МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН-КОНСУЛЬТАНТА ДЛЯ ВЕБСАЙТІВ ІЗ ЗАСТОСУВАННЯМ WEBSOCKET»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
І.Є. Дробний
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Дробному Іллі Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн-консультанта для вебсайтів із застосуванням WebSocket

керівник роботи Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка онлайн консультанта для вебсайтів з використанням WebSocket.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	26.01.2024	
2.	Збір вихідних даних.	16.02.2024	
3.	Обробка методичних та теоретичних	25.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	26.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2024	

Студент _____ І.Є. Дробний
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Класифікаційна робота «Розробка онлайн-консультанта для вебсайтів із застосуванням WebSocket»: 45 сторінок, 9 рисунків, 12 джерел, 2 додатки.

ВЕБСАЙТ, МЕТОДИКА РОЗРОБКИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОНЛАЙН-КОНСУЛЬТАНТ, WEBSOCKET.

Об’єкт дослідження: програмне забезпечення на основі технології WebSocket.

Предмет дослідження: розробка онлайн консультанта для вебсайтів з використанням WebSocket.

Мета роботи: розробка програмного забезпечення для надання онлайн консультацій на вебсайтах за допомогою технології WebSocket.

Метод дослідження: аналіз, порівняння, описовий та структурний аналіз.

Розглянуто етапи розробки онлайн консультанта для вебсайтів з використанням WebSocket, виявлено ключові аспекти та застосовано нові методи для забезпечення ефективної розробки програмного забезпечення. Ці зміни спрямовані на поліпшення управління процесом розробки та підвищення ефективності програмного забезпечення.

Робота базується на дослідженні процесу розробки вебдодатків та виявлених труднощів, що виникають під час цього процесу.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Consultant for Websites Using WebSocket”: 45 pages, 9 figures, 12 references, 2 supplements.

WEBSITE, DEVELOPMENT METHODOLOGY, SOFTWARE, ONLINE CONSULTANT, WEBSOCKET.

Object of study: software based on WebSocket technology.

Subject of study: development of an online consultant for websites using WebSocket.

Purpose of the work: development of software for providing online consultations on websites using WebSocket technology.

Research method: analysis, comparison, descriptive, and structural analysis.

The stages of developing an online consultant for websites using WebSocket have been considered, key aspects have been identified, and new methods have been applied to ensure effective software development. These changes are aimed at improving the management of the development process and increasing the efficiency of the software.

The work is based on the study of the web application development process and the difficulties encountered during this process.

ЗМІСТ

ВСТУП

Зворотний зв’язок від користувачів є ключовим елементом будь-якого вебсайту, особливо в інтернет-магазині. У сучасному цифровому світі, де конкуренція безмежна, важливо не лише привабити відвідувачів на вебсайт, а й зробити їх перебування на ньому якнайбільш приємним та продуктивним. Розробка онлайн консультанта на основі WebSocket відповідає цьому завданню, створюючи можливість для реального часу взаємодії між користувачами та адміністраторами сайту.

Завдяки WebSocket, вебсайт може забезпечити інтерактивність та миттєву відповідь на запитання відвідувачів, не потребуючи перезавантаження сторінок. Це дає користувачам відчуття оперативності та особистого обслуговування, що позитивно впливає на їх враження від взаємодії з сайтом.

У процесі розробки такого консультанта необхідно враховувати потреби користувачів та забезпечувати їх якісними та оперативними відповідями. Передбачення різноманітних сценаріїв взаємодії та розробка зручного інтерфейсу важливі для успішної реалізації проєкту.

Такий онлайн консультант може стати не тільки інструментом підтримки користувачів, але й ефективним засобом збору даних та аналізу поведінки відвідувачів. Використання WebSocket дозволить збирати дані про запитання, зацікавленість та проблеми в реальному часі, що надають можливість адаптувати сайт до потреб аудиторії та підвищити його ефективність.

1 ТЕОРЕТИЧНИЙ ОГЛЯД ТА ВИВЧЕННЯ WEBSOCKET

1.1 WebSocket та їх використання

WebSocket – це двонаправлений повнодуплексний протокол зв’язку між клієнтом та сервером. Що це значить? На відміну від HTTP-протоколу, який працює за принципом «запит від клієнта – відповідь від сервера», у вебсокетах і сервер, і клієнт можуть надсилати один одному повідомлення.

Основними особливостями WebSocket є:

– **двосторонній зв’язок в режимі реального часу:** WebSocket дозволяє як клієнту, так і серверу надсилати повідомлення один одному в будь-який момент, без необхідності чекати відповіді – це забезпечує миттєвий обмін даними в обох напрямках;

– **легкість впровадження:** WebSocket працює на рівні протоколу, що робить його простим у реалізації та інтеграції в вебдодатки – розробники можуть швидко підключити WebSocket до своїх проєктів без складних налаштувань чи додаткових залежностей;

– **постійне з’єднання:** WebSocket підтримує постійне з’єднання між клієнтом і сервером, яке залишається активним протягом всього сеансу обміну даними – це відрізняє його від протоколу HTTP, де кожен запит вимагає встановлення нового з’єднання.

У моделі WebSocket існує відношення між клієнтом і сервером. Клієнт – це браузер або будь-який інший клієнтський додаток, що взаємодіє з сервером через WebSocket. Сервер – це програма або служба, яка приймає запити від клієнтів, обробляє їх і передає відповіді через WebSocket з’єднання. Ця модель дозволяє встановити постійне з’єднання між клієнтом і сервером. При встановленні з’єднання відбувається рукостискання (handshake) між клієнтом і сервером, що дозволяє їм підтвердити, що вони готові до взаємодії. Після встановлення з’єднання обидві сторони можуть передавати повідомлення одна одній в реальному часі.

Кожна сторона комунікації здатна одночасно отримувати та відправляти дані (див. рис. 1.1).

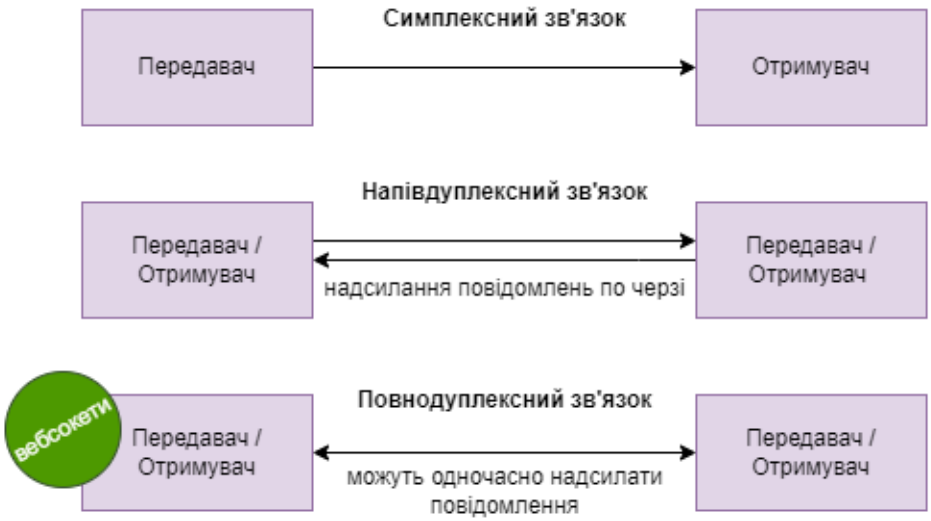

Рисунок 1.1 – Схема з’єднання WebSocket

У вебсокетах обмін повідомленнями проходить через єдиний канал зв’язку. Він залишається відкритим протягом усієї комунікації, а за необхідності будь-яка зі сторін може його закрити.

Вебсокет протокол існує як надбудова TCP. Специфікація визначає дві URI схеми для вебсокетів: ws:// для нешифрованого з’єднання та wss:// відповідно – для шифрованого. Протокол складається з початкового хендшейку і безпосередньо обміну даними.

У код-блоці, який наведено на рисунку 1.2 можемо розглянути, як виглядає handshake з боку клієнта. Тут присутній header Connection: Upgrade. Також видно, який саме upgrade пропонується, – Upgrade: websocket.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 8
```

Рисунок 1.2 – Приклад HTTP-запиту для встановлення WebSocket-з’єднання

Сервер підтверджує handshake статус кодом 101 – зміна протоколу, і так само надсилає деталі про новий connection (див. рис. 1.3).

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Рисунок 1.3 – HTTP-відповідь сервера на запит про встановлення WebSocket-з’єднання

Дані надсилаються у вигляді фреймів із заданим типом. Кожне повідомлення може складатися з одного чи більше фреймів. Усі вони повинні бути однакового типу. Такими типами можуть бути текст, бінарні дані та контроль фрейми, призначені не для передачі даних, а для службових сигналів на рівні протоколу. Наприклад, що з’єднання потрібно закрити.

Socket.IO – найпопулярніших інструментів для роботи з вебсокет протоколом. Він складається з вебсокет сервера та клієнтської бібліотеки. Спершу вони були реалізовані на JavaScript, але згодом з’явилися реалізації на багатьох інших мовах програмування. Socket.IO реалізує додатковий функціонал, якого немає у чистому WebSocket. Наприклад, автоматичні перепідключення, якщо з’єднання втрачене; або fallback до HTTP long polling, якщо WebSocket протокол не підтримується; а також впровадження неймспейсів і кімнат.

Namespaces – необхідні для розділення відповідальності (separation of concerns) у межах одного застосунку. Socket.IO дозволяє створювати декілька неймспейсів, які поводитимуться як окремі канали комунікації. Водночас під капотом вони будуть використовувати одне й те саме з’єднання. Розділяти на неймспейси може бути логічно за модулями у програмі або, наприклад, за спільними пермішенами.

Кімнати – це другий рівень ієрархії. У кожному неймспейсі, у тому числі дефолтному, можна створювати окремі канали – так звані кімнати, до яких

клієнти можуть долучатися та виходити з них. Таким чином, можна транслювати повідомлення у «кімнату», і його отримають всі клієнти, котрі приєдналися до неї. Це може бути зручно для того, щоб одночасно відправити повідомлення групі користувачів або зібрати повідомлення з декількох девайсів для одного юзера.

Якщо нам постійно потрібно слідкувати за оновленням інформації на сервері, в деяких випадках це можна реалізувати за допомогою HTTP-протоколу – у вигляді HTTP Polling.

Існує декілька видів HTTP Polling.

Short polling – простий підхід, однак вважається поганою практикою. Клієнт постійно перепитує сервер, чи готова запитувана інформація. Сервер обробляє реквести щойно вони надходять і відповідає пустим респонсом, якщо дані не готові. В такому випадку велика кількість реквестів перевантажує сервер.

Long polling – клієнт надсилає один реквест на сервер і очікує відповіді. Сервер своєю чергою затримує реквест, допоки потрібні дані не стануть доступними чи у реквеста не закінчиться заданий таймаут. За оптимальних умов отримуємо відповідь щойно на сервері змінюються дані й не створюємо так багато трафіку, як при short polling. Однак на практиці доволі складно налаштувати відправку реквестів так, щоб вони не були ні занадто частими (і в більшості випадків не отримували відповіді), ні занадто повільними, тобто довго затримувались на сервері й марно витрачали ресурси.

У результаті HTTP polling – не дуже зручний підхід, якщо нам вкрай важливо отримувати інформацію в реальному часі.

Оскільки вебсокети підтримують двонаправлений зв’язок, вони ідеально підходять для ситуацій, коли потрібен швидкий двосторонній обмін даними. Це онлайн-ігри, чати, фінансові застосунки, новини, обмін даними з IoT-девайсами.

Не варто використовувати вебсокети, якщо немає потреби в інтерактивності, немає постійного двостороннього трафіку, а натомість – є високі вимоги щодо безпеки. Адже довготривале відкрите з’єднання додає чимало ризиків.

Як будь-яка комунікація в мережі, комунікація по вебсокет протоколу має певні вразливості.

Виділяють такі можливі напрями атак:

– XSS (cross site scripting), SQL-ін’єкції – впровадження шкідливого коду у повідомлення;
– Man-in-the-middle – перехоплення інформації з каналу зв’язку;
– DoS (Denial of Service) – відправка великої кількості запитів, аби зробити ресурс недоступним користувачам;
– неавторизований доступ до інформації.

Захищатися від цих атак можна багатьма способами, але більш надійним може бути додавання аутентифікації на етапі хендшейку. Один із варіантів – використовувати ticket-based аутентифікацію. Це коли клієнт перед апгрейдом з’єднання контактує із HTTP-сервером, який генерує ticket з необхідною інформацією про юзера. Далі клієнт надсилає цей ticket WebSocket-серверу, котрий валідує його і тільки потім надає згоду на з’єднання.

WebSocket використовує спеціальний протокол, який ґрунтується на TCP (Transmission Control Protocol). Цей протокол має власні правила та формати повідомлень, які забезпечують ефективну взаємодію між клієнтом і сервером. Використання WebSocket дозволяє обмінюватися повідомленнями в реальному часі з мінімальними витратами на обробку запитів [11].

1.2 Можливості WebSocket у розробці вебзастосунків

У сучасному світі сфера розробки вебзастосунків розвивається надзвичайно швидко. Користувачі очікують від вебсайтів не лише статичної інформації, але й інтерактивних елементів, швидкої реакції на їх дії та зручної комунікації в реальному часі. Одним з найефективніших рішень для реалізації таких можливостей є використання технології WebSocket. Цей протокол дозволяє встановити двостороннє з’єднання між клієнтом та сервером, що забезпечує обмін даними у режимі реального часу з мінімальною затримкою [12].

WebSocket надає ряд важливих можливостей, які роблять його незамінним у розробці сучасних вебзастосунків:

– двостороннє з’єднання: WebSocket забезпечує постійне двостороннє з’єднання між клієнтом і сервером – це означає, що як клієнт, так і сервер можуть надсилати повідомлення один одному в будь-який час без необхідності повторного встановлення з’єднання;
– реальний час: WebSocket ідеально підходить для застосунків, що вимагають обміну даними в реальному часі, таких як чати, онлайн-ігри, біржові системи та системи моніторингу – завдяки WebSocket дані передаються миттєво, без затримок, властивих іншим протоколам;
– зменшення навантаження на сервер: завдяки постійному з’єднанню, WebSocket зменшує кількість HTTP-запитів до серверу – це значно знижує навантаження на сервер та зменшує використання мережевих ресурсів;
– ефективність і швидкість: WebSocket використовує менше ресурсів у порівнянні з традиційними методами, такими як AJAX-полінг – з’єднання встановлюється лише один раз, і дані передаються у вигляді невеликих пакетів, що робить обмін інформацією швидшим і ефективнішим;
– підтримка стандартів: WebSocket підтримується всіма сучасними браузерами та має стандартизований API, що робить його доступним для широкого використання у веброзробці;
– безпека: WebSocket забезпечує безпечний обмін даними за допомогою протоколу WSS (WebSocket Secure), який використовує SSL/TLS для шифрування з’єднання [1].

Розробка онлайн консультанта для вебсайтів з використанням WebSocket надає значні переваги:

– миттєве спілкування: завдяки WebSocket, оператори та клієнти можуть спілкуватися у реальному часі без затримок – це забезпечує високу якість обслуговування та швидке реагування на запити клієнтів;

- сповіщення в реальному часі: WebSocket дозволяє надсилати миттєві оповіщення про нові повідомлення, що забезпечує зручність та ефективність роботи операторів;
- інтерактивність: використання WebSocket дозволяє реалізувати різні інтерактивні функції, такі як спільне заповнення форм, відстеження активності клієнтів на сайті та надання контекстної допомоги;
- масштабованість: завдяки ефективному використанню ресурсів WebSocket допомагає зменшити навантаження на сервер та забезпечити масштабованість додатку навіть при великій кількості одночасних з’єднань.

Використання WebSocket у розробці вебзастосунків відкриває нові можливості для створення інтерактивних, ефективних і масштабованих рішень. Розробка онлайн консультанта для вебсайтів з використанням WebSocket дозволяє забезпечити високу якість обслуговування користувачів, миттєву комунікацію та зручність у використанні, що є важливими аспектами для сучасних вебдодатків [2].

1.3 Висновки до розділу 1

У першому розділі було проведено теоретичний огляд та вивчення технології WebSocket, що є невід’ємною складовою сучасної веброзробки. Розглянуто основні принципи функціонування WebSocket та його відмінності від традиційних методів обміну даними, таких як HTTP-запити [9].

Основні концепції WebSocket включають встановлення постійного двостороннього з’єднання між клієнтом і сервером, що дозволяє обмінюватися даними в реальному часі з мінімальними затримками. Це забезпечує високу ефективність і швидкість роботи вебзастосунків, особливо у випадках, коли необхідна миттєва реакція на дії користувачів або обробка великих обсягів даних.

Можливості WebSocket у розробці вебзастосунків відкривають нові горизонти для створення інтерактивних та продуктивних рішень. Використання цієї технології дозволяє значно зменшити навантаження на сервер, підвищити ефективність передачі даних і забезпечити високий рівень інтерактивності. Особливо це актуально для розробки вебзастосунків, які потребують реального часу, таких як чати, онлайн-ігри, системи моніторингу та біржові платформи.

Таким чином, у цьому розділі було встановлено теоретичне підґрунтя для подальшої розробки онлайн консультанта для вебсайтів з використанням WebSocket, що дозволить забезпечити високу якість обслуговування користувачів і ефективну взаємодію в реальному часі [10].

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

2.1 Стек технологій для розробки онлайн консультанта

У процесі розробки вебзастосунків, особливо таких, що потребують функціональності в реальному часі, важливо вибрати оптимальний стек технологій. Розглянемо чотири популярні стеки, які можуть бути використані для розробки онлайн консультанта з підтримкою WebSocket:

- MEAN (MongoDB, Express.js, Angular, Node.js);
- MERN (MongoDB, Express.js, React, Node.js);
- LAMP (Linux, Apache, MySQL, PHP) із використанням WebSocket бібліотек;
- .NET стек (C#, ASP.NET Core, SignalR).

Для розробки нашого проєкту обрано .NET стек, який включає мову програмування C#, вебфреймворк ASP.NET Core та бібліотеку SignalR. Цей вибір був зроблений з кількох ключових причин, які забезпечують ефективність, продуктивність та гнучкість при створенні складних вебзастосунків.

C# є сучасною об’єктно-орієнтованою мовою програмування, яка розроблена корпорацією Microsoft. Вона забезпечує високий рівень безпеки та стабільності, а також підтримку широкого спектру парадигм програмування, включаючи функціональне програмування. Однією з ключових переваг C# є можливість використання асинхронного програмування, що дозволяє створювати високопродуктивні вебзастосунки, які можуть обробляти велику кількість одночасних запитів без блокування потоків [3].

ASP.NET Core є кросплатформним, високопродуктивним вебфреймворком з відкритим кодом, що розроблений для побудови сучасних вебзастосунків, включаючи хмарні та інтернет-застосунки. Однією з головних переваг ASP.NET Core є його кросплатформність, що дозволяє запускати застосунки на різних

операційних системах, таких як Windows, Linux та macOS. Крім того, ASP.NET Core підтримує модульну структуру, що дозволяє розробникам вибирати тільки ті компоненти, які потрібні для конкретного проєкту, що знижує загальну складність та розмір застосунку.

ASP.NET Core також забезпечує високий рівень безпеки за рахунок вбудованих механізмів захисту від загроз, таких як XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery). Фреймворк підтримує просту інтеграцію з іншими технологіями, такими як Entity Framework Core для роботи з базами даних, що забезпечує зручне і ефективне управління даними [6].

SignalR є бібліотекою для ASP.NET, яка дозволяє додавати до вебзастосунків функціонал в реальному часі. Використовуючи SignalR, можна легко реалізувати функції двонаправленого зв’язку між сервером і клієнтом. Це особливо важливо для проєктів, які вимагають миттєвого обміну даними, таких як онлайн-консультанти, чати, системи сповіщень та інші інтерактивні вебзастосунки.

SignalR автоматично визначає та використовує найкращий транспортний протокол (WebSocket, Server-Sent Events, Long Polling), доступний у даному середовищі, що забезпечує максимальну ефективність і мінімальні затримки. Крім того, бібліотека SignalR підтримує масштабування за рахунок використання різних бекендів для зберігання стану з’єднань, таких як Redis або SQL Server.

Переваги .NET стеку

- висока продуктивність: оптимізація на рівні компіляції та виконання коду;
- кросплатформність: можливість запуску застосунків на різних операційних системах;
- безпека: вбудовані механізми захисту від загроз;
- асинхронне програмування: підтримка асинхронного коду для обробки великої кількості одночасних запитів;

– модульність: можливість вибору тільки необхідних компонентів;
– масштабованість: підтримка масштабування через SignalR та інші інструменти;
– інтеграція: легка інтеграція з іншими технологіями та сервісами.

Вибір .NET стек для розробки нашого проєкту забезпечує високу якість, продуктивність та надійність вебзастосунку, що задовольнить потреби сучасних користувачів та вимоги ринку [5].

2.2 Постанова задачі

У сучасному світі онлайн-комунікації набувають дедалі більшої популярності, а бізнес-вебсайти стають місцем, де користувачі шукають інформацію та взаємодіють з компаніями. У цьому контексті наявність онлайн-консультанта, який надає підтримку в режимі реального часу та негайно відповідає на запитання користувачів, має велике значення.

Проте традиційні методи реалізації онлайн-консультанта мають свої обмеження. Зазвичай вони використовують AJAX або HTTP-запити для взаємодії між клієнтом та сервером, що передбачає постійне надсилання запитів з браузера на сервер для отримання оновленої інформації. Такий підхід є неефективним, оскільки створює зайвий трафік і затримки в оновленні даних.

Для розв’язання цих проблем пропонується використання WebSocket-технологію, яка забезпечує постійне двостороннє з’єднання між клієнтом і сервером. WebSocket дозволяє передавати дані в режимі реального часу без необхідності постійного оновлення сторінки.

Наше завдання полягає у розробці онлайн-консультанта для вебсайтів з використанням WebSocket. Планується створити інтерактивний інтерфейс, який дозволить користувачам взаємодіяти з консультантом у режимі реального часу.

Перш за все, необхідно розробити серверну частину, яка відповідатиме за

обробку запитів WebSocket. Для цього використаємо популярний фреймворк або бібліотеку для реалізації серверного коду.

Крім того, потрібно створити клієнтську частину, яка відображатиме інтерфейс консультанта на вебсайті. Розробимо зручний та привабливий дизайн, що забезпечить комфортну взаємодію користувачів з консультантом.

Одним із ключових завдань є реалізація можливості обміну повідомленнями в режимі реального часу між консультантом та користувачем. Забезпечимо швидку та надійну передачу даних через WebSocket, що дозволить уникнути затримок і забезпечить миттєву відповідь.

Також важливо забезпечити можливість аутентифікації та авторизації користувачів, щоб гарантувати конфіденційність та безпеку обміну інформацією між консультантом і клієнтами.

Окрім цього, необхідно надати можливість зберігати історію повідомлень та забезпечити функціональність пошуку і фільтрації для полегшення навігації та аналізу даних. Система повинна бути масштабованою та надійною, щоб працювати з великою кількістю одночасних підключень та забезпечувати стабільну роботу навіть при високому навантаженні.

В результаті розробки онлайн-консультанта з використанням WebSocket планується створити потужний та ефективний інструмент, який забезпечить зручну комунікацію між консультантом і користувачами. Таке рішення дозволить покращити якість обслуговування та підвищити рівень задоволеності клієнтів.

Для більшого розуміння концепції розглянемо функціональні та нефункціональні вимоги для проєкту онлайн консультанта для вебсайтів:

Функціональні вимоги:

– реєстрація та авторизація користувачів;
– інтерактивний чат у реальному часі з можливістю взаємодії з консультантом;
– зберігання історії повідомлень та функція пошуку в ній;
– мобільна сумісність з адаптивним дизайном.

Нефункціональні вимоги:

– висока швидкодія передачі повідомлень і надійність з’єднання;
– захист даних користувачів і конфіденційність інформації;
– масштабованість системи для ефективної роботи при великому навантаженні;
– інтуїтивно зрозумілий і привабливий інтерфейс для користувачів.

Ці вимоги визначають ключові характеристики системи, які дозволять забезпечити якісну та зручну взаємодію між користувачами та консультантами на вебсайті.

2.3 Проєктування бази даних

Створення бази даних для онлайн консультанта з використанням WebSocket може включати в себе використання реляційної системи управління базами даних (СУБД) MySQL. MySQL є популярною вибором для багатьох вебдодатків через свою швидкодію, надійність і широкі можливості налаштування. Ось деякі переваги використання MySQL для цього проєкту:

– швидкодія та ефективність: MySQL відома своєю високою швидкодією операцій з базами даних, що дозволяє ефективно обробляти великі обсяги даних і запитів;
– надійність: MySQL забезпечує стабільну роботу бази даних, маючи вбудовані механізми для відновлення даних в разі відмови системи або інших непередбачуваних ситуацій;
– підтримка транзакцій: MySQL підтримує транзакційну обробку, що дозволяє гарантувати цілісність даних під час одночасної обробки багатьох запитів;
– масштабованість: MySQL може бути легко масштабована, що дозволяє розширювати обсяги бази даних і забезпечувати високу доступність системи при необхідності;

– SQL-сумісність: MySQL підтримує стандартні SQL-запити, що робить розробку і підтримку бази даних більш зручною і стандартизованою.

Таким чином, MySQL є ідеальним вибором для забезпечення структури бази даних для онлайн консультанта з використанням WebSocket, оскільки вона надійно підтримує всі необхідні функціональні можливості і забезпечує ефективну роботу системи в режимі реального часу [4].

2.4 Створення ER-діаграми та діаграм прецедентів

ER-діаграма (Entity-Relationship diagram) – це графічний інструмент, який використовується для моделювання структури даних у базах даних. Вона зображає зв’язки між різними сутностями (таблицями) бази даних та їх атрибути. Основними компонентами ER-діаграми є сутності, зв’язки і атрибути.

Сутності представляють собою прямокутники, які відображають таблиці бази даних. Кожна сутність має свою назву, яка описує концептуальний об’єкт, наприклад, «Користувач» або «Замовлення».

Зв’язки показують взаємозв’язки між сутностями і з’єднують їх за допомогою ліній. Наприклад, зв’язок «Має» може з’єднувати сутності «Користувач» і «Профіль», показуючи, що кожен користувач має один профіль.

Атрибути описують характеристики кожної сутності. Наприклад, у сутності «Користувач» можуть бути атрибути «Ім’я», «Прізвище» і «Електронна пошта».

Основна мета ER-діаграми полягає в тому, щоб чітко визначити структуру бази даних, визначити типи зв’язків між сутностями і встановити правила, які регулюють ці зв’язки. Це дозволяє розробникам і аналітикам легко розуміти взаємозв’язки у базі даних і планувати оптимальну організацію даних для певного додатка або системи.

На рисунку 2.1 зображено ER-діаграма бази даних для нашого онлайн консультанта.

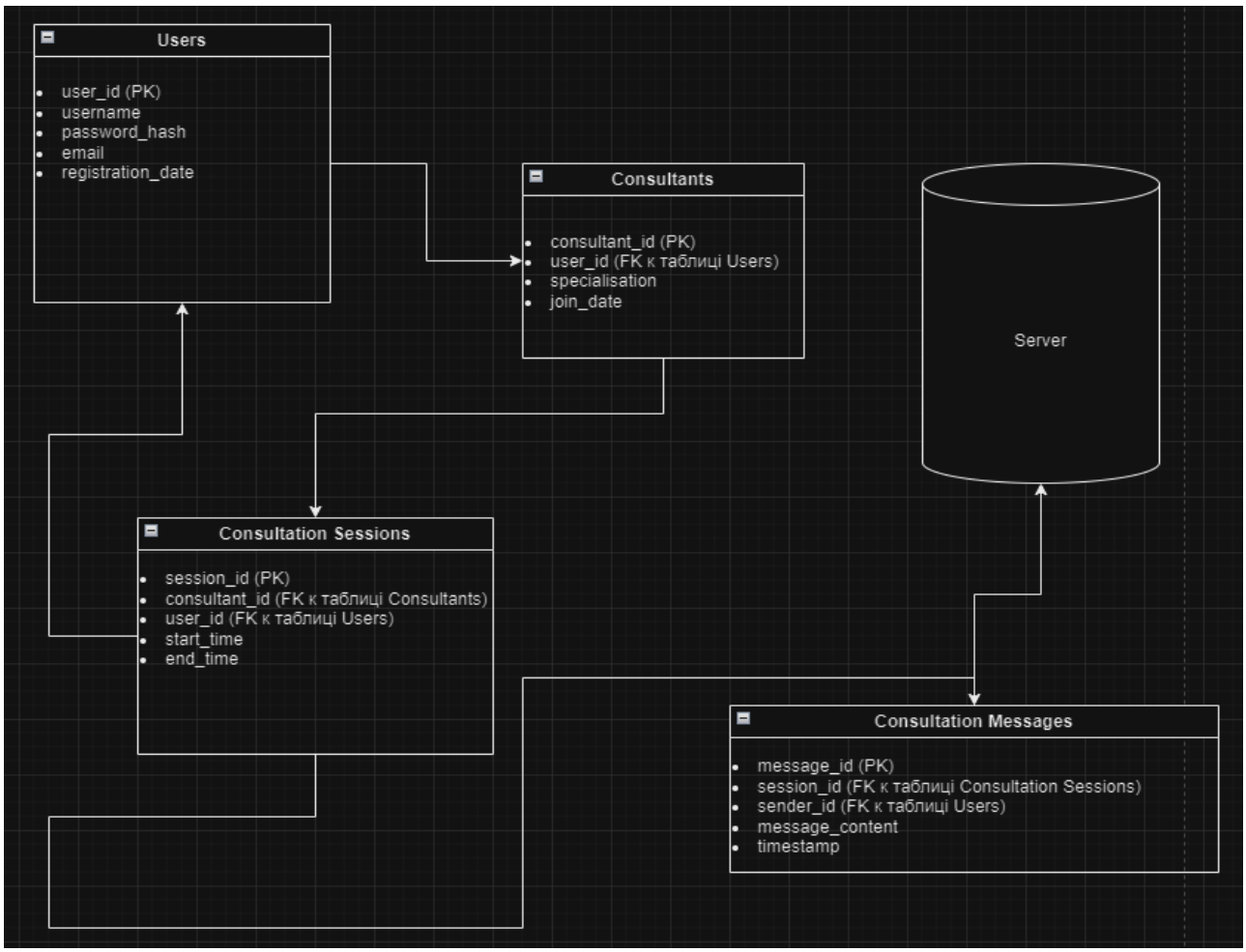

Рисунок 2.1 – ER-діаграма бази даних

Таблиця “Users” (Користувачі):

– user_id (PK): ідентифікатор користувача;

– username: ім’я користувача;

– password_hash: хеш паролю;

– email: електронна пошта;

– registration_date: дата реєстрації.

Таблиця “Consultants” (Консультанти):

– consultant_id (PK): ідентифікатор консультанта;

– user_id (FK, посилання на таблицю Users): ідентифікатор користувача;

– specialisation: спеціалізація;

– join_date: дата приєднання.

Таблиця “Consultation Sessions” (Сеанси консультацій):

– session_id (PK): ідентифікатор сеансу;

- consultant_id (FK, посилання на таблицю Consultants): Ідентифікатор консультанта;
- user_id (FK, посилання на таблицю Users): Ідентифікатор користувача;
- start_time: час початку;
- end_time: час завершення;

Таблиця “Consultation Messages” (Повідомлення консультацій):

- message_id (PK): ідентифікатор повідомлення;
- session_id (FK, посилання на таблицю Consultation Sessions): ідентифікатор сеансу;
- sender_id (FK, посилання на таблицю Users): ідентифікатор відправника;
- message_content: зміст повідомлення;
- timestamp: часова мітка.

Діаграма прецедентів є одним з ключових інструментів моделювання в системному аналізі та проєктуванні програмних систем. Вона використовується для візуалізації функціональності системи з точки зору зовнішніх акторів (користувачів системи) та їх взаємодій з самою системою. Основна мета діаграми прецедентів – це визначення всіх можливих дій, які можуть бути виконані користувачами в рамках системи та визначення їхніх взаємовідносин.

Основні складові діаграми прецедентів включають:

- актори: це сутності або ролі, які взаємодіють з системою (актори можуть бути конкретними користувачами системи, іншими системами, зовнішніми сервісами або будь-якими іншими сутностями, які мають відношення до системи);
- прецеденти: це конкретні дії або функції, які можуть бути виконані акторами у системі (кожен прецедент описується назвою та описом того, що актор може зробити в системі);
- взаємодії: відображаються стрілки, які показують взаємодії між акторами та прецедентами (ці взаємодії показують, як актори взаємодіють з прецедентами для досягнення певної функціональності.

– система: це центральна частина діаграми, яка відображає всі прецеденти та взаємодії між акторами та системою.

Переваги використання діаграм прецедентів включають:

– комунікація: ці діаграми є чітким інструментом для комунікації між розробниками програмного забезпечення та замовниками щодо функціональних вимог системи;
– розуміння вимог: вони допомагають зрозуміти, які функції повинна виконувати система з точки зору користувачів;
– аналіз потоків роботи: вони дозволяють аналізувати різні потоки роботи та взаємодії між користувачами та системою, щоб ідентифікувати можливі проблеми або покращення.

Загалом, діаграма прецедентів є потужним інструментом для аналізу, проєктування та комунікації функціональних вимог до програмного забезпечення, що дозволяє створити зрозумілу та ефективну модель системи.

Для розробки діаграми прецедентів для нашого веб застосунку треба ідентифікувати основні дії користувачів (акторів) і системи, які будуть включені в функціональність консультанта. Розглянемо основні прецеденти, які можуть бути визначені для цього проєкту.

Відправлення повідомлення користувачем:

– актор: користувач;
– опис: користувач може відправляти повідомлення консультанту через інтерфейс вебсайту, повідомлення передаються через WebSocket для миттєвого відображення консультанту.

Отримання повідомлення консультантом:

– актор: консультант;
– опис: консультант отримує повідомлення від користувачів через WebSocket, він може відповідати на ці повідомлення в реальному часі.

Реєстрація користувача:

– актор: новий користувач;
– опис: новий користувач може зареєструватися в системі для отримання

доступу до консультаційних послуг, реєстрація відбувається через стандартну форму вебсайту з використанням WebSocket для підтвердження процесу.

Авторизація користувача:

– актор: користувач;
– опис: користувач може авторизуватися в системі для отримання персоналізованого доступу до консультаційних послуг.

Адміністрування системи:

– актор: адміністратор системи;
– опис: адміністратор має доступ до панелі управління для керування користувачами, консультантами, історією повідомлень та іншими аспектами системи.

Зберігання історії повідомлень:

– актор: система;
– опис: система зберігає історію всіх повідомлень для подальшого доступу користувачами та аналітичних цілей.

На рисунку 2.2 візуалізовано повну діаграму прецедентів нашого проєкту.

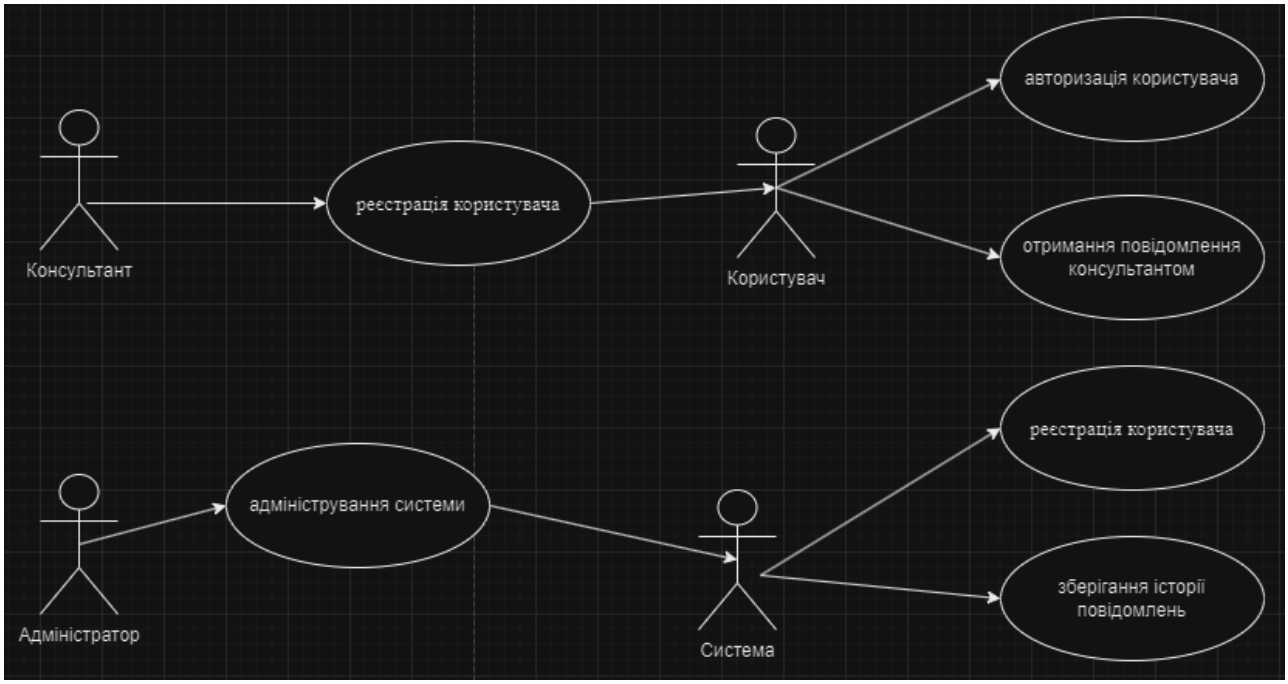

Рисунок 2.2 – Діаграма прецедентів

2.5 Висновки до розділу 2

У результаті проведеного аналізу та проєктування системи онлайн консультанта була розроблена детальна архітектура і необхідні компоненти, необхідні для її реалізації. Визначені взаємозв’язки між цими компонентами та методи їх взаємодії.

Проєктування системи онлайн консультанта сприяє ефективній та структурованій розробці, знижує ризики та гарантує високу якість результуючої системи. Цей процес визначає основні принципи, архітектурні рішення та структуру, необхідні для побудови системи онлайн консультанта.

Результатом проєктування є докладний план, що містить опис компонентів системи, їх функціональність та взаємозв’язки, а також детальні технічні специфікації. Цей план є основою для подальшої розробки та імплементації системи з використанням відповідних технологій та інструментів.

У даному розділі представлені результати проєктування системи онлайн консультанта, які будуть використані для подальшої розробки та впровадження системи. Ці результати включають діаграми архітектури, опис компонентів та їх взаємодію, що є ключовими аспектами для успішної реалізації задуманої системи онлайн консультанта.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

У даному розділі кваліфікаційної роботи розглядається процес реалізації та тестування системи онлайн консультанта, який є ключовим етапом в цьому проєкті. Після проведення аналізу, проєктування та моделювання системи настав час перейти до фази активної розробки, де ідеї перетворюються в живий функціональний продукт.

В цьому розділі докладно описується процес розробки програмного забезпечення для системи онлайн консультанта. Це включає вибір оптимальних технологій, написання коду, налаштування робочого середовища та створення архітектури системи. Кожен етап реалізації розглядається в розрізі фокуса на стабільність, ефективність та надійність системи.

Також у цьому розділі описується процес тестування системи з метою перевірки її функціональності, відповідності вимогам та виявлення помилок. Для цього використовуються різні методи тестування, спрямовані на перевірку окремих компонентів системи та їх взаємодії, з метою забезпечення якісної та безперебійної роботи.

Результатом цього розділу є детальний опис реалізації системи онлайн консультанта, включаючи архітектурні рішення, використані технології, принципи розробки та використані інструменти. Також будуть представлені результати проведеного тестування, його висновки та огляд.

3.1 Основні аспекти серверної частини онлайн консультанта

Основні аспекти серверної частини онлайн консультанта включають ряд ключових аспектів, які забезпечують правильну і ефективну роботу системи.

Архітектура серверної частини: важливо обрати відповідну архітектурну модель, яка забезпечить масштабованість та ефективність системи. Наприклад,

можна використовувати архітектуру з мікросервісами або модульно побудовану систему, яка дозволяє легко розширювати функціонал.

Комунікація через WebSocket: оскільки основна функція онлайн консультанта полягає в реальному часі, використання WebSocket є ключовим елементом. WebSocket забезпечує постійне з'єднання між клієнтом і сервером, що дозволяє обмінюватися даними без необхідності постійного опитування сервера з боку клієнта.

Управління сесіями: для зберігання стану сесії консультанта та користувача важливо використовувати механізми управління сесіями. Це дозволяє зберігати контекст інтеракції, а також забезпечує безпеку та конфіденційність даних.

Безпека та автентифікація: забезпечення безпеки даних та автентифікації користувачів є важливим аспектом. Використання протоколів шифрування, наприклад TLS/SSL, і механізмів автентифікації дозволяє захистити конфіденційні дані.

Управління базою даних: інформація про сесії, історію повідомлень та інші дані повинні зберігатися в базі даних. Важливо обрати підходящу базу даних та розробити ефективну схему її організації для забезпечення швидкодії та надійності.

Моніторинг та логування: для ефективного управління та підтримки системи важливо вести моніторинг роботи серверної частини та забезпечувати логування подій. Це дозволяє оперативно виявляти проблеми та вчасно реагувати на них.

3.2 Моделі та їх реалізація

Моделі в контексті розробки онлайн консультанта є важливим елементом, оскільки вони визначають структуру даних та логіку їх обробки в системі. Реалізація моделей забезпечує коректну роботу всіх компонентів системи, що базуються на цих даних.

Коли клієнт відправляє повідомлення через вебсокети, дані впаковуються в об’єкт моделі Message, який містить такі поля, як ідентифікатор повідомлення, текст, ідентифікатор чату та ідентифікатор користувача. Цей об’єкт моделі потім передається на сервер.

На сервері, отримане через вебсокети повідомлення розпаковується у вигляді об’єкта моделі Message. Це дозволяє серверу отримати доступ до всіх даних цього повідомлення. Після отримання сервер може обробити це повідомлення, зберегти його в базі даних, відправити іншим клієнтам або виконати інші дії, що вимагає логіка додатку.

У випадку, коли сервер відправляє повідомлення клієнту через вебсокети, він також використовує модель Message для упакування даних перед відправленням. Клієнт на своєму кінці розпаковує отримане повідомлення в об’єкт моделі Message, щоб мати доступ до даних та відобразити повідомлення на користувацькому інтерфейсі.

Ось деякі ключові аспекти моделей та їх реалізації для онлайн консультанта:

– структура даних: моделі визначають структуру та типи даних, які система обробляє, наприклад, для системи онлайн консультанта можуть бути такі моделі, як User (користувач), Consultant (консультант), Message (повідомлення), Session (сесія) тощо – кожна модель має свої атрибути, які відображають характеристики об’єкту;
– взаємодія з базою даних: моделі пов’язані з базою даних, де зберігаються їхні екземпляри (для реалізації моделей використовуються ORM (Object-Relational Mapping) бібліотеки, такі як Entity Framework у середовищі .NET, які дозволяють легко і зручно працювати з об’єктами бази даних);
– бізнес-логіка: в моделях часто визначається бізнес-логіка, яка описує правила обробки даних та взаємодії між компонентами системи, наприклад, для моделі Message може бути визначено правило, що нові повідомлення мають відправлятися до консультанта для обробки;

– валідація даних: моделі також відповідають за валідацію вхідних даних перед їх збереженням у базі, вони встановлюють обмеження та правила для кожного поля моделі, що дозволяє попередити збереження невірних або недопустимих даних;
– серіалізація та десеріалізація: моделі також використовуються для серіалізації (перетворення в об’єкт JSON) та десеріалізації (перетворення з JSON в об’єкт) даних при обміні даними між клієнтом та сервером через WebSocket або HTTP;
– інтеграція з іншими компонентами системи: моделі є основою для взаємодії з іншими компонентами системи, такими як контролери, сервіси та інші, що забезпечують виконання бізнес-логіки та обробку запитів.

Реалізація моделей в онлайн консультанті відіграє важливу роль у забезпеченні правильної та ефективної роботи системи, а також в її майбутньому розширенні та підтримці.

На рисунку 3.1 зображена модель Message, у якій використовується для стандартизації формату даних при обміні повідомленнями між сервером та клієнтами через вебсокети, що спрощує обробку та передачу цих даних у режимі реального часу.

```
[Key]
[Column("id")]
0 references
public int Id { get; set; }

[Column("text")]
[Required]
0 references
public string Text { get; set; } = null!;

[Column("chat_id")]
[Required]
1 reference
public int ChatId { get; set; }

1 reference
public Chat Chat { get; set; } = null!;

[Column("user_id")]
[Required]
1 reference
public int UserId { get; set; }

2 references
public User User { get; set; } = null!;
```

Рисунок 3.1 – Вигляд моделей в WebSocket

3.3 Контролери та їх реалізація

Контролери в архітектурі вебдодатків відповідають за обробку та управління запитами від користувачів. Вони відтворюють ключову роль у взаємодії між клієнтами і сервером, забезпечуючи обробку запитів і відповідних відповідей. У контексті вебдодатків, що використовують WebSocket, контролери також відповідають за обробку повідомлень, що надходять через цей протокол.

Реалізація контролерів зазвичай включає такі аспекти:

– обробка маршрутів: контролери визначають, які URL-адреси (маршрути) будуть оброблятися і які дії потрібно виконати при отриманні запиту на кожен конкретний маршрут;
– обробка запитів: контролери містять методи, які виконують необхідні операції при отриманні запитів від клієнтів – це може включати отримання даних від клієнта, валідацію цих даних, виконання бізнес-логіки та підготовку відповіді;
– взаємодія з сервісами: контролери можуть взаємодіяти з різними сервісами (наприклад, сервісами доступу до даних або бізнес-логіки), щоб отримати необхідні дані або виконати операції, які потрібні для обробки запиту;
– формування відповіді: після обробки запиту контролер формує відповідь, яку потім передає назад клієнту – це може бути HTML-сторінка, JSON-об’єкт або інші дані, залежно від типу запиту та призначення додатка;
– керування станом з’єднання: у випадку використання WebSocket контролери можуть відповідати за керування станом з’єднання, управління сесіями та взаємодію з клієнтами в реальному часі.

Основні цілі використання контролерів вебсокетів включають:

– встановлення з’єднання: контролер може містити методи, які обробляють вхідні запити на встановлення вебсокет-з’єднання (після успішного встановлення з’єднання можуть виконуватись додаткові дії,

такі як аутентифікація або ініціалізація сеансу);

– обробка повідомлень: контролери дозволяють обробляти вхідні повідомлення, отримані через вебсокет-з'єднання – це може включати аналіз та перетворення даних, валідацію, збереження в базі даних, сповіщення інших клієнтів тощо;
– відправлення відповідей: контролери надають можливість відправляти відповіді назад до клієнтів через вебсокет-з'єднання.

Реалізація контролерів включає в себе використання конкретних технологій і фреймворків, таких як ASP.NET Core у випадку C#/.NET стеку, що надає інструменти для швидкої та ефективної розробки вебдодатків з WebSocket підтримкою [8].

У додатку А наведено приклад контролеру UserController, який відповідає за обробку HTTP-запитів, пов'язаних з операціями користувачів у системі. Для взаємодії з базою даних він використовує залежності ApplicationDbContext і IMapper, які передаються через конструктор. Логіка запитів включає отримання даних з бази даних через ApplicationDbContext та мапування об'єктів між різними типами даних за допомогою IMapper. Додатково, у контролері реалізована операція реєстрації користувача у чаті.

3.4 Міграції та їх реалізація

У контексті WebSocket міграції означають автоматизоване керування структурою бази даних під час розгортання або оновлення додатку, що використовує WebSocket для обміну даними в режимі реального часу. Наведемо детальніші відомості про це.

Застосування міграцій до WebSocket додатків: WebSocket додатки можуть використовувати міграції для створення або оновлення структури бази даних. Це особливо корисно для додатків, які зберігають історію чатів або потокові дані, які вимагають постійної синхронізації структури даних.

Впровадження нових функцій через міграції: міграції дозволяють впроваджувати нові функції, які потребують змін у структурі бази даних, наприклад, додавання нових таблиць для зберігання додаткових метаданих чи налаштувань, що впливають на роботу WebSocket.

Автоматичне оновлення структури даних: при використанні WebSocket міграції система автоматично перевіряє поточний стан бази даних і застосовує потрібні зміни, якщо вони ще не були застосовані. Це дозволяє легко розгортати оновлення додатку, не переймаючись за цілісність даних.

Приклади сценаріїв міграцій у WebSocket додатках:

– додавання нових полів у таблиці повідомлень для зберігання метаданих про час доставки;
– створення нових таблиць для зберігання історії сесій WebSocket з метою аналізу і відновлення з’єднань;
– оновлення структури чатів для включення нових функціональних можливостей, таких як зображення чи відео у чатах.

Система міграцій у контексті розробки WebSocket: розробники можуть використовувати інструменти, які підтримують міграції для конкретної бази даних (наприклад, Entity Framework Core для реляційних баз даних), а також писати власний код міграцій для нереляційних баз даних або інших типів даних, що підтримуються WebSocket додатками.

Переваги використання міграцій:

– забезпечення структурної цілісності бази даних у процесі розгортання і оновлення;
– автоматизація процесу збереження і синхронізації структури даних;
– зменшення ризику помилок і конфліктів при оновленні бази даних.

Використання міграцій у WebSocket додатках дозволяє зосередитися на розробці функціональності, не витрачаючи час на ручне керування структурою бази даних, що є критичним аспектом для сучасних розподілених додатків з реальним часом обміну даними [7].

У додатку Б наведено приклад міграцій, що використовуються для

забезпечення структурної цілісності бази даних та автоматизації оновлення. При запуску додатку ця система перевіряє поточний стан бази даних і автоматично застосовує необхідні зміни, що дозволяє легко розгортати додаток на нових середовищах або виконувати його оновлення без необхідності ручного втручання в базу даних.

3.5 Сигнал P

Сигнал P в контексті вебсокетів може мати різні значення залежно від контексту, в якому він використовується. Зазвичай сигнал P означає передачу повідомлення або сигналу через вебсокетне з’єднання.

Розглянемо деякі можливі інтерпретації сигналу P.

Протокол: у деяких системах сигнал P може використовуватись як частина протоколу взаємодії між клієнтом і сервером через вебсокети. Наприклад, під час використання WebSocket API для реалізації чату або системи сповіщень, сигнал P може використовуватись для відправлення повідомлення про підписку або отримання нового повідомлення.

Повідомлення: у контексті обміну повідомленнями через вебсокети, сигнал P може позначати саме цей процес – передачу чи отримання повідомлення від одного з’єднання до іншого. Наприклад, клієнт може ініціювати відправлення повідомлення з певним сигналом, який сервер потім обробляє і відправляє відповідь чи обробляє подію відповідно до отриманого сигналу P.

Обробка подій: в деяких системах сигнал P може використовуватись для позначення особливих подій або команд, які клієнт або сервер можуть відправляти один одному через вебсокети. Це може бути, наприклад, сигнал для підписки на певні канали чи теми, сигнал для виконання певної операції на сервері або сигнал для оновлення статусу.

Реалізація: у реалізації програмного забезпечення з вебсокетами сигнал P може відповідати конкретному методу чи дії, яку потрібно виконати. Наприклад,

при використанні WebSocket API для управління обміном даними в реальному часі, сигнал P може ідентифікувати метод, що має бути викликаний для обробки отриманого повідомлення або події.

Загалом, сигнал P є контекстно залежним і може використовуватись для різних цілей у системах, що використовують вебсокети, від позначення повідомлень до управління подіями і взаємодією між клієнтом і сервером.

3.6 Тестування онлайн консультанта

У даному підрозділі проводиться функціональне тестування розробленого онлайн консультанта з метою перевірки відповідності функціональним вимогам і очікуванням користувачів. Цей вид тестування дозволяє виявити потенційні проблеми та дефекти у роботі консультанта, а також переконатися, що всі його функції працюють належним чином.

Основною метою функціонального тестування є забезпечення роботи консультанта відповідно до очікувань користувачів, виявлення можливих недоліків та покращення загальної якості продукту.

У даній роботі будуть використані наступні тестові сценарії для досягнення цих цілей.

Реєстрація нового користувача:

– ввести валідні дані для реєстрації нового користувача;
– перевірити, чи успішно створено обліковий запис користувача в системі;
– спробувати зареєструватися з невалідними або вже існуючими даними та перевірити, чи виведено відповідне повідомлення про помилку.

Авторизація користувача:

– ввести валідні дані для авторизації в системі;
– перевірити, чи успішно авторизовано користувача та перенаправлено його на відповідну сторінку;

– спробувати авторизуватися з невалідними або неправильними даними та перевірити, чи виведено відповідне повідомлення про помилку.

Розпочати консультацію:

– увійти в систему як зареєстрований користувач;
– вибрати доступного консультанта для початку сесії консультування;
– перевірити, чи успішно відкрито сесію консультування та відображено інтерфейс для обміну повідомленнями.

Взаємодія з консультантом:

– відправити повідомлення консультанту та перевірити, чи відображено його відповідь;
– спробувати надіслати невалідне повідомлення (наприклад, порожнє або неприпустимої довжини) та перевірити, чи виведено відповідне повідомлення про помилку.

Завершення консультації:

– завершити сесію консультування та перевірити, чи правильно збережені всі повідомлення та додаткові дані сесії;
– переконатися, що після завершення консультації користувач може оцінити якість консультанта.

На рисунках 3.2 – 3.4 реалізовано тестування нашого онлайн консультанта.

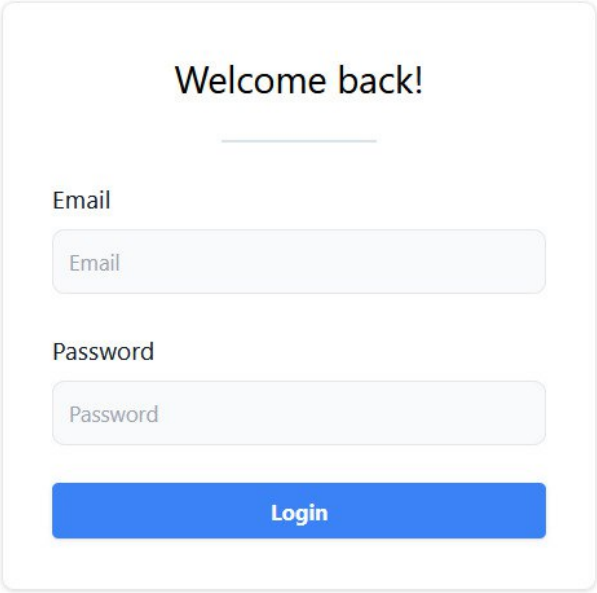

Рисунок 3.2 – Авторизація користувача

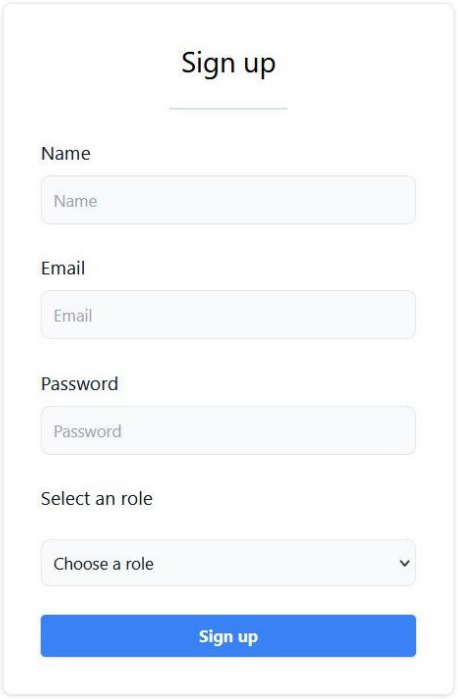

Рисунок 3.3 – Реєстрація користувача

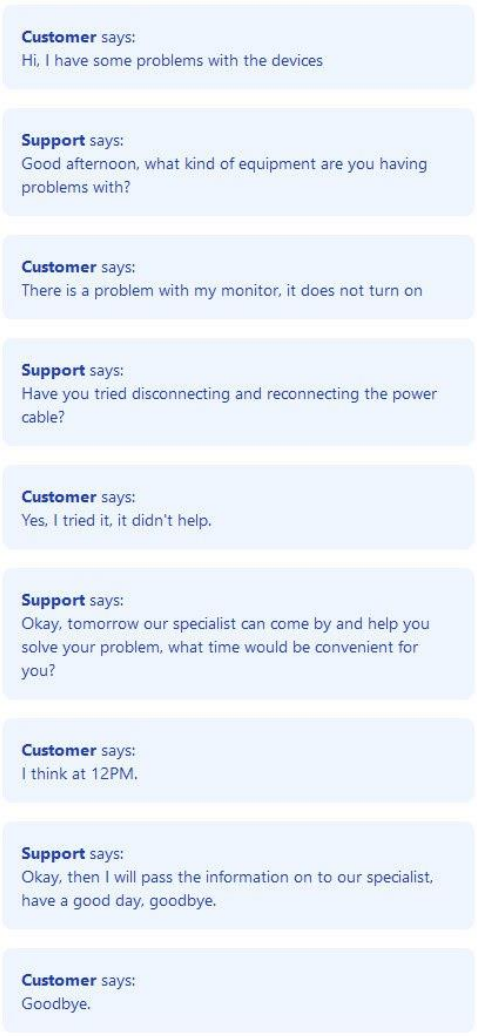

Рисунок 3.4 – Взаємодія користувача з онлайн консультантом

3.7 Висновки до розділу 3

У рамках цього розділу була проведена розробка та тестування онлайн консультанта для вебсайтів з використанням WebSocket. Процес реалізації включав створення серверної та клієнтської частини з використанням технологій, таких як WebSocket, MySQL, React та C#.

Під час розробки були реалізовані основні функції онлайн консультанта, включаючи встановлення з’єднання між клієнтом і сервером за допомогою WebSocket, обмін повідомленнями в реальному часі та збереження даних в базі даних MySQL. Клієнтська частина була розроблена з використанням React, а серверна частина – з використанням C#.

Для забезпечення якості системи були виконані функціональні тести, які дозволили перевірити коректність роботи основних функціональних можливостей консультанта. Були також виконані інтеграційні тести для перевірки взаємодії між компонентами системи. В процесі тестування були виявлені та виправлені деякі помилки та проблеми, що покращило якість системи.

У результаті реалізації та тестування був отриманий функціональний онлайн консультант, який задовольняє поставлені вимоги та забезпечує ефективну комунікацію між клієнтами та консультантами на вебсайтах. Система готова до подальшого розгортання та використання в реальних умовах.

ВИСНОВКИ

На основі виконаного проєкту було реалізовано інноваційний вебзастосунок, спрямований на покращення ефективності обміну повідомленнями між користувачами за допомогою технології WebSocket. Проєкт включав аналіз вимог до програмного забезпечення, формулювання мети і завдань, а також розробку серверної і клієнтської частин застосунку.

На клієнтській стороні було використано JavaScript з використанням бібліотеки React для створення динамічного та інтерактивного інтерфейсу, що забезпечує зручну взаємодію користувачів з повідомленнями в реальному часі. Для стилізації та організації компонентів інтерфейсу було використано Bootstrap, що забезпечило сучасний та естетичний вигляд застосунку.

Серверна частина застосунку була реалізована на базі Node.js з використанням бібліотеки Socket.io для обробки і керування WebSocket з’єднаннями. Це дозволило забезпечити надійне з’єднання між клієнтами та сервером і ефективну обробку повідомлень у реальному часі.

Проєкт пройшов інтенсивне тестування для підтвердження коректності роботи всіх функціональних можливостей та стабільності застосунку при різних умовах використання. У майбутньому планується розвиток застосунку шляхом розширення його функціональності, вдосконалення дизайну та зручності використання на основі отриманого фідбеку від реальних користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft Documentation – WebSocket in .NET. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 03.04.2024).
2. WebSocket Protocol RFC 6455. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 04.04.2024).
3. Документація C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 05.04.2024).
4. Документація MySQL. URL: https://dev.mysql.com/doc/ (дата звернення: 08.04.2024).
5. GitHub – ASP.NET Core WebSocket Sample. URL: https://github.com/dotnet/aspnetcore/tree/main/src/Samples/WebSockets (дата звернення: 08.04.2024).
6. Документація ASP.NET. URL: https://jakeydocs.readthedocs.io/en/latest/ (дата звернення: 10.04.2024).
7. WebSockets: навіщо потрібні та як з ними працювати. URL: http://surl.li/upkia (дата звернення: 11.04.2024).
8. YouTube – Implementing WebSocket in C#. URL:https://www.youtube.com/watch?v=P32x1Xb7BKE (дата звернення:11.04.2024).
9. Переваги і недоліки Вебсокетів. URL: https://what.com.ua/javascript-websocket-opis-pr/ (дата звернення: 14.04.2024).
10. Переваги і недоліки C#. URL: https://dou.ua/forums/topic/42888/ (дата звернення: 21.04.2024).
11. CodeProject Tutorial – WebSocket in C#. URL: https://is.gd/eqPJba (дата звернення: 21.04.2024).
12. WebSocket Client Library – WebSocket4Net. URL: https://github.com/kerryjiang/WebSocket4Net (дата звернення: 21.04.2024).

ДОДАТОК А

Приклад контролеру UserController

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Runtime.InteropServices;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

namespace YourNamespace.Controllers
{
  [ApiController]
  [Route("api/[controller]")]
  public class UserController : ControllerBase
  {
    private readonly ApplicationDbContext _context;
    private readonly IMapper _mapper;

    public UserController(ApplicationDbContext context, IMapper mapper)
    {
      _context = context;
      _mapper = mapper;
    }

    [HttpPost("register")]
    public async Task<ActionResult<UserWithTokenDto>> Register(UserRegisterDto user)
    {
      if (!ModelState.IsValid)
      {
        return BadRequest(ModelState);
      }

      user.Password = BCrypt.Net.BCrypt.HashPassword(user.Password);
      var userResult = _context.Users.Add(_mapper.Map<User>(user));
      await _context.SaveChangesAsync();

      var secretKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Environment.GetEnvironmentVariable("JWT_SECRET_KEY")!));
      var signinCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);

      var userData = await _context.Users
        .Include(u => u.Role)
        .FirstOrDefaultAsync(u => u.Id == userResult.Entity.Id);

      JwtSecurityToken tokenOptions = new JwtSecurityToken(
        issuer: Environment.GetEnvironmentVariable("APP_URL"),
        audience: Environment.GetEnvironmentVariable("APP_URL"),
        claims: new List<Claim>
        {
```

```
                new Claim("Id", userData.Id.ToString()),
                new Claim("Email", userData.Email),
            },
            expires: DateTime.Now.AddDays(30),
            signingCredentials: signinCredentials
        );

        string token = new JwtSecurityTokenHandler().WriteToken(tokenOptions);
        var userWithToken = _mapper.Map<UserWithTokenDto>(userData);
        userWithToken.Token = token;

        return userWithToken;
    }
  }
}
```

ДОДАТОК Б

Приклад міграцій

```
public partial class InitialDataTables : Migration
{
   protected override void Up(MigrationBuilder migrationBuilder)
   {
      migrationBuilder.CreateTable(
         name: "Roles",
         columns: table => new
         {
            Id = table.Column<int>(type: "int", nullable: false)
               .Annotation("MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn),
            Name = table.Column<string>(type: "longtext", nullable: false)
               .Annotation("MySql:CharSet", "utf8mb4")
         },
         constraints: table =>
         {
            table.PrimaryKey("PK_Roles", x => x.Id);
         })
         .Annotation("MySql:CharSet", "utf8mb4");

      migrationBuilder.CreateTable(
         name: "Users",
         columns: table => new
         {
            Id = table.Column<int>(type: "int", nullable: false)
               .Annotation("MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn),
            Name = table.Column<string>(type: "longtext", nullable: false)
               .Annotation("MySql:CharSet", "utf8mb4"),
            Email = table.Column<string>(type: "longtext", nullable: false)
               .Annotation("MySql:CharSet", "utf8mb4"),
            Password = table.Column<string>(type: "longtext", nullable: false)
               .Annotation("MySql:CharSet", "utf8mb4"),
            RoleId = table.Column<int>(type: "int", nullable: false)
         },
         constraints: table =>
         {
            table.PrimaryKey("PK_Users", x => x.Id);
            table.ForeignKey(
               name: "FK_Users_Roles_RoleId",
               column: x => x.RoleId,
               principalTable: "Roles",
               principalColumn: "Id",
               onDelete: ReferentialAction.Cascade);
         })
         .Annotation("MySql:CharSet", "utf8mb4");

      migrationBuilder.CreateTable(
         name: "Chats",
         columns: table => new
         {
            Id = table.Column<int>(type: "int", nullable: false)
               .Annotation("MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn),
```

```
            CustomerId = table.Column<int>(type: "int", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Chats", x => x.Id);
            table.ForeignKey(
                name: "FK_Chats_Users_CustomerId",
                column: x => x.CustomerId,
                principalTable: "Users",
                principalColumn: "Id",
                onDelete: ReferentialAction.Restrict);
        })
        .Annotation("MySql:CharSet", "utf8mb4");

    migrationBuilder.CreateTable(
        name: "Messages",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn),
            Text = table.Column<string>(type: "longtext", nullable: false)
                .Annotation("MySql:CharSet", "utf8mb4"),
            ChatId = table.Column<int>(type: "int", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Messages", x => x.Id);
            table.ForeignKey(
                name: "FK_Messages_Chats_ChatId",
                column: x => x.ChatId,
                principalTable: "Chats",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        })
        .Annotation("MySql:CharSet", "utf8mb4");

    migrationBuilder.CreateIndex(
        name: "IX_Chats_CustomerId",
        table: "Chats",
        column: "CustomerId");

    migrationBuilder.CreateIndex(
        name: "IX_Messages_ChatId",
        table: "Messages",
        column: "ChatId");

    migrationBuilder.CreateIndex(
        name: "IX_Users_RoleId",
        table: "Users",
        column: "RoleId");
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Messages");

    migrationBuilder.DropTable(
        name: "Chats");
```

```
        migrationBuilder.DropTable(
            name: "Users");

        migrationBuilder.DropTable(
            name: "Roles");
    }
}
```