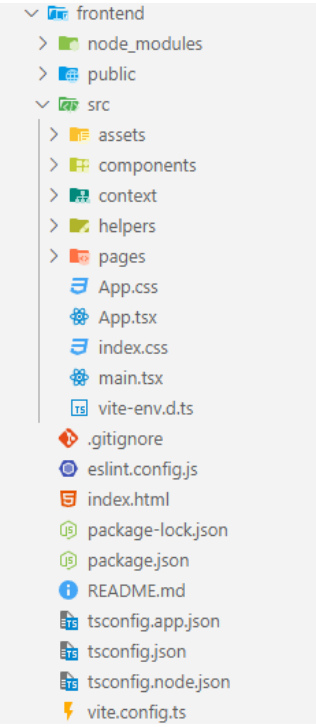

Рисунок. 2.2 – Архітектура клієнтської частини

Дана архітектурна складова фронт-енду складається з декількох ключових деталей, а саме:

- Елементи сторінок веб-сайту;
- Візуальні елементи;
- Хедер елемент сторінки;
- Допоміжні команди;

Кожний з цих перелічений елементів проекту відповідає за візуалізацію та часткову логіку, яка в більшості прописана на боці сервера.

Також варто зазначити, що на рисунку видно певні елементи на кшталт node_modules, вони також є головними частинами фронтенд та бекенд проекту, які у пізніх розділах будуть оглянуті більш детально.

Потрібно також звернути увагу на те, що архітектурне древо візуальної частини тісно пов’язано із серверною частиною, так як будуть прописані певні команди, які будуть відповідати за потрібний функціонал (введення повідомлення, відправка, тощо).

У наступному підрозділі буде оглянуто та обґрунтовано вибір технологій, які будуть використовуватись при розробці бекенду.

2.3. Вибір та обґрунтування складових елементів фронтенд частини

Фронтенд частина як було вище згадано потребує плагіни та фреймворки, які будуть забезпечувати роботу проекту та створить привабливий для користувача інтерфейс. Візуальна частина буде написана із використанням фреймворку React та мовою програмування TypeScript, який підтримує можливість створення додатків на базі фреймворку React.

Фронтенд також потребує для взаємодії із бекендом ряд плагінів, які зможуть через принцип REST API обробляти запити з боку користувача на сервер. Так само й для налаштування переходів між сторінками у веб-версії потребує плагінів до прикладу react-router-dom.

На рисунку 2.3. буде наведено повний перелік плагінів, які використовувались для створення візуальної частини проекту, разом з пов’язанням API фронт-енду та сервером, який створений у бекенд частині.

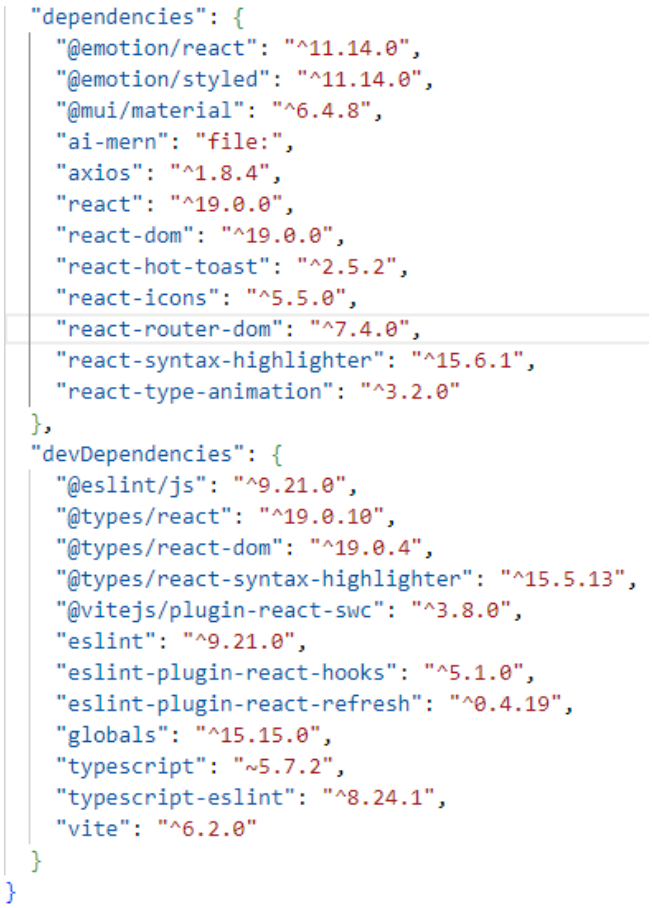

Рисунок. 2.3 – Перелік встановлених плагінів для клієнтської частини

Більшість зображених плагінів на рисунку 2.3., які встановлені у рядок dependencies відповідають за візуалізацію необхідних елементів, коли в той час як axios, react-router-dom, react-dom відповідають за обробку запитів в бік серверу з боку користувача та переходів між сторінками на веб-сайті (сторінка логіну, чату, головна сторінка).

У наступному підрозділі буде розкрито вибір із подальшим обґрунтуванням елементів бекенд частини проекту.

2.4. Вибір та обґрунтування складових елементів бекенд частини

Бекенд частина потребує повноцінної реалізації закладених задач у самому проекті – вхід/вихід користувача, зв’язок між програмою та рушієм штучного інтелекту, збереження даних користувача й даних на сторінці чату.

Для подальшої розробки бекенд частини потрібно використовувати, як й в усьому проекті – мову програмування TypeScript – ця мова програмування є

аналогом JavaScript, який має на меті розширити можливості JS для розробки складних проектів.

До складових елементів варто віднести плагіни, які будуть встановлені у проект та як буде виглядати архітектурне дерево бекенд проекту з поясненням.

Плагіни, які встановлюються у проект відповідатиму за можливості побудови сервера, який буде мати повноцінне навантаження та відповідатимуть також за реалізацію потрібного функціоналу. Нижче на рисунку 2.4. наведено архітектуру серверної частини.

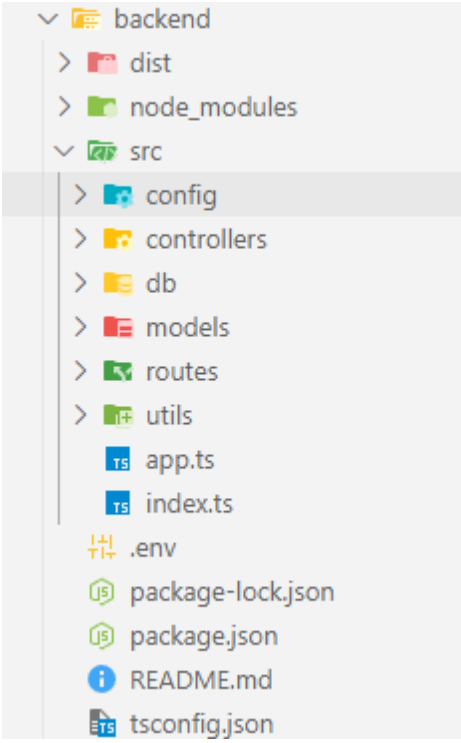

Рисунок. 2.4 – Архітектура серверної частини проекту

Власне бекенд частина, що зображена на рисунку 2.4. по архітектурі не є масштабною, як фронтенд частина, але саме вона має більший пріоритет у створенні повноцінної веб-версії проекту.

У бекенді будуть прописані усі можливі маршрути проекту, моделі (користувач, чат), база даних, контролери тощо.

Для даного проекту буде використано з десяток плагінів, які потрібні для реалізації того чи іншого функціоналу. Далі наведено рисунок 2.5. на якому зображено та приведено перелік встановлених плагінів для бекенду.

Рисунок. 2.5 – Перелік плагінів для серверної частини, що були інстальовані через використання node package manager

Як ми можемо спостерігати на рисунку 2.5., встановлені плагіни та фреймворки йдуть в алфавітному порядку та відповідають за реалізацію того чи іншого функціоналу, до прикладу express, який встановлений для створення та запуску сервера програми.

Разом з express є й інші плагіни, які взаємодіють із сервером до прикладу як cors, cookie-parser, jsonwebtoken – усі вони відповідають за потрібний функціонал, як збереження даних користувача при авторизації, спрощенні роботи сервера тощо.

Згідно з дослідженням Brito et al. (2023), використання статичних аналізаторів коду є ефективним методом виявлення вразливостей у Node.js-додатках. Це підкреслює важливість інтеграції middleware, таких як cors та express-validator, для забезпечення безпеки веб-застосунків. [8].

У той же час плагін dotenv відповідає за створення середовища для необхідних проекту ключів, що дозволяють використовувати певні застосунки – для прикладу – з'єднання нереляційної бази даних MongoDB або створення секретного ключа JWT токену [18], чи взагалі пропису номеру іншого порту сервера, який буде підключений та працюватиме у програмі.

Встановлене у проект розширення cors дозволяє втілити безпеку веб-сторінки, яка розробляється поза політикою одного походження та не відповідає їй.

Інстальований express-validator дозволяє проекту автоматично виявляти помилки перевірки та генерує змістовні повідомлення про помилки, що спрощує виявлення недопустимих вхідних даних у веб-версію.

Також усі встановлені компоненти йдуть в графі dependencies, що свідчить про те, що вони є складовою частиною проекту та використовуються в елементах бекенд частини так й для фронтенд половини проекту. Присутні також в останніх рядках devDependencies вони означають, що плагіни встановлені у проект є допоміжними інструментами, що підвищують ефективність в написанні коду.

У підсумку можна зробити висновок, що навіть за умови використання обмеженої кількості плагінів і фреймворків у серверній частині, їхня роль не втрачає своєї вагомості. Навпаки — це сприяє ускладненню процесу та наближенню архітектури бекенду до ідеального стану. Аналогічна ситуація спостерігається і при розробці клієнтської частини: мінімальна кількість цільових розширень забезпечує ефективне вирішення поставлених завдань веб-версії проєкту.

3. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМИ

3.1. Структура бекенд складової програми та її обґрунтування

Серверна частина або ж бекенд є необхідною складовою проекту та потребує створення зрозумілої структури, яка матиме зручне використання й буде легкою в подальшому рефакторингу проекту іншими розробниками.

Нижче наведено на рисунку 3.1. з чого складається бекенд складова програми.

Структурна складова програми, як й було сказано в минулих розділах складається з бази даних, утиліт, маршрутів, контролерів та конфігурацій, які необхідні для повноцінної працездатності проекту.

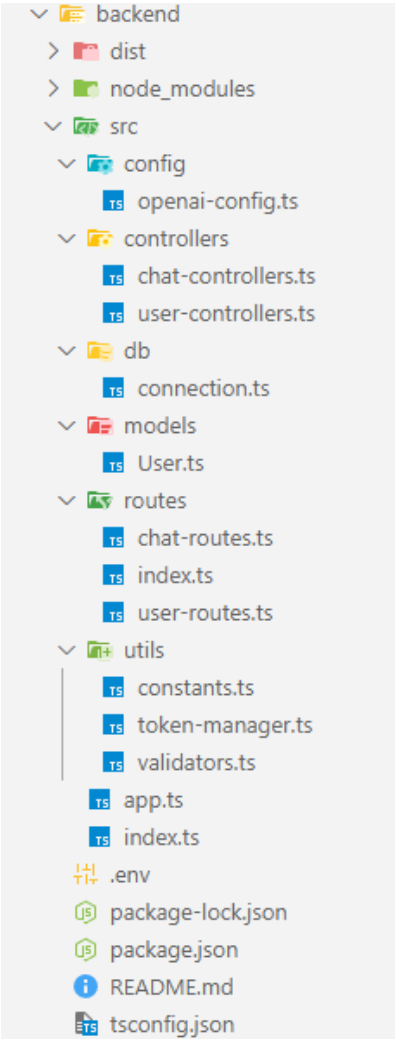

Рисунок. 3.1 – Структура серверної частини та її складові елементи

Як можемо спостерігати на рисунку 3.1. у проекті створено маршрутизації на боці клієнта, чата, які з’єднанні між собою та відповідають за авторизацію/реєстрацію користувача в систему, вихід користувача з системи, на боці чату воно реалізує перелік чатів, створення нового та видалення історії чатів на сторінці [15].

Файл конфігурації потрібен для подальшої інтеграції рушія OpenAI у саму серверну структуру проекту для реалізації функціоналу спілкування користувача із штучним інтелектом.

У цей же момент контролери для чату та користувача дозволяють провести операції, що вивірені за API самого застосунка та показані у відповідному додатку Ж та додатку З.

В каталозі бази даних створено файл, який відповідає за інтеграцію нереляційної бази даних у проект.

Каталог моделей показує створену модель користувача, у якій прописано модель, яка відповідає за створення облікового запису у веб-версії застосунку.

Файл із розширенням env відповідає за встановлення необхідних ключів, які потрібні для працездатності ключових елементів серверної частини до прикладу cookies, токени для захисту від небажаних хакерів тощо.

Головні елементи app та index відповідають за налаштування сервера на потрібному порті та створення маршрутизації [17].

На рисунку 3.2 показано структурну схему, як працює серверна частина застосунку по принципу клієнт-серверної архітектури, де клієнт відправляє на сервер запит, яке воно обробляє та передає далі до бази даних, у якій зберігаються потрібні користувачу дані, формулює відповідь та передає назад клієнту через сервер, який створювався.

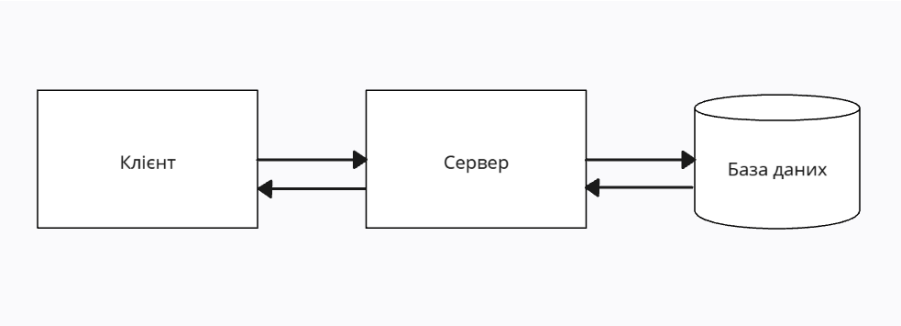

Рисунок. 3.2 – Структурна схема серверної частини, яка працює по принципу клієнт-серверної архітектури

Проміжне ПЗ або ж middleware встановлено у головний файл app та дозволяє провести проміжну обробку запитів на сервер з боку користувача [11].

3.2. Реалізація функціоналу бекенд частини

Функціонал бекенд частини складається з декількох частин: авторизація користувача, реєстрація користувача, вихід з системи, відправка повідомлень, чистка чатів, перевірка користувача.

Почнемо з реалізації функції авторизації. Задача авторизації користувача в системі полягає в тому, щоб вже зареєстрований користувач міг зайти у систему та взаємодіяти із нею, а саме переходити між сторінками та подавати запити чату.

Також було реалізовано систему створення токенів та зберігання cookie для повноцінного збереження інформації про вхід зареєстрованого користувача у систему. Ця система також була реалізована й для реалізації функціоналу реєстрації користувача.

Функція перевірки користувача має на собі мету перевірити автентичність та наявність користувача в базі даних системи, якщо його не існує – то користувач буде вимушений реєструватись в самій системі для подальшої роботи в ній та показано у додатку Ї.

Також у додатку Ї показано, що є функціонал перевірки наявності тексту в повідомленні – якщо користувач спробує відправити порожнє повідомлення – то з’явиться помилка із вимогою прописати у полі текст для повторної відправки повідомлення.

Сама перевірка користувача в системі являє собою перевірку токену, який генерується автоматично в самій веб-версії, якщо не знаходиться токен користувача – система покаже повідомлення про те, що користувача не є зареєстрованим в системі або токен в цьому випадку має збій. Якщо токен є, але він не співпадає з жодним наявним в базі токеном – то буде наступне повідомлення в якому буде сказано, що дозволи не співпадають.

Реалізація реєстрації, виходу користувача майже ідентичні із попередньо вказаними елементами серверної частини, але їх варто згадати в декількох словах.

Реєстрація користувача також має на меті створення облікового запису, який матиме доступ до системи та дозволить працювати із середовищем програми.

Під час створення даного елементу було вказано декілька вимог:

- Пароль повинен складатись з 6 або більше символів;
- Наявність електронної пошти;
- Ім’я повинно складатись з 3 або більше символів;

Якщо одна з цих вимог не виконується, то користувач не матиме змогу потрапити у веб-версію.

Реалізація виходу користувача практично ідентична із функціоналом перевірки автентичності користувача, тільки відрізняється тим, що перед виходом користувача із системи проводиться перевірка дійсності облікового запису, і якщо перевірка пройдена – користувач може полишити систему та знову зайти у неї при бажанні.

Окрім доведення вище згаданих елементів до працездатного стану, головною задачею є інтеграція рушія штучного інтелекту у проект.

У додатку Л зображено код, який відповідає за інтеграцію рушія штучного інтелекту в проект.

OPEN_AI_SECRET є ключем API для проекту з боку рушія ШІ та дозволяє запустити сам штучний інтелект.

В той же час OPEN_AI_ORGANIZATION є «ключем», який відноситься до профілю організації у середовищі OpenAI. [14]

Разом з цим прописано в окремому контролері поведінку цього елемента (додавання, видалення чатів), а також пов’язано із фронтендом, про що буде згодом описано у наступному розділі.

Також було згадано про прописані маршрутизації сервера в бік чату та у бік користувача, до прикладу нижче буде наведено маршрутизацію для складової чату, що показано у додатках И та І.

4. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ПРОГРАМИ

4.1. Обґрунтування структури фронтенд складової та її реалізація

Однією з ключових частин дипломного проекту, а також головною складовою застосунку є фронтенд – він відповідає за візуалізацію та частковий бекенд (з'єднання API із бекендом).

У даному проекті на боці фронтенду є декілька задач та складових, які потребують реалізації та з'єднання із бекендом. Нижче також буде наведено структуру із поясненням до кожного елементу.

На рисунку 4.1. ми бачимо розбиті на декілька елементів необхідні для проекту складові програми.

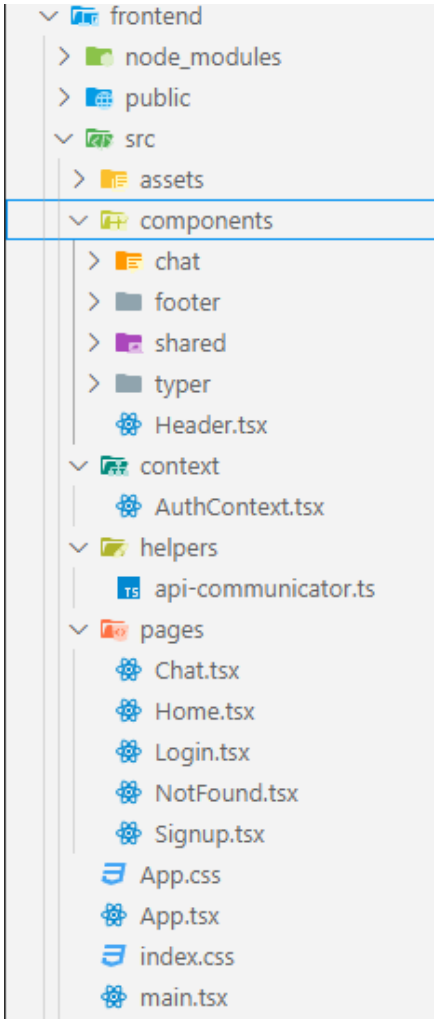

Рисунок. 4.1 – Складові елементи клієнтської частини

Як було раніше згадано в попередніх розділах кожен плагін, пакет можливостей встановлюється в систему через використання команди npm install

[9]. Це дозволяє встановити у проект потрібне розширення для реалізації потрібного функціоналу, разом з цим при встановленні з’являється каталог node_modules із великою кількістю файлів у яких прописані певні можливості та правила використання на велику кількість рядків в коді.

Усі зображені елементи візуальної частини з’єднані між собою під окремі файли та інтегровані у основний файл App.tsx та API реалізація фронтенду зроблена в окремому файлі index.tsx [9].

Важливі елементи візуальної частини занесені у каталог components. В даному каталозі йде розподілення на додаткові елементи: сторінка чату, елементи чату, елементи навігації, що з’єднанні із основним файлом.

Також варто зазначити, що стилізація елементів виконувалась в CSS, а установка самого фреймворку здійснювалась через локальний сервер розробки Vite, який дозволяє встановлювати бажані фреймворки від Angular до Svelte у проекти із вибором мови програмування між JavaScript та TypeScript.

Використання Vite значно спрощує ініціалізацію проектів та дозволяє не використовуючи застарілий метод інсталяції NPX, який використовувався раніше до 2020-го року включно.

Сама стилізація деяких компонентів робилась виключно в самих елементах, для спрощення та наглядного показу, що стилізація може бути зроблена не створюючи окремий CSS-документ для окремого компонента.

У цього метода є перевага та водночас недолік, так як по правилах розробки треба прописувати безпосередньо в окремі CSS-документи кожного компонента.

Перевага внесення налаштувань візуальної частини безпосередньо у сам компонент спрощує процес стилізації та дозволяє в будь-який момент змінити параметри у самому файлі безпосередньо.

Головним недоліком є те, що воно робиться безпосередньо у самому файлі, а не в окремому CSS-документі, так як в більшості проектів робиться акцент на використання CSS-документів або SCSS/SASS розширень, які ще більше дозволяють розширити можливості таблиць каскадного стилю.

Розширення .tsx є варіацією документів React, що використовує мову програмування TypeScript [12], без використання кожний файл мав би стандартне розширення .jsx, що свідчить про використання мови JavaScript.

Враховуючи використання TypeScript разом з React, можна сказати, що використання просунутого аналогу JavaScript значно більше розширяє можливості в розробці проектів – починаючи від строгої типізації даних та закінчуючи поліпшеним ООП (об'єктно-орієнтовним програмуванням), а також ще воно використовує синтаксис JSX, але із системою типів мови Typescript.

Але у загальному при використанні будь-якого фреймворку починаючи від React та закінчуючи Vue.JS буде зберігатись саме синтаксис того чи іншого фреймворка, але із орієнтацією на використання системи типів мови TypeScript [19].

В даному проекті для реалізації візуальної частини було розроблено сім каталогів з кожним елементом, який має відповідний функціонал та безпосереднє використання API. Самі каталоги взаємопов'язані між собою та виконують певну функцію – каталог pages відповідає за візуалізацію окремих сторінок та переходу між ними, footer відповідає за нижню частину веб-сайту (у випадку нашої веб-версії відповідає за нижню частину головної сторінки), каталог shared за візуалізацію елементів, які інтегровані у різні сторінки. [20]

І всі ці файли як було сказано вище, з'єднані з основним файлом app.tsx та має наступний вигляд.

У додатку А показано, що усі компоненти інтегровані в один файл app.tsx та підв'язані до маршрутизації згідно встановленого пакету react-router-dom, який дозволяє користувачу переміщуватись між сторінками у веб-версії.

У додатку Б показано код, який виводить на веб-браузер головну сторінку веб-чату, який має у собі переходи на сторінку реєстрації та показаний у додатку або В у випадку наявної у користувача сторінки – у розділ авторизації, який показаний у додатку Г.

При успішній авторизації користувач буде бачити сторінку чату, який виписаний та показаний у додатку Ґ.

Із виконанням реалізації можемо перейти до опису вимог та засобів тестування програми.

4.2. Вимоги та засоби до тестування програми

В програмному забезпеченні повинні бути реалізовані наступні функції:

- Запуск клієнта та сервера;
- Стабільне з’єднання між клієнтом та сервером;
- Базовий функціонал спілкування між клієнтом та сервером;
- Реалізований графічний інтерфейс клієнта на різних пристроях;

Програмне забезпечення «AI Chat MERN» було протестовано на тестовій платформі ноутбука ASUS TUF Gaming A15 із наступними характерстиками:

- Відеокарта: Nvidia GeForce RTX 4050 (6 ГБ відеопам’яті);
- ОЗУ: 16 Гб;
- Процесор: AMD Ryzen 5 7535HS with Radeon Graphics;
- Накопичувач даних: Western Digital WD PC SN560 SDDPNQE-1T00-1002 (1 ТБ);

Для тестування складових сервера використовувалось програмне забезпечення Postman API, яке перевіряє доступність та виконання усіх поставлених задач, а також із використанням терміналу у середовищі Visual Studio Code.

4.3. Тестування програми

Тестування програми починається із запуску сервера з терміналу Visual Studio Code. У терміналі запускаємо програму через команду npm run dev [10] та очікуємо відповідного сповіщення про коректний запуск сервера.

На рисунку 4.2. бачимо успішний запуск сервера, разом з цим наступним кроком в тестуванні буде запуск клієнта так само із командою npm run dev.

```
21:32:49 - Starting compilation in watch mode...
[0]
[1] Server Open & Connected To Database 🤟
[0]
[0] 21:32:52 - Found 0 errors. Watching for file changes.
[1] Server Open & Connected To Database 🤟
```

Рисунок. 4.2 – Успішний запуск сервера Node.JS, який відображається в терміналі

На рисунку 4.3. можемо побачити, що програма запустилась успішно без видачі будь-якої помилки в терміналі редактора коду.

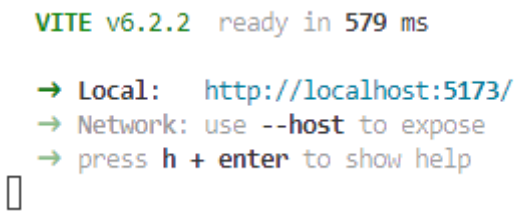

Рисунок. 4.3 – Успішний запуск клієнта у терміналі редактора коду

Клієнт працює на порті 5173, так як це порт, який є за замовченням для запуску цього проекту. На боці сервера, який запускається та працює на порту 5000, ми прописали через плагін cors з'єднання із портом 5173 на боці клієнта для подальшої успішної взаємодії між самим клієнтом та сервером, що показано в додатку Є.

Із перевіркою функціоналу реєстрації, авторизації та виходу користувача з системи у консолі відображається відповідні сповіщення про успішне виконання певного функціоналу, або помилки при виконанні.

Усі результати, що зображені на рисунку 4.4., що здійснені на боці клієнта та з'єднанні із сервером відображають відповідні сповіщення з відміткою 200 (успішне виконання), 304 є еквівалентом до сповіщення 200, 401 є сповіщенням про те, що користувач не є авторизованим у системі та потребує або реєстрації або авторизації в саму систему.

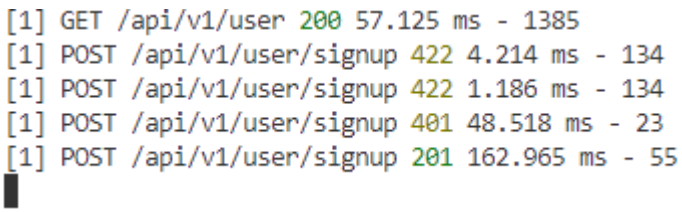

Рисунок. 4.4 – Результати дій з боку клієнта реєструються у терміналі середовища коду

Для перевірки коректності виконання відповідних дій, нам потрібно запустити Postman API та перевірити коректну працездатність авторизації, реєстрації, збору інформації про користувачів.

У першу чергу потрібно перевірити працездатність реєстрації через вищевказану програму.

Нижче на рисунку 4.5. у форматі JSON після відправки запиту видно, що користувач з іменем Oleh та поштою Oleh@test1.com й паролем «1234556» створився із відповідною позначкою 201.

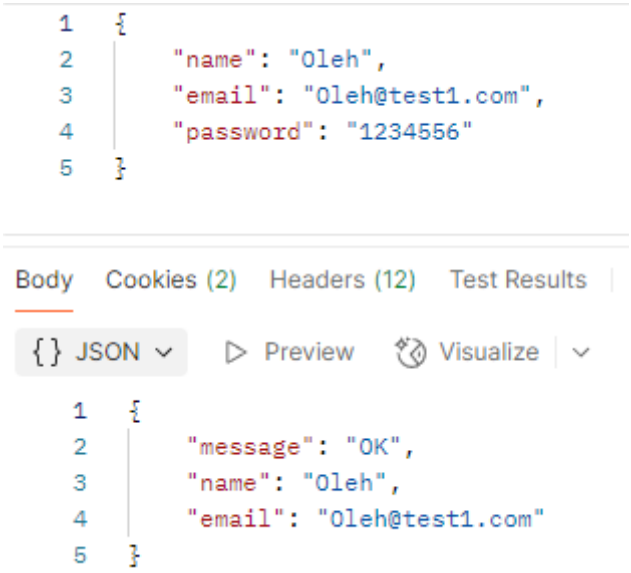

Рисунок. 4.5 – Успішна реєстрація користувача у застосунку Postman API

Функціонал реєстрації працює через систему POST-запитів [16] через чистий код та формуванням у кінцевому висновку даних у JSON форматі з трьох елементів: ім’я, електрона пошта та пароль.

На рисунку 4.6. зображено, що реєстрація користувача в системі не пройдена із наступним сповіщенням. Користувач пробував зареєструватись у системі використовуючи п’ятизначний пароль та ім’я, яке складається з двох літер – коли для повноцінної реєстрації треба мати пароль від 6 та більше символів, й ім’я від 3 та більше символів та у показано у додатку Ї.

Рисунок. 4.6 – Помилка при реєстрації користувача

Разом з успішними перевірками реєстрації, ми можемо перевірити, скільки користувачів зареєстровано у нашому застосунку.

Він поставляється окремим списком та не є видним в самій програмі, окрім як вибору логіна та паролю на сторінці авторизації у веб-версії.

Нижче буде зображення такого списку із подальшим поясненням усіх елементів списку.

На рисунку 4.7. зображено перелік наступних елементів: ідентифікаційний номер користувача, ім’я користувача, електронна пошта, пароль, який переведено у багатозначний формат шифрування, масив чатів, які почали користувачі.

```
{
    "message": "OK",
    "users": [
        {
            "_id": "67de6983d94f3561a558e7d2",
            "name": "James",
            "email": "james@test.com",
            "password": "$2b$10$VTMffDby5zsekgsq6kXUKeauE3xQSAZQhHGYadHg16oSD5vtTuZz2",
            "chats": [],
            "__v": 0
        },
        {
            "_id": "67de69b483d1a6e8a59b0164",
            "name": "John",
            "email": "john@test.com",
            "password": "$2b$10$GLwodcbDPHumF211uSHlUetBUFgOtvbecBAUI1yoyldZWpZ69VobO",
            "chats": [],
            "__v": 0
        },
        {
            "_id": "67de750b1c2149f8e9a2b166",
            "name": "Hlib",
            "email": "Hlib@test.com",
            "password": "$2b$10$..LCz.oYaci6KRdeewWb8uH1YUXqypi217ARfLSnoT3uKp7qJ0B3u",
            "chats": [],
            "__v": 0
        },
        {
            "_id": "67de79b093e2ac00965cb309",
            "name": "Oleh",
            "email": "Oleh@test.com",
            "password": "$2b$10$E3.MMQwLDXLRJOLUP5B1Ce7/iaFY8cvEaEN4Ww.fEZLzCdrHymMbW",
            "chats": [],
            "__v": 0
        },
```


Рисунок. 4.7 – Перелік зареєстрованих користувачів

Після перевірки кількості користувачів, ми можемо перевірити систему авторизації, щоб вона відпрацьовувала вірно та допустила користувача, який виконав відповідні вимоги для доступу.

У додатку Й показано модель користувача, яка показує, що сама модель складається з трьох складових: електронна пошта, ім’я користувача та пароль. Ця модель є складовою частиною бекенду та дозволяє в подальшому створити профіль користувача або зайти у сам профіль.

У додатку Е показано функціонал аутентифікації користувача, де проходить перевірка користувача на відповідність наявних у базі всіх даних (пароль, ел.пошта та ім’я користувача).

Далі, як можемо бачити на рисунку 4.8, користувач з ел.поштою Oleh@test.com та паролем 123456789 пройшов авторизацію та має доступ до системи й має змогу виконувати дії на сайті.

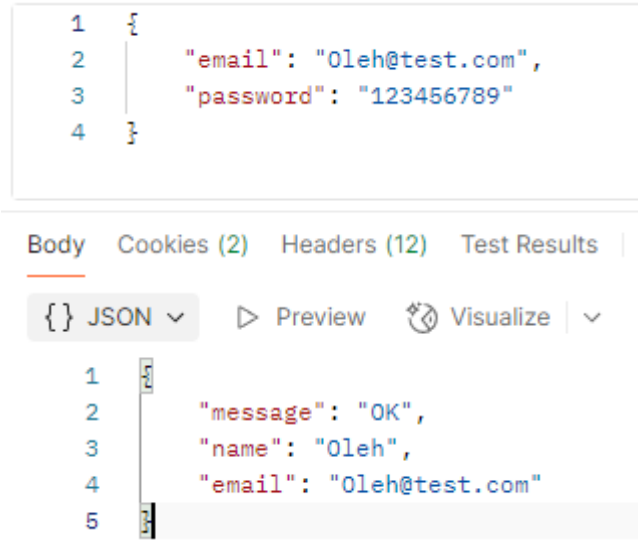

Рисунок. 4.8 – Успішна авторизація користувача

На рисунку 4.9. можемо бачити відповідну спробу авторизації, яка не відповідає вимогам із відповідним результатом.

Рисунок. 4.9. – Неуспішна спроба авторизації із введенням невідповідних даних

Якщо користувач правильно увів свою електронну адресу, але увів пароль, що не відповідає вимогам, то йому впливе повідомлення, що потрібно мати пароль від 6 символів та більше.

Також у систему було встановлено систему JWT, що спрацьовує наступним чином – якщо через відповідний час спливає токен, то користувач буде вимушений вийти із системи та повторно запустити сервер, це зроблено для того, щоб сервер мав якомога більшу безпеку в разі можливих хакерських атак в свій бік.

Також ми маємо разом з цим змогу перевірити, чи зможе користувач відправляти повідомлення в бік сервера.

У додатку Д показано, що прописано API для клієнтської частини серверу, що дозволяє приводити в дію наявний функціонал від реєстрації користувача та відправки повідомлення в чат.

Повідомлення на рисунку 4.10 відправлено, але на сервері воно зберігається та власне проходить опрацювання повідомлення та підготовку відповіді.

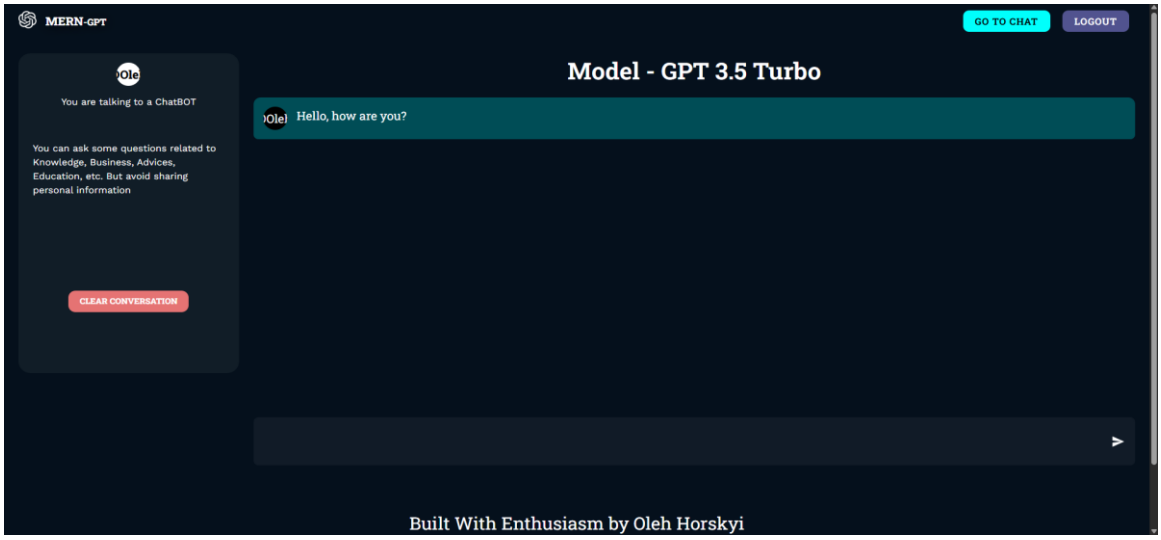

Рисунок. 4.10 – Відправка повідомлення

Але так як ми не отримали зворотного повідомлення з боку сервера, через відповідну помилку 422, ми можемо протестувати наступну функцію.

Увесь функціонал сторінки чату складається з двох складових – можливість відправки повідомлення та чистка чату – і цей функціонал зображений у додатку Д.

На рисунку 4.11 зображено, що якщо користувач відправив повідомлення та не бажає тримати його довго – він може натиснути на відповідну кнопку «Clear Conversation» видаливши разом з цим відправлене повідомлення.

Рисунок. 4.11 – Реалізація функціоналу видалення повідомлення

Також було реалізовано в даному проекті можливість працездатності веб-версії на різних пристроях – від персональних комп’ютерів до мобільних телефонів.

На рисунку 4.12. зображено, що застосунок може працювати на планшеті та візуально буде зручним для використання користувачем, як бачимо є тільки можливість відправки повідомлення та переходу на головну сторінку й відсутність можливості видалення відправлених повідомлень.

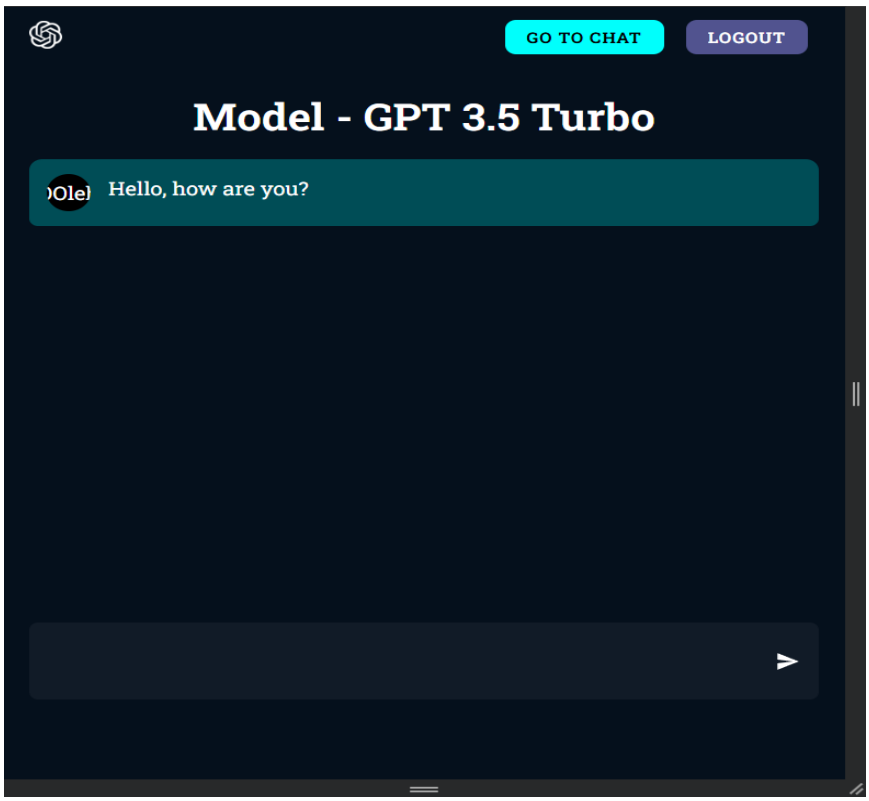

Рисунок. 4.12 – Інтерфейс чату на умовному планшеті

На рисунку 4.13. зображено як виведено інтерфейс чату на телефоні, так само як й на версії для планшету помітна відсутність функціоналу чистки повідомлень.

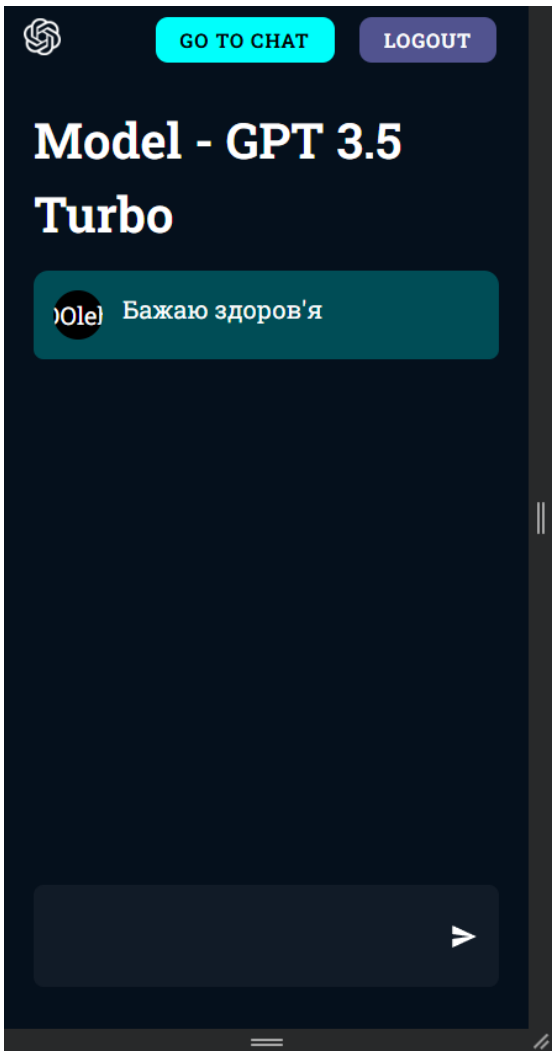

Рисунок. 4.13 – Інтерфейс чату на умовному телефоні

В таблиці 4.1. наведено результати тестування, що свідчить про відносну працездатність програми та реалізована згідно завдання, але функціонал не є стабільним та досконалим.

Таблиця 4.1 – Результати тестування веб-застосунку

№	Функція, зауваження, помилка	Відповідно технічному завданню (ТАК/НІ)
1	Запуск сервера та клієнта	ТАК
2	Стабільне з’єднання між сервером та клієнтом	ТАК
3	Реалізований графічний інтерфейс клієнта	ТАК
4	Реалізація системи авторизації	ТАК
5	Реалізація реєстрації користувача	ТАК
6	Реалізація виходу користувача з системи	ТАК
7	Видалення історії чату	ТАК
8	Зауваження – повідомлення не відправляються при натисканні Enter	-
9	Помилка – повідомлення з боку сервера не відправляється у відповідь на повідомлення користувача	-

ВИСНОВКИ

В результаті виконання даної роботи, було успішно здійснено інтеграцію рушія штучного інтелекту у веб-версію на базі стеку MERN з використанням мови програмування TypeScript. Дана веб-версія дозволяє користувачам відправляти в бік сервера повідомлення у режимі реального часу.

Під час виконання роботи було опрацьовано можливість інтеграції рушія штучного інтелекту в веб-застосунок, роботу API та принципу REST API, взаємодію між клієнтом та сервером. Для виконання роботи було використано фреймворк React, серверне середовище Node.JS з мінімалістичним фреймворком Express.js із різноманітними бібліотеками на стороні клієнта та сервера, нереляційну базу даних MongoDB.

У процесі розробки даної програми було враховано та реалізовано усі технічні вимоги, які були присутні, а також було враховано забезпечення надійності відправки повідомлень на сервер, підтримки безпеки веб-версії та авторизації користувачів у саму веб-сторінку.

Отримані результати під час тестування застосунку підтвердили працездатність програми. Програма працює стабільно та буде постійно оновлюватись з часом для поліпшення наявного функціоналу й додавання нового функціоналу в програму. Дана веб-версія може бути використана як основа для подальших розширень, удосконалень з метою поліпшення працездатності програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Порівняння ChatGPT з іншими аналогами ШІ від сайту gaper.io - https://gaper.io/chatgpt-vs-gemini-vs-llama-vs-meta-ai-vs-claude/

2. Порівняння чат-ботів від сайту citynews.com.ua - https://citynews.com.ua/porivniannia-suchasnykh-shtuchnykh-intelektiv-chatgpt-gemini-claude-ta-llama/

3. Naznin, K., Al Mahmud, A., Nguyen, M. T., & Chua, C. (2025). “ChatGPT Integration in Higher Education for Personalized Learning, Academic Writing, and Coding Tasks: A Systematic Review.” Computers, 14(2), 53 DOI:/10.3390/computers14020053

4. Ho, J. E., Ooi, B. Y., & Westner, M. (2024). “Application Integration Framework for Large Language Models.” In Proceedings of the 5th Int. Conference on Artificial Intelligence and Data Sciences (AiDAS 2024). DOI:10.1109/AiDAS63860.2024.10730541

5. Yenduri, H. L., Sangeetha, Y., Vyshnavi, K. P. S., & Fyzulla, S. (2023). “A Website for a Consultancy using MERN Stack.” In Proceedings of the 3rd Int. Conf. on Smart Data Intelligence (ICSMDI 2023), pp. 195–200. DOI:10.1109/ICSMDI57622.2023.00044

6. K., Hema Sai, B., Balaji, B., & Vishnu Vardhan, A. (2024). “Constructing a Study Buddy Using MERN (MongoDB, Express.js, React, Node.js) Stack Technologies.” Engineering Proceedings, 66(1), 27. DOI:10.3390/engproc2024066027

7. Bogner, J., & Merkel, M. (2022). “To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub.” In Proceedings of the 19th IEEE/ACM Int. Conference on Mining Software Repositories (MSR 2022), pp. 658–669. DOI: /10.1145/3524842.3528454

8. Brito, T., Ferreira, M., Monteiro, M., & Lopes, P. (2023). "Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages." DOI: 10.1109/TR.2023.3286301

9. Офіційна документація фреймворку React. – Режим доступу до ресурсу: https://react.dev/

10. Офіційна документація Node.JS – Режим доступу до ресурсу: https://nodejs.org/uk

11. Офіційна документація фреймворку Express.JS – Режим доступу до ресурсу: https://expressjs.com/uk/

12. Офіційна документація мови програмування TypeScript – Режим доступу до ресурсу: https://www.typescriptlang.org/

13. Офіційна документація нереляційної бази даних MongoDB – Режим доступу до ресурсу: https://www.mongodb.com/docs/

14. Офіційна документація рушія штучного інтелекту OpenAI – Режим доступу до ресурсу: https://platform.openai.com/docs/overview

15. Adam Freeman - Mastering Node.js Web Development / Adam Freeman., 2024. P. 779

16. Dave Westerveld - API Testing and Development with Postman, 2nd Edition / Dave Westerveld,. 2024., P. 359

17. Amit K - Node Js For Beginner: The Ultimade Guide for Intermediate Developer to Unleash the Power of Server Side Javascript / Amit K., 2024., P. 123

18. Paul Done - Practical MongoDB Aggregations / Paul Done., 2023., P. 313

19. Adam Boduch - React and React Native, 5th Edition: Build cross-platform JavaScript and TypeScript apps for the web, desktop, and mobile / Adam Boduch., 2024., P. 509

20. Petri Silén - Clean Code Principles and Patterns, 2nd Edition: A Software Practitioner’s Handbook / Petri Silén., 2023., P. 489

ДОДАТОК А. КЛАС ОСНОВНОГО КОМПОНЕНТУ ФРОНТ-ЕНДУ

```
import Header from "./components/Header";
import { Routes, Route } from "react-router-dom";  239.8k (gzipped: 75.5k)
import Home from "./pages/Home";
import Login from "./pages/Login";
import Signup from "./pages/Signup";
import Chat from "./pages/Chat";
import NotFound from "./pages/NotFound";
import { useAuth } from "./context/AuthContext";
import Footer from "./components/footer/Footer";
function App() {
  const auth = useAuth();

  return (
    <main>
      <Header />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
        {auth?.isLoggedIn && auth.user && (
          <Route path="/chat" element={<Chat />} />
        )}
        <Route path="*" element={<NotFound />} />
      </Routes>
      <Footer/>
    </main>
  );
}

export default App;
```

ДОДАТОК Б. КЛАС ДОМАШНЬОЇ СТОРІНКИ ЧАТУ

```jsx
import { Box, useMediaQuery, useTheme } from "@mui/material";  45.9k (gzipped: 15.1k)
import TypingAnim from "../components/typer/TypingAnim";

const Home = () => {
  const theme = useTheme();
  const isBelowMd = useMediaQuery(theme.breakpoints.down("md"));
  return (
    <Box width={"100%"} height={"100%"}>
      <Box
        sx={{
          display: "flex",
          width: "100%",
          flexDirection: "column",
          alignItems: "center",
          mx: "auto",
          mt: 3,
        }}
      >
        <Box>
          <TypingAnim />
        </Box>
        <Box
          sx={{
            width: "100%",
            display: "flex",
            flexDirection: { md: "row", xs: "column", sm: "column" },
            gap: 5,
            my: 10,
          }}
        >
          <img
            src="robot.png"
            alt="robot"
            style={{ width: "200px", margin: "auto" }}
          />
          <img
            className="image-inverted rotate"
            src="openai.png"
            alt="openai"
            style={{ width: "200px", margin: "auto" }}
          />
        </Box>
        <Box sx={{ display: "flex", mx: "auto" }}>
          <img
            src="chat.png"
            alt="chatbot"
            style={{
              display: "flex",
              margin: "auto",
              width: isBelowMd ? "80%" : "60%",
              borderRadius: 20,
              boxShadow: "-5px -5px 105px #64f3d5",
              marginTop: 20,
              marginBottom: 20,
              padding: 10,
            }}
          />
        </Box>
      </Box>
    </Box>
  );
};

export default Home;
```

ДОДАТОК В. КЛАС СТОРІНКИ АВТОРИЗАЦІЇ

```tsx
import React, { useEffect } from "react";  7.8k (gzipped: 3k)
import { IoIosLogIn } from "react-icons/io";  3.4k (gzipped: 1.5k)
import { Box, Typography, Button } from "@mui/material";  79k (gzipped: 25k)
import CustomizedInput from "../components/shared/CustomizedInput";
import { toast } from "react-hot-toast";  8.6k (gzipped: 3.4k)
import { useAuth } from "../context/AuthContext";
import { useNavigate } from "react-router-dom";  239.8k (gzipped: 75.5k)
const Login = () => {
  const navigate = useNavigate();
  const auth = useAuth();
  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    const formData = new FormData(e.currentTarget);
    const email = formData.get("email") as string;
    const password = formData.get("password") as string;
    try {
      toast.loading("Signing In", { id: "login" });
      await auth?.login(email, password);
      toast.success("Signed In Successfully", { id: "login" });
    } catch (error) {
      console.log(error);
      toast.error("Signing In Failed", { id: "login" });
    }
  };
  useEffect(() => {
    if (auth?.user) {
      navigate("/chat");
    }
  }, [auth]);
  return (
    <Box width={"100%"} height={"100%"} display="flex" flex={1}>
      <Box padding={8} mt={8} display={{ md: "flex", sm: "none", xs: "none" }}>
        <img src="airobot.png" alt="Robot" style={{ width: "400px" }} />
      </Box>
      <Box
        display={"flex"}
        flex={{ xs: 1, md: 0.5 }}
        justifyContent={"center"}
        alignItems={"center"}
        padding={2}
        ml={"auto"}
        mt={16}
      >
        <form
          onSubmit={handleSubmit}
          style={{
            margin: "auto",
            padding: "30px",
            boxShadow: "10px 10px 20px #000",
            borderRadius: "10px",
            border: "none",
          }}
        >
          <Box
            sx={{
              display: "flex",
              flexDirection: "column",
              justifyContent: "center",
            }}
          >
            <Typography
              variant="h4"
              textAlign="center"
              padding={2}
              fontWeight={600}
            >
              Login
            </Typography>
            <CustomizedInput type="email" name="email" label="Email" />
            <CustomizedInput type="password" name="password" label="Password" />
```

```
                <Button
                  type="submit"
                  sx={{
                    px: 2,
                    py: 1,
                    mt: 2,
                    width: "400px",
                    borderRadius: 2,
                    bgcolor: "#00fffc",
                    ":hover": {
                      bgcolor: "white",
                      color: "black",
                    },
                  }}
                  endIcon={<IoIosLogIn />}
                >
                  Login
                </Button>
              </Box>
            </form>
          </Box>
        </Box>
    );
};

export default Login;
```

ДОДАТОК Г. КЛАС СТОРІНКИ РЕЄСТРАЦІЇ

```
import React, { useEffect } from "react";  7.8k (gzipped: 3k)
import { IoIosLogIn } from "react-icons/io";  3.4k (gzipped: 1.5k)
import { Box, Typography, Button } from "@mui/material";  79k (gzipped: 25k)
import CustomizedInput from "../components/shared/CustomizedInput";
import { toast } from "react-hot-toast";  8.6k (gzipped: 3.4k)
import { useAuth } from "../context/AuthContext";
import { useNavigate } from "react-router-dom";  239.8k (gzipped: 75.5k)
const Signup = () => {
  const navigate = useNavigate();
  const auth = useAuth();
  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    const formData = new FormData(e.currentTarget);
    const name = formData.get("name") as string;
    const email = formData.get("email") as string;
    const password = formData.get("password") as string;
    try {
      toast.loading("Signing Up", { id: "signup" });
      await auth?.signup(name, email, password);
      toast.success("Signed Up Successfully", { id: "signup" });
    } catch (error) {
      console.log(error);
      toast.error("Signing Up Failed", { id: "signup" });
    }
  };
  useEffect(() => {
    if (auth?.user) {
      navigate("/chat");
    }
  }, [auth]);
  return (
    <Box width={"100%"} height={"100%"} display="flex" flex={1}>
      <Box padding={8} mt={8} display={{ md: "flex", sm: "none", xs: "none" }}>
        <img src="airobot.png" alt="Robot" style={{ width: "400px" }} />
      </Box>
      <Box
        display={"flex"}
        flex={{ xs: 1, md: 0.5 }}
        justifyContent={"center"}
        alignItems={"center"}
        padding={2}
        ml={"auto"}
        mt={16}
      >
        <form
          onSubmit={handleSubmit}
          style={{
            margin: "auto",
            padding: "30px",
            boxShadow: "10px 10px 20px #000",
            borderRadius: "10px",
            border: "none",
          }}
        >
```

```
            <Box
              sx={{
                display: "flex",
                flexDirection: "column",
                justifyContent: "center",
              }}
            >
              <Typography
                variant="h4"
                textAlign="center"
                padding={2}
                fontWeight={600}
              >
                Signup
              </Typography>
              <CustomizedInput type="text" name="name" label="Name" />
              <CustomizedInput type="email" name="email" label="Email" />
              <CustomizedInput type="password" name="password" label="Password" />
              <Button
                type="submit"
                sx={{
                  px: 2,
                  py: 1,
                  mt: 2,
                  width: "400px",
                  borderRadius: 2,
                  bgcolor: "#00fffc",
                  ":hover": {
                    bgcolor: "white",
                    color: "black",
                  },
                }}
                endIcon={<IoIosLogIn />}
              >
                Signup
              </Button>
            </Box>
          </form>
        </Box>
      </Box>
    );
};

export default Signup;
```

ДОДАТОК Ґ. КЛАС СТОРІНКИ ЧАТУ

```tsx
import { useEffect, useLayoutEffect, useRef, useState } from "react";  4.9k (gzipped: 2k)
import { Box, Avatar, Typography, Button, IconButton } from "@mui/material";  89.1k (gzipped: 27.5k)
import red from "@mui/material/colors/red";  449 (gzipped: 296)
import { useAuth } from "../context/AuthContext";
import ChatItem from "../components/chat/ChatItem";
import { IoMdSend } from "react-icons/io";  2.9k (gzipped: 1.3k)
import { useNavigate } from "react-router-dom";  239.8k (gzipped: 75.5k)
import {
  deleteUserChats,
  getUserChats,
  sendChatRequest,
} from "../helpers/api-communicator";
import toast from "react-hot-toast";  8.6k (gzipped: 3.4k)
type Message = {
  role: "user" | "assistant";
  content: string;
};
const Chat = () => {
  const navigate = useNavigate();
  const inputRef = useRef<HTMLInputElement | null>(null);
  const auth = useAuth();
  const [chatMessages, setChatMessages] = useState<Message[]>([]);
  const handleSubmit = async () => {
    const content = inputRef.current?.value as string;
    if (inputRef && inputRef.current) {
      inputRef.current.value = "";
    }
    const newMessage: Message = { role: "user", content };
    setChatMessages((prev) => [...prev, newMessage]);
    const chatData = await sendChatRequest(content);
    setChatMessages([...chatData.chats]);
    //
  };
  const handleDeleteChats = async () => {
    try {
      toast.loading("Deleting Chats", { id: "deletechats" });
      await deleteUserChats();
      setChatMessages([]);
      toast.success("Deleted Chats Successfully", { id: "deletechats" });
    } catch (error) {
      console.log(error);
      toast.error("Deleting chats failed", { id: "deletechats" });
    }
  };
  useLayoutEffect(() => {
    if (auth?.isLoggedIn && auth.user) {
      toast.loading("Loading Chats", { id: "loadchats" });
      getUserChats()
        .then((data) => {
          setChatMessages([...data.chats]);
          toast.success("Successfully loaded chats", { id: "loadchats" });
        })
        .catch((err) => {
          console.log(err);
          toast.error("Loading Failed", { id: "loadchats" });
        });
    }
```

```
  useEffect(() => {
    const checkAuth = async () => {
      if (!auth?.user) {
        navigate("/login");
      }
    };
    checkAuth();
  }, [auth]);
  return (
    <Box
      sx={{
        display: "flex",
        flex: 1,
        width: "100%",
        height: "100%",
        mt: 3,
        gap: 3,
      }}
    >
      <Box
        sx={{
          display: { md: "flex", xs: "none", sm: "none" },
          flex: 0.2,
          flexDirection: "column",
        }}
      >
        <Box
          sx={{
            display: "flex",
            width: "100%",
            height: "60vh",
            bgcolor: "rgb(17,29,39)",
            borderRadius: 5,
            flexDirection: "column",
            mx: 3,
          }}
        >
          <Avatar
            sx={{
              mx: "auto",
              my: 2,
              bgcolor: "white",
              color: "black",
              fontWeight: 700,
            }}
          >
            {auth?.user?.name[0]}
            {auth?.user?.name.split("")}
          </Avatar>
          <Typography sx={{ mx: "auto", fontFamily: "work sans" }}>
            You are talking to a ChatBOT
          </Typography>
          <Typography sx={{ mx: "auto", fontFamily: "work sans", my: 4, p: 3 }}>
            You can ask some questions related to Knowledge, Business, Advices,
            Education, etc. But avoid sharing personal information
          </Typography>
          <Button
            onClick={handleDeleteChats}
```

```
            sx={{
              width: "200px",
              my: "auto",
              color: "white",
              fontWeight: "700",
              borderRadius: 3,
              mx: "auto",
              bgcolor: red[300],
              ":hover": {
                bgcolor: red.A400,
              },
            }}
          >
            Clear Conversation
          </Button>
        </Box>
      </Box>
      <Box
        sx={{
          display: "flex",
          flex: { md: 0.8, xs: 1, sm: 1 },
          flexDirection: "column",
          px: 3,
        }}
      >
        <Typography
          sx={{
            fontSize: "40px",
            color: "white",
            mb: 2,
            mx: "auto",
            fontWeight: "600",
          }}
        >
          Model - GPT 3.5 Turbo
        </Typography>
        <Box
          sx={{
            width: "100%",
            height: "60vh",
            borderRadius: 3,
            mx: "auto",
            display: "flex",
            flexDirection: "column",
            overflow: "scroll",
            overflowX: "hidden",
            overflowY: "auto",
            scrollBehavior: "smooth",
          }}
        >
          {chatMessages.map((chat, index) => (
            //@ts-ignore
            <ChatItem content={chat.content} role={chat.role} key={index} />
          ))}
        </Box>
        <div
          style={{
```

```
            style={{
              width: "100%",
              borderRadius: 8,
              backgroundColor: "rgb(17,27,39)",
              display: "flex",
              margin: "auto",
            }}
          >
            {" "}
            <input
              ref={inputRef}
              type="text"
              style={{
                width: "100%",
                backgroundColor: "transparent",
                padding: "30px",
                border: "none",
                outline: "none",
                color: "white",
                fontSize: "20px",
              }}
            />
            <IconButton onClick={handleSubmit} sx={{ color: "white", mx: 1 }}>
              <IoMdSend />
            </IconButton>
          </div>
        </Box>
      </Box>
    );
};

export default Chat;
```

ДОДАТОК Д. КЛАС API ФРОНТ-ЕНДУ

```
import axios from "axios";  62.6k (gzipped: 23.2k)
export const loginUser = async (email: string, password: string) => {
  const res = await axios.post("/user/login", { email, password });
  if (res.status !== 200) {
    throw new Error("Unable to login");
  }
  const data = await res.data;
  return data;
};

export const signupUser = async (
  name: string,
  email: string,
  password: string
) => {
  const res = await axios.post("/user/signup", { name, email, password })
  if (res.status !== 201) {
    throw new Error("Unable to Signup");
  }
  const data = await res.data;
  return data;
};

export const checkAuthStatus = async () => {
  try {
    const res = await axios.get("/user/auth-status");
    if (res.status !== 200) {
      throw new Error("Unable to authenticate");
    }
    const data = await res.data;
    return data;
  } catch (error: any) {
    if (error.response && error.response.status === 500) {
      throw new Error("Server error, please try again later.");
    }
    throw error;
  }
};

export const sendChatRequest = async function (message: string) {
  try {
    const res = await axios.post("/chat/new", { params: { message } });
    if (res.status !== 200) {
      throw new Error("Unable to send chat");
    }
    const data = await res.data;
    return data;
  } catch (error: any) {
    if (error.response && error.response.status === 500) {
      throw new Error("Server error, please try again later.");
    }
    throw error;
  }
};
```

```
export const getUserChats = async () => {
  const res = await axios.get("/chat/all-chats");
  if (res.status !== 200) {
    throw new Error("Unable to send chat");
  }
  const data = await res.data;
  return data;
};

export const deleteUserChats = async () => {
  const res = await axios.delete("/chat/delete");
  if (res.status !== 200) {
    throw new Error("Unable to delete chats");
  }
  const data = await res.data;
  return data;
};

export const logoutUser = async () => {
  const res = await axios.get("/user/logout");
  if (res.status !== 200) {
    throw new Error("Unable to delete chats");
  }
  const data = await res.data;
  return data;
};
```

ДОДАТОК Е. КЛАС АУТЕНТИФІКАЦІЇ КОРИСТУВАЧА

```tsx
import {
  ReactNode,
  createContext,
  useContext,
  useEffect,
  useState,
} from "react";  5k (gzipped: 2k)
import {
  checkAuthStatus,
  loginUser,
  logoutUser,
  signupUser,
} from "../helpers/api-communicator";

type User = {
  name: string;
  email: string;
};
type UserAuth = {
  isLoggedIn: boolean;
  user: User | null;
  login: (email: string, password: string) => Promise<void>;
  signup: (name: string, email: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
};
const AuthContext = createContext<UserAuth | null>(null);

export const AuthProvider = ({ children }: { children: ReactNode }) => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  useEffect(() => {
    // fetch if the user's cookies are valid then skip login
    async function checkStatus() {
      const data = await checkAuthStatus();
      if (data) {
        setUser({ email: data.email, name: data.name });
        setIsLoggedIn(true);
      }
    }
    checkStatus();
  }, []);
```

```
43    const login = async (email: string, password: string) => {
44      const data = await loginUser(email, password);
45      if (data) {
46        setUser({ email: data.email, name: data.name });
47        setIsLoggedIn(true);
48      }
49    };
50    const signup = async (name: string, email: string, password: string) => {
51      const data = await signupUser(name, email, password);
52      if (data) {
53        setUser({ email: data.email, name: data.name });
54        setIsLoggedIn(true);
55      }
56    };
57    const logout = async () => {
58      await logoutUser();
59      setIsLoggedIn(false);
60      setUser(null);
61      window.location.reload();
62    };
63
64    const value = {
65      user,
66      isLoggedIn,
67      login,
68      logout,
69      signup,
70    };
71    return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
72  };
73
74  export const useAuth = () => useContext(AuthContext);
```

ДОДАТОК Є. КЛАС ОСНОВНОГО ФАЙЛУ СЕРВЕРА

```
import express from "express";
import { config } from "dotenv";  6.8k (gzipped: 3k)
import morgan from "morgan";  14.2k (gzipped: 5.3k)
import appRouter from "./routes/index.js";
import cookieParser from "cookie-parser";  5k (gzipped: 2k)
import cors from "cors";  5k (gzipped: 2.1k)
config();
const app = express();

//middlewares
app.use(cors({ origin: "http://localhost:5173", credentials: true }));
app.use(express.json());
app.use(cookieParser(process.env.COOKIE_SECRET));

//remove it in production
app.use(morgan("dev"));

app.use("/api/v1", appRouter);

export default app;
```

ДОДАТОК Ж. КЛАС КОНТРОЛЕРУ ЧАТА

```
import { NextFunction, Request, Response } from "express";
import User from "../models/User.js";
import { configureOpenAI } from "../config/openai-config.js";
import { OpenAIApi, ChatCompletionRequestMessage, Configuration, ConfigurationParameters } from "openai";  65.2k (gzipped: 16.4k)
export const generateChatCompletion = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const { message } = req.body;
  try {
    const user = await User.findById(res.locals.jwtData.id);
    if (!user)
      return res
        .status(401)
        .json({ message: "User not registered OR Token malfunctioned" });
    // беремо чати з бази даних
    // та формуємо їх у формат, який приймає openAI API
    const chats = user.chats.map(({ role, content }) => ({
      role,
      content,
    })) as ChatCompletionRequestMessage[];
    chats.push({ content: message, role: "user" });
    user.chats.push({ content: message, role: "user" });

    // відправляємо всі чати з новим
    // чатом до openAI API
    const config = configureOpenAI();
    const configuration = new Configuration(config as unknown as ConfigurationParameters);
    const openai = new OpenAIApi(configuration);
    // отримуємо відповідь від openAI
    const chatResponse = await openai.createChatCompletion({
      model: "gpt-3.5-turbo",
      messages: chats,
    });
    user.chats.push(chatResponse.data.choices[0].message);
    await user.save();
    return res.status(200).json({ chats: user.chats });
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message: "Something went wrong" });
  }
};
```

```
45  export const sendChatsToUser = async (
46    req: Request,
47    res: Response,
48    next: NextFunction
49  ) => {
50    try {
51      //перевірка токена користувача
52      const user = await User.findById(res.locals.jwtData.id);
53      if (!user) {
54        return res.status(401).send("User not registered OR Token malfunctioned");
55      }
56      if (user._id.toString() !== res.locals.jwtData.id) {
57        return res.status(401).send("Permissions didn't match");
58      }
59      return res.status(200).json({ message: "OK", chats: user.chats });
60    } catch (error) {
61      console.log(error);
62      return res.status(200).json({ message: "ERROR", cause: error.message });
63    }
64  };
65
66  export const deleteChats = async (
67    req: Request,
68    res: Response,
69    next: NextFunction
70  ) => {
71    try {
72      //перепровірка токена
73      const user = await User.findById(res.locals.jwtData.id);
74      if (!user) {
75        return res.status(401).send("User not registered OR Token malfunctioned");
76      }
77      if (user._id.toString() !== res.locals.jwtData.id) {
78        return res.status(401).send("Permissions didn't match");
79      }
80      //@ts-ignore
81      user.chats = [];
82      await user.save();
83      return res.status(200).json({ message: "OK" });
84    } catch (error) {
85      console.log(error);
86      return res.status(200).json({ message: "ERROR", cause: error.message });
87    }
88  };
```

ДОДАТОК З. КЛАС КОНТРОЛЕРУ КОРИСТУВАЧА

```
import { NextFunction, Request, Response } from "express";
import User from "../models/User.js";
import { configureOpenAI } from "../config/openai-config.js";
import { OpenAIApi, ChatCompletionRequestMessage, Configuration, ConfigurationParameters } from "openai";  65.2k (gzipped: 16.4k)
export const generateChatCompletion = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const { message } = req.body;
  try {
    const user = await User.findById(res.locals.jwtData.id);
    if (!user)
      return res
        .status(401)
        .json({ message: "User not registered OR Token malfunctioned" });
    // беремо чати з бази даних
    // та формуємо їх у формат, який приймає openAI API
    const chats = user.chats.map(({ role, content }) => ({
      role,
      content,
    })) as ChatCompletionRequestMessage[];
    chats.push({ content: message, role: "user" });
    user.chats.push({ content: message, role: "user" });

    // відправляємо всі чати з новим
    // чатом до openAI API
    const config = configureOpenAI();
    const configuration = new Configuration(config as unknown as ConfigurationParameters);
    const openai = new OpenAIApi(configuration);
    // отримуємо відповідь від openAI
    const chatResponse = await openai.createChatCompletion({
      model: "gpt-3.5-turbo",
      messages: chats,
    });
    user.chats.push(chatResponse.data.choices[0].message);
    await user.save();
    return res.status(200).json({ chats: user.chats });
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message: "Something went wrong" });
  }
};
```

```
45  export const sendChatsToUser = async (
46    req: Request,
47    res: Response,
48    next: NextFunction
49  ) => {
50    try {
51      //перевірка токена користувача
52      const user = await User.findById(res.locals.jwtData.id);
53      if (!user) {
54        return res.status(401).send("User not registered OR Token malfunctioned");
55      }
56      if (user._id.toString() !== res.locals.jwtData.id) {
57        return res.status(401).send("Permissions didn't match");
58      }
59      return res.status(200).json({ message: "OK", chats: user.chats });
60    } catch (error) {
61      console.log(error);
62      return res.status(200).json({ message: "ERROR", cause: error.message });
63    }
64  };
65
66  export const deleteChats = async (
67    req: Request,
68    res: Response,
69    next: NextFunction
70  ) => {
71    try {
72      //перепровірка токена
73      const user = await User.findById(res.locals.jwtData.id);
74      if (!user) {
75        return res.status(401).send("User not registered OR Token malfunctioned");
76      }
77      if (user._id.toString() !== res.locals.jwtData.id) {
78        return res.status(401).send("Permissions didn't match");
79      }
80      //@ts-ignore
81      user.chats = [];
82      await user.save();
83      return res.status(200).json({ message: "OK" });
84    } catch (error) {
85      console.log(error);
86      return res.status(200).json({ message: "ERROR", cause: error.message });
87    }
88  };
```

ДОДАТОК И. КЛАС МАРШРУТИЗАЦІЇ ЧАТУ

```
import { Router } from "express";
import { verifyToken } from "../utils/token-manager.js";
import { chatCompletionValidator, validate } from "../utils/validators.js";
import {
  deleteChats,
  generateChatCompletion,
  sendChatsToUser,
} from "../controllers/chat-controllers.js";

//Захищене API для чату
// Використовуємо токен для перевірки доступу до API
const chatRoutes = Router();
chatRoutes.post(
  "/new",
  validate(chatCompletionValidator),
  verifyToken,
  generateChatCompletion
);
chatRoutes.get("/all-chats", verifyToken, sendChatsToUser);
chatRoutes.delete("/delete", verifyToken, deleteChats);

export default chatRoutes;
```

ДОДАТОК І. КЛАС МАРШРУТИЗАЦІЇ КОРИСТУВАЧА

```
import { Router } from "express";
import {
  getAllUsers,
  userLogin,
  userLogout,
  userSignup,
  verifyUser,
} from "../controllers/user-controllers.js";
import {
  loginValidator,
  signupValidator,
  validate,
} from "../utils/validators.js";
import { verifyToken } from "../utils/token-manager.js";

const userRoutes = Router();

userRoutes.get("/", getAllUsers);
userRoutes.post("/signup", validate(signupValidator), userSignup);
userRoutes.post("/login", validate(loginValidator), userLogin);
userRoutes.get("/auth-status", verifyToken, verifyUser);
userRoutes.get("/logout", verifyToken, userLogout);

export default userRoutes;
```

ДОДАТОК Ї. КЛАС ВАЛІДАЦІЇ КОРИСТУВАЧА

```
import { NextFunction, Request, Response } from "express";
import { body, ValidationChain, validationResult } from "express-validator";  218.7k (gzipped: 68.7k)

export const validate = (validations: ValidationChain[]) => {
  return async (req: Request, res: Response, next: NextFunction) => {
    for (let validation of validations) {
      const result = await validation.run(req);
      if (!result.isEmpty()) {
        break;
      }
    }
    const errors = validationResult(req);
    if (errors.isEmpty()) {
      return next();
    }
    return res.status(422).json({ errors: errors.array() });
  };
};

export const loginValidator = [
  body("email").trim().isEmail().withMessage("Email is required"),
  body("password")
    .trim()
    .isLength({ min: 6 })
    .withMessage("Password should contain atleast 6 characters"),
];

export const signupValidator = [
  body("name").notEmpty().withMessage("Name is required"),
  ...loginValidator,
];

export const chatCompletionValidator = [
  body("message").notEmpty().withMessage("Message  is required"),
];
```

ДОДАТОК Й. КЛАС МОДЕЛІ КОРИСТУВАЧА

```
import mongoose from "mongoose";  839.9k (gzipped: 226.4k)
import { randomUUID } from "crypto";
const chatSchema = new mongoose.Schema({
  id: {
    type: String,
    default: randomUUID(),
  },
  role: {
    type: String,
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
});
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  chats: [chatSchema],
});

export default mongoose.model("User", userSchema);
```

ДОДАТОК К. КЛАС З’ЄДНАННЯ БАЗИ ДАНИХ ІЗ СЕРВЕРОМ

```
import { connect, disconnect } from "mongoose";  840k (gzipped: 226.5k)
async function connectToDatabase() {
  try {
    await connect(process.env.MONGODB_URL);
  } catch (error) {
    console.log(error);
    throw new Error("Could not Connect To MongoDB");
  }
}

async function disconnectFromDatabase() {
  try {
    await disconnect();
  } catch (error) {
    console.log(error);
    throw new Error("Could not Disconnect From MongoDB");
  }
}

export { connectToDatabase, disconnectFromDatabase };
```

ДОДАТОК Л. КЛАС ІНТЕГРАВОНОГО РУШІЯ ШТУЧНОГО ІНТЕЛЕКТУ

```
1   import { Configuration, OpenAIApi } from "openai";
2
3   export const configureOpenAI = () => {
4     const configuration = new Configuration({
5       apiKey: process.env.OPEN_AI_SECRET,
6       organization: process.env.OPEN_AI_ORGANIZATION,
7     });
8     const openai = new OpenAIApi(configuration);
9     return openai;
10  };
```