**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН КОНСУЛЬТАНТА ДЛЯ ВЕБСАЙТІВ З ВИКОРИСТАННЯМ WEBSOCKET»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.В. Тимошенко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD, Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

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

Запоріжжя – 2023

**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)

“ 07 ” 02 2023 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

**Тимошенку Данилу Вячеславовичу**
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн консультанта для вебсайтів з використанням WebSocket

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 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 докладу

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.02.2023 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. | 08.02.2023 | |
| | | | |
| 2. | Збір вихідних даних. | 20.02.2023 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 13.03.2023 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 17.04.2023 | |
| | | | |
| 5. | Розробка третього розділу. | 15.05.2023 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 01.06.2023 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 23.06.2023 | |

Студент _____ Д.В. Тимошенко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

**Нормоконтроль пройдено**

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

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн консультанта для вебсайтів з використанням WebSocket»: 45 с., 13 рис., 8 джерел, 1 додаток.

КОРИСТУВАЧ, ЧАТ, ASPNET, C#, MYSQL, REACT, WEBSOCKET.

Об’єкт дослідження – розробка онлайн консультанта для вебсайтів з використанням WebSocket.

Мета роботи: розробити функціонального та практичного консультанта вебсайтів з використанням WebSocket, для вирішення проблем користувачів.

Метод дослідження – методи програмної інженерії.

Результати та їх новизна: був розроблений онлайн чат для більш тонкого контакту користувача з онлайн консультантом, для подальшого вирішення питань користувача.

Взаємозв’язок з іншими роботами: дана робота базується на дослідженні WebSocket та взаємодії їх зв’язків для передачі даних між браузерами та вебсерверами, використовуючи при цьому цілодобове з’єднання.

Рекомендації щодо використання результатів роботи: розроблений онлайн консультант може бути використаний в різних сферах підприємств для вирішення проблем клієнтів.

Значимість роботи та висновки: розробка онлайн консультанта є дуже актуальною так як здебільшого він може вирішити проблеми легкого характеру та дати пораду клієнту/користувачу. Не потребує довгого очікування відповіді на задану проблему. Отже, розробка онлайн консультанта для вебсайтів з використанням WebSocket дуже економить час клієнта та допомогає вирішити проблеми легкого характеру, або переадресувати проблему все до рук спеціаліста.

# SUMMARY

Master’s qualification work «Development of a Online Consultant for Website using WebSocket»: 45 p., 13 figures, 8 references, 1 supplement.

USER, CHAT, ASPNET, C#, MYSQL, REACT, WEBSOCKET.

Object of research is development of an online consultant for websites using WebSocket.

Purpose of the study is to develop a functional and practical website consultant using WebSocket to solve user problems.

Research method is software engineering methods.

Results and novelty: an online chat has been developed for more subtle contact between the user and the online consultant, for further solving user issues.

Relationship to other works: this work is based on the study of WebSocket and the interaction of their connections to transfer data between browsers and web servers, while using a round-the-clock connection.

Recommendations for using the results of the work: the developed online consultant can be used in various areas of enterprises to solve customer problems.

Significance of the work and conclusions: the development of an online consultant is very relevant because it can mostly solve problems of a light nature and give advice to the client/user. It does not require a long wait for an answer to a given problem. Thus, the development of an online consultant for websites using WebSocket saves the client’s time and helps to solve light problems or to redirect the problem to a specialist.

**ЗМІСТ**

# ВСТУП

Розвиток технологій в кардинальний спосіб змінює різні аспекти нашого життя, включаючи способи комунікації та надання послуг в Інтернеті. У сучасній цифровій епохи важливо, щоб вебсайти мали ефективні та зручні інструменти для надання онлайн консультацій. Цей проєкт присвячений розробці онлайн консультанта для вебсайтів з використанням потужного протоколу WebSocket та сучасних технологій.

У сучасному цифровому світі, вебсайти є важливим інструментом для забезпечення ефективної комунікації з клієнтами та відвідувачами. Компаніям потрібні рішення, які надають можливість реального часу взаємодії з клієнтами, відповідаючи на їх запитання та надаючи необхідну допомогу. Вебсайт, що використовує консультанта на основі WebSocket, може надати цю можливість з миттєвим оновленням вмісту та миттєвою передачею повідомлень.

Метою цього проєкту є розробка онлайн консультанта для вебсайтів, який використовує протокол WebSocket для забезпечення швидкої та плавної взаємодії з відвідувачами. WebSocket – це протокол двостороннього зв’язку, який дозволяє вебсайтам встановлювати постійне з’єднання з сервером та обмінюватися даними в режимі реального часу.

У рамках цього проєкту будуть розроблені необхідні компоненти та інтерфейс для онлайн консультанта. За допомогою технологій, таких як WebSocket, буде забезпечена швидка передача повідомлень та оновлення контенту без необхідності перезавантаження сторінки. Крім того, будуть розглянуті можливості зберігання даних та інтеграції з базами даних для забезпечення зручного керування консультаціями та клієнтськими запитами.

Цей проєкт має велике значення, оскільки він дозволяє покращити якість обслуговування клієнтів та відвідувачів вебсайту, забезпечуючи швидку та ефективну комунікацію. Використання WebSocket та сучасних технологій дозволяє створити інтерактивне середовище, яке задовольняє потреби користувачів та сприяє покращенню їх задоволення від взаємодії з вебсайтом.

# 1 ТЕОРЕТИЧНИЙ ОГЛЯД ТА ВИВЧЕННЯ WEBSOCKET

## 1.1 Огляд основних понять WebSocket, та їх використання

WebSocket – це протокол зв’язку між вебпрограмою (клієнтом) та вебсервером, що дозволяє підтримувати постійне та багатониткове з’єднання між ними. Це означає, що після встановлення з’єднання, клієнт і сервер можуть обмінюватися даними в реальному часі без необхідності повторного встановлення з’єднання при кожному запиті. WebSocket використовується для створення інтерактивних вебдодатків з можливістю миттєвого оновлення даних для користувачів.

Основними особливостями WebSocket є:

– багатонитковість: WebSocket може передавати дані в режимі реального часу в обох напрямках (це означає, що як клієнт, так і сервер можуть надсилати повідомлення один одному в будь-який момент без затримок);
– простота використання: WebSocket працює на рівні протоколу, що робить його легким у реалізації та використанні (розробники можуть легко підключити WebSocket до своїх вебдодатків, не потребуючи складних налаштувань чи залежностей);
– постійне з’єднання: WebSocket дозволяє підтримувати активне з’єднання між клієнтом і сервером без переривання під час обміну даними (це відрізняє його від протоколу HTTP, де кожен запит вимагає нове з’єднання).

WebSocket може використовуватися в багатьох сценаріях, включаючи:

– інтерактивні додатки: WebSocket дозволяє створювати інтерактивні додатки, такі як чати або спільна робота над документами (користувачі можуть бачити оновлення в режимі реального часу без необхідності оновлення сторінки);

– системи сповіщень: WebSocket дозволяє швидко та ефективно надсилати сповіщення користувачам (це можуть бути повідомлення про нові повідомлення, події або оновлення в системі);
– онлайн-ігри: WebSocket використовується в онлайн-іграх для передачі даних про дії гравців та оновлення гри (він забезпечує швидкий та ефективний обмін даними між гравцями в реальному часі).

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. 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.

Також WebSocket використовує спеціальний протокол, що базується на TCP (Transmission Control Protocol). Протокол WebSocket має власні правила і формати повідомлень, які використовуються для комунікації між клієнтом і сервером. Використання протоколу WebSocket дозволяє передавати повідомлення в реальному часі без зайвих накладних витрат на обробку запитів [1].

## 1.2 Можливості WebSocket, переваги та недоліки

Звичайні вебпротоколи, такі як HTTP, працюють за моделлю «клієнт-сервер», де клієнтський браузер виконує запит до сервера, а сервер надсилає відповідь. Цей процес вимагає постійного запиту-відповіді для отримання нових даних, що не є ефективним для багатьох сценаріїв, особливо тих, які вимагають миттєвої актуалізації даних або взаємодії в реальному часі.

Однак WebSocket змінює цю модель і дозволяє більш динамічну та ефективну взаємодію між клієнтом і сервером. Він надає наступні можливості, які наведено нижче.

*Постійне з'єднання*: WebSocket забезпечує постійне з'єднання між клієнтом і сервером, що дозволяє надсилати та отримувати дані в режимі реального часу без необхідності повторного встановлення з'єднання при кожному запиті. Це зменшує накладні витрати на з'єднання та забезпечує швидку взаємодію.

*Багатонитковість*: WebSocket дозволяє обмінюватися даними між клієнтом і сервером одночасно в режимі багатонитковості. Це означає, що сервер може ініціювати відправку повідомлень клієнту в будь-який час, а клієнт також може надсилати дані на сервер у будь-який момент. Це дозволяє створювати динамічні та інтерактивні додатки, які можуть відображати оновлення в реальному часі.

*Ефективність*: WebSocket має низьку накладну витрату через збереження постійного з'єднання. Він використовує оптимізований механізм обміну даними, що дозволяє ефективно передавати повідомлення без непотрібних протоколів, таких як HTTP-заголовки, які необхідні для кожного запиту-відповіді.

*Розширені можливості передачі даних*: WebSocket дозволяє передавати різні типи даних, включаючи текстові повідомлення, бінарні дані та навіть стріми даних. Це розширює можливості вебдодатків, дозволяючи передавати складні дані та медіафайли безпосередньо через з'єднання WebSocket.

*Низька затримка*: WebSocket дозволяє знизити затримку між відправленням та отриманням даних, оскільки немає необхідності у постійному встановленні нових з'єднань. Це особливо важливо для додатків, які вимагають швидкого обміну даними в реальному часі, таких як фінансові системи, чати або онлайн-ігри.

*Сумісність*: WebSocket працює на багатьох платформах і браузерах, що забезпечує його широку сумісність та доступність для розробників. Він

підтримується більшістю сучасних браузерів та вебсерверів, що робить його ідеальним вибором для розробки різноманітних вебдодатків.

Ці можливості WebSocket роблять його потужним інструментом для розробки вебдодатків, які потребують миттєвої взаємодії, оновлення даних в реальному часі та швидкого обміну інформацією між клієнтом і сервером. Проте, варто враховувати певні переваги і недоліки WebSocket.

Переваги використання WebSocket [7]:

– ефективність: WebSocket використовує менше ресурсів сервера та мережі порівняно з традиційними методами зв’язку, такими як запити HTTP (це забезпечує більш швидку та ефективну передачу даних);
– реальний час: WebSocket дозволяє миттєве оновлення даних без необхідності повторного встановлення з’єднання при кожному запиті (це особливо важливо для додатків, які потребують реалізації функцій у режимі реального часу, таких як чати або сповіщення);
– надійність: WebSocket автоматично відновлює з’єднання при втраті зв’язку або зміні мережевих параметрів (це забезпечує надійну комунікацію між клієнтом і сервером навіть при нестабільному з’єднанні).

Недоліки використання WebSocket:

– підтримка браузерами: деякі старі версії браузерів можуть не підтримувати WebSocket, що може створювати проблеми для сумісності із застарілими браузерами;
– загальнодоступність сервера: WebSocket вимагає наявності серверної підтримки для обробки з’єднань та обміну даними (це може вимагати – додаткового налаштування та управління серверною інфраструктурою);
– збільшення навантаження: постійне з’єднання WebSocket може збільшити навантаження на сервер, оскільки з’єднання залишається відкритим протягом тривалого часу (це може вплинути на масштабованість та продуктивність сервера при великій кількості одночасних з’єднань).

## 1.3 Висновки до розділу 1

У даному розділі ми детально розглянули WebSocket як ключовий технологічний компонент для розробки нашого онлайн-консультанта. WebSocket дозволяють встановлювати постійне двостороннє з’єднання між клієнтом і сервером, що відкриває широкі можливості для реалізації реального часу, спілкування та оновлення даних без необхідності постійного перезавантаження сторінки.

За допомогою WebSocket ми зможемо створити інтерактивний та динамічний інтерфейс для нашого онлайн-консультанта. Користувачі зможуть спілкуватися з консультантом в режимі реального часу, отримувати миттєві відповіді та отримувати актуальну інформацію без затримок.

Використання WebSocket дозволить нам забезпечити швидку та ефективну взаємодію з користувачами, забезпечити зручний та надійний сервіс консультування в режимі реального часу.

# 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Стек технологій

C# (C-Sharp) – це мова програмування, розроблена компанією Microsoft, яка широко використовується для розробки різноманітних програмних додатків. C# є частиною платформи .NET і забезпечує потужність, гнучкість та широкий функціонал для створення високоякісного програмного забезпечення.

Використання мови C# у нашому онлайн консультанті має численні переваги. Об’єктно-орієнтований підхід: C# підтримує концепцію об’єктно-орієнтованого програмування (ООП), що дозволяє організувати код у вигляді класів, об’єктів, спадкування, поліморфізму та інших ООП-принципів. Це спрощує розробку, підтримку та розширення коду.

Завдяки платформі .NET, C# може легко взаємодіяти з іншими мовами програмування, такими як VB.NET, F#, C++ і т.д. Це дозволяє нам використовувати найкращі можливості різних мов для реалізації складних функцій у нашому консультанті.

Також він має вбудовану підтримку для роботи з протоколом WebSocket. WebSocket дозволяє забезпечити двосторонню зв’язок між клієнтом і сервером у режимі реального часу. Ми можемо використовувати цей протокол для побудови інтерактивного середовища консультанта, де користувачі можуть [3].

Реакт (React) – це популярний фреймворк для розробки користувацьких інтерфейсів у мові JavaScript. Він надає зручні інструменти та структуру для створення інтерактивних та масштабованих вебдодатків. Реакт базується на концепції компонентів, які можуть бути повторно використані та забезпечують ефективне керування станом додатка. Також він має високу продуктивність завдяки використанню віртуального DOM (Document Object Model) та розумного оновлення компонентів [8].

У нашому онлайн консультанті ми можемо використовувати Реакт для

створення інтерфейсу, з яким користувачі будуть взаємодіяти. Основна перевага використання Реакт полягає в його компонентній архітектурі. Ми можемо розбити інтерфейс на окремі компоненти, які будуть відповідати за певні функціональні частини. Наприклад, ми можемо мати компоненти для відображення повідомлень, форму для введення даних, кнопки тощо. Кожен компонент буде мати свій внутрішній стан та методи для його зміни.

Компоненти Реакт також можуть бути повторно використані. Це означає, що ми можемо створити універсальні компоненти, які можна використовувати на різних сторінках нашого онлайн консультанта. Наприклад, компонент для відображення повідомлень може бути використаний як на сторінці чату, так і на сторінці зворотного зв’язку [6].

ASP.NET – це фреймворк розробки вебдодатків, розроблений компанією Microsoft. Він надає потужні інструменти та можливості для створення високопродуктивних вебдодатків з використанням мови програмування C#. Ось основні особливості та способи використання ASP.NET для онлайн консультанта.

ASP.NET пропонує архітектурний шаблон Model-View-Controller (MVC), який дозволяє організувати додаток у вигляді окремих компонентів, що взаємодіють між собою. Модель представляє дані та бізнес-логіку, Представлення відповідає за відображення інтерфейсу користувача, а Контролер обробляє запити та керує потоком даних між моделлю та представленням. З використанням шаблону MVC ми зможемо ефективно розділити логіку, відображення та дані, що дозволить зберігати наш код організованим і легким для розуміння та підтримки. Ідеально підходить для використання з вебсервером Internet Information Services (IIS), що є стандартним сервером вебдодатків для операційних систем Windows. IIS надає потужну та безпечну платформу для розгортання та виконання вебдодатків, побудованих на основі ASP.NET. За допомогою IIS ми зможемо легко налаштувати та керувати нашим онлайн консультантом на сервері.

Також він надає велику кількість готових бібліотек класів, які спрощують

розробку вебдодатків. Наприклад, бібліотека класів ASP.NET MVC надає зручні інструменти для створення контролерів, відображень, маршрутизації та керування даними. Ми можемо використовувати ці бібліотеки для швидкого розгортання базового функціоналу нашого онлайн консультанта та заощадження часу розробки [2].

ASP.NET має вбудовані можливості для забезпечення безпеки та аутентифікації вебдодатків. Ми можемо використовувати вбудовані механізми аутентифікації та авторизації, такі як ролі, клейми та токени, для контролю доступу до функцій нашого онлайн консультанта. ASP.NET також надає можливості для захисту даних, включаючи шифрування з’єднання та захист від атак хакерів.

Загалом, використання фреймворка ASP.NET в нашому онлайн консультанті дозволить нам швидко створити потужний та безпечний вебдодаток з допомогою мови програмування C# та шаблону розробки MVC. Ми зможемо розділити логіку, відображення та дані, використовуючи готові бібліотеки класів та забезпечити безпеку та аутентифікацію нашого консультанта [4].

## 2.2 Постанова задачі

У сучасному світі онлайн-комунікація набуває все більшої популярності, а бізнес-вебсайти стають місцем, де користувачі шукають інформацію та взаємодіють з компаніями. У такому контексті велике значення має наявність онлайн консультанта, який надає реальний часовий супровід та відповідає на запитання користувачів негайно.

Однак, традиційні методи реалізації онлайн-консультанта мають свої обмеження. Зазвичай вони використовують AJAX або HTTP-запити для взаємодії між клієнтом та сервером. Це означає, що клієнтський браузер постійно відправляє запити на сервер, щоб отримати оновлену інформацію.

Такий підхід не є ефективним, оскільки він призводить до зайвого трафіку та затримок в оновленні даних.

Для розв’язання цих проблем ми пропонуємо використати WebSocket – технологію, яка надає можливість встановлення постійного двостороннього з’єднання між клієнтом і сервером. WebSocket дозволяє передавати дані в реальному часі без необхідності постійного оновлення сторінки.

Наша задача полягає в розробці онлайн консультанта для вебсайтів з використанням WebSocket. Ми плануємо створити інтерактивний інтерфейс, який дозволить користувачам взаємодіяти з консультантом в режимі реального часу.

Розробити серверну частину, яка буде відповідати за обробку запитів вебсокетів. Ми використаємо популярний фреймворк або бібліотеку для реалізації серверного коду.

Створити клієнтську частину, яка буде відповідати за відображення інтерфейсу консультанта на вебсайті. Ми розробимо зручний та привабливий дизайн, що забезпечить зручну взаємодію з користувачами.

Реалізувати можливість відправки повідомлень в режимі реального часу між консультантом та користувачем. Ми забезпечимо швидку та надійну передачу даних через вебсокети, що дозволить уникнути затримок та забезпечить миттєву відповідь.

Забезпечити можливість аутентифікації та авторизації користувачів, щоб забезпечити конфіденційність та безпеку обміну інформацією між консультантом та клієнтами.

Надати можливість зберігати історію повідомлень та надавати функціональність пошуку та фільтрації для полегшення навігації та аналізу даних. Забезпечити масштабованість та надійність системи, щоб вона могла працювати з великою кількістю одночасних підключень та забезпечувати стабільну роботу навіть при високому навантаженні.

В результаті розробки онлайн консультанта з використанням WebSocket ми плануємо створити потужний та ефективний інструмент, який забезпечить

зручну комунікацію між консультантом та користувачами. Такий рішення дозволить покращити якість.

### 2.3 Створення ER-діаграми та діаграм прецедентів

ER-діаграма (Entity-Relationship diagram) є графічним інструментом, який використовується для моделювання структури даних в базах даних. Вона дозволяє візуалізувати зв’язки між різними сутностями (або таблицями) в базі даних та описати їх атрибути.

Далі ми можемо спостерігати нашу ER-діаграми на рисунку 2.1.

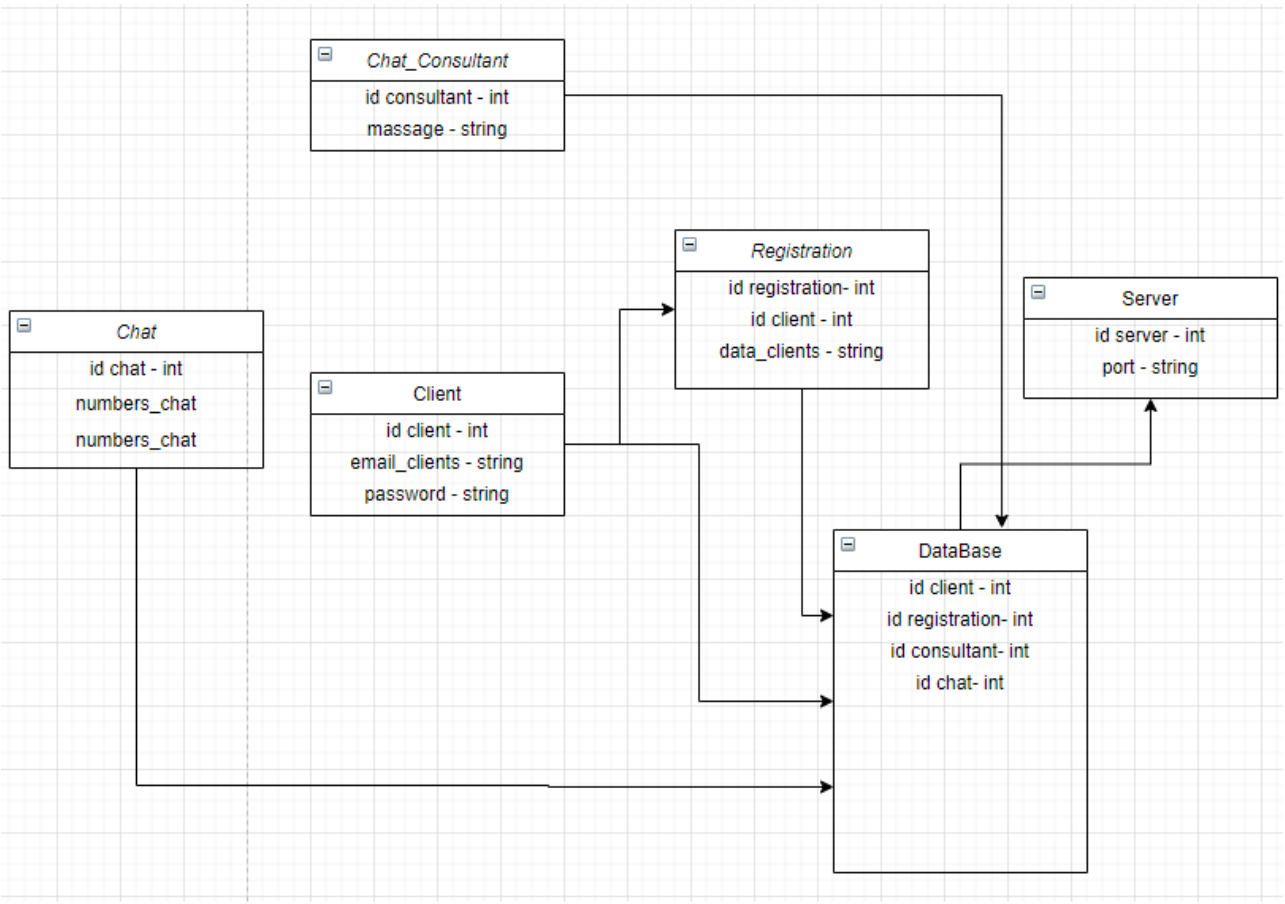


Рисунок 2.1 – ER-діаграма бази даних

Основними компонентами ER-діаграми є сутності (таблиці), зв’язки між ними та атрибути, які характеризують кожну сутність. Сутності відображаються

у вигляді прямокутників, а зв’язки – у вигляді ліній, які з’єднують сутності.

Основна мета ER-діаграми – встановити зв’язки між сутностями і описати їх характеристики. Завдяки цьому, ми можемо розробити оптимальну структуру бази даних, визначити типи зв’язків (один-до-одного, один-до-багатьох, багато-до-багатьох) та визначити обмеження цих зв’язків (наприклад, обов’язковість або унікальність) [5].

Діаграма прецедентів – це одна з ключових діаграм, використовуваних при проєктуванні програмного забезпечення. Вона зображує взаємодії між системою (або програмою) та її користувачами, які називаються акторами.

Діаграма прецедентів складається з акторів, прецедентів та взаємодій між ними.

На рисунку 2.2 ми можемо спостерігати нашу діаграму прецедентів.

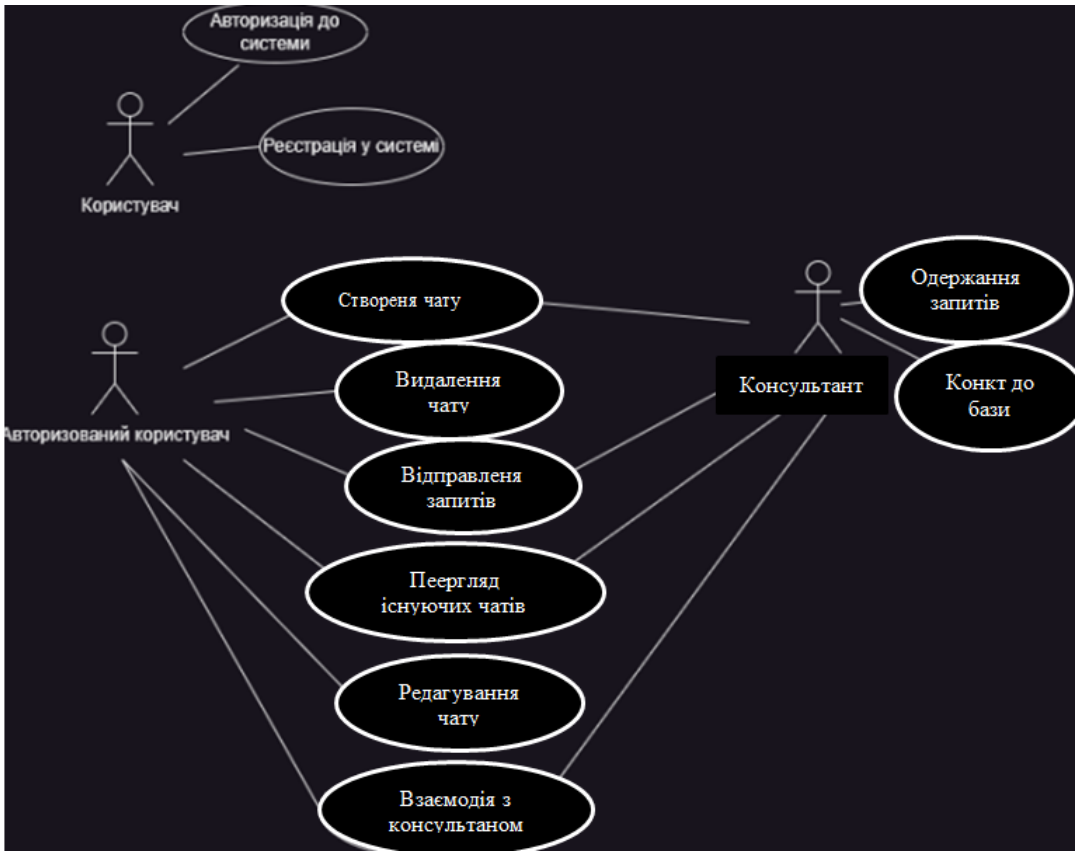


Рисунок 2.2 – Діаграма прецедентів

Актори – це користувачі системи або інші системи, які взаємодіють з нею.

Прецеденти – це опис дії, яку може виконати система або її частина.

Взаємодії – це способи, за якими актор взаємодіє з системою.

Діаграма прецедентів може бути корисна при проєктуванні програмного забезпечення, оскільки вона допомагає зрозуміти вимоги користувачів та функціональність системи. Вона також може використовуватися для комунікації між розробниками та клієнтами, оскільки вона надає чітке візуальне представлення того, як буде використовуватися програмне забезпечення.

## 2.4 Висновки до розділу 2

В результаті аналізу та проєктування системи онлайн консультанта було розроблено детальну архітектуру та компоненти, необхідні для її реалізації. Були визначені взаємозв’язки між цими компонентами та способи взаємодії між ними.

Проєктування системи онлайн консультанта допомагає забезпечити ефективне та структуроване виконання розробки, зменшує ризики та забезпечує високу якість результуючої системи. Цей процес визначає основні принципи, архітектурні рішення та структуру, які використовуються для побудови системи онлайн консультанта.

Результатом проєктування системи онлайн консультанта є докладний план, що включає 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 [6].

В цьому розділі були представлені результати проєктування системи онлайн консультанта, що будуть використані для подальшої розробки та реалізації системи. Вони включають діаграми архітектури, опис компонентів, взаємодію між ними та інші важливі аспекти, які допоможуть зрозуміти та втілити задуману систему онлайн консультанта.

# 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

В даному розділі дипломної роботи розглядається процес реалізації та тестування системи онлайн консультанта, що є центральним елементом даного проєкту. Після проведення аналізу, проєктування та моделювання системи, настав час перейти до фази реальної розробки, де ідеї перетворюються на живий функціональний продукт.

У цьому розділі детально описано процес розробки програмного забезпечення для системи онлайн консультанта. Він включає вибір оптимальних технологій, написання коду, налаштування робочого середовища та створення архітектури системи. Кожен етап реалізації детально розглядається з орієнтацією на стабільність, ефективність та надійність системи. Також у даному 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 даного розділу є детальний опис реалізації системи онлайн консультанта, включаючи архітектурні рішення, використані технології, принципи розробки та використані інструменти. Крім того, будуть представлені результати проведеного тестування, його висновки та огляд.

## 3.1 Основні аспекти серверної части онлайн консультанта

Серверна частина консультанта є важливою складовою його функціональності і забезпечує взаємодію з клієнтськими пристроями та зберігання даних.

Клієнт-серверна архітектура для обміну даними з клієнтськими

пристроями. Клієнти взаємодіють з сервером, надсилаючи запити та отримуючи відповіді через мережу.

Вебсервер: для реалізації серверної частини був використаний вебсервер, який забезпечує обробку HTTP-запитів. Він приймає запити від клієнтів, обробляє їх і повертає відповіді.

Безпека: найбільш важливий аспект – безпека обміну даними. Вона повинна забезпечувати захист конфіденційної інформації та персональних даних клієнтів. Для цього використовуються шифрування даних, механізми автентифікації та авторизації користувачів, а також застосування відповідних безпекових протоколів.

Обробка запитів,  отриманих від клієнтів, і виконання необхідний дій. Це може включати перевірку валідності запитів, обробку бізнес-логіки, доступ до бази даних та виконання різних операцій.

Використовує базу даних для зберігання і управління інформацією. Вона може зберігати дані про користувачів, історію консультацій, налаштування системи та іншу важливу інформацію. База даних може бути реляційною або нереляційною, залежно від потреб проєкту.

Масштабованість – здатність обробляти багато запитів одночасно та масштабуватись залежно від потреб. Це досягається за допомогою розподіленої архітектури, використання хмарних ресурсів або інших механізмів масштабування. Також включає в себе механізми моніторингу та управління для контролю стану системи, виявлення помилок та вчасного реагування на проблеми. Журналювання подій, моніторинг навантаження сервера, аналіз логів та інші інструменти.

### 3.2 Моделі та їх реалізація

Моделі в контексті WebSocket використовуються для представлення та обміну даними між сервером та клієнтами посередником в режимі реального

часу. У даному коді, модель Message використовується для представлення структури повідомлення, яке передається між сервером та клієнтом через вебсокети (див. рис. 3.1).

Коли клієнт надсилає повідомлення через вебсокети, дані повідомлення упаковуються в екземпляр моделі Message з відповідними властивостями, такими як ідентифікатор повідомлення, текст, ідентифікатор чату та ідентифікатор користувача. Ця модель потім передається на сервер.

На сервері, отримане повідомлення з вебсокетів розпаковується в екземпляр моделі Message, що дозволяє серверу отримати доступ до всіх даних повідомлення. Сервер може обробити це повідомлення, зберегти його в базі даних, відправити його до інших клієнтів або виконати будь-яку іншу логіку відповідно до вимог додатку.

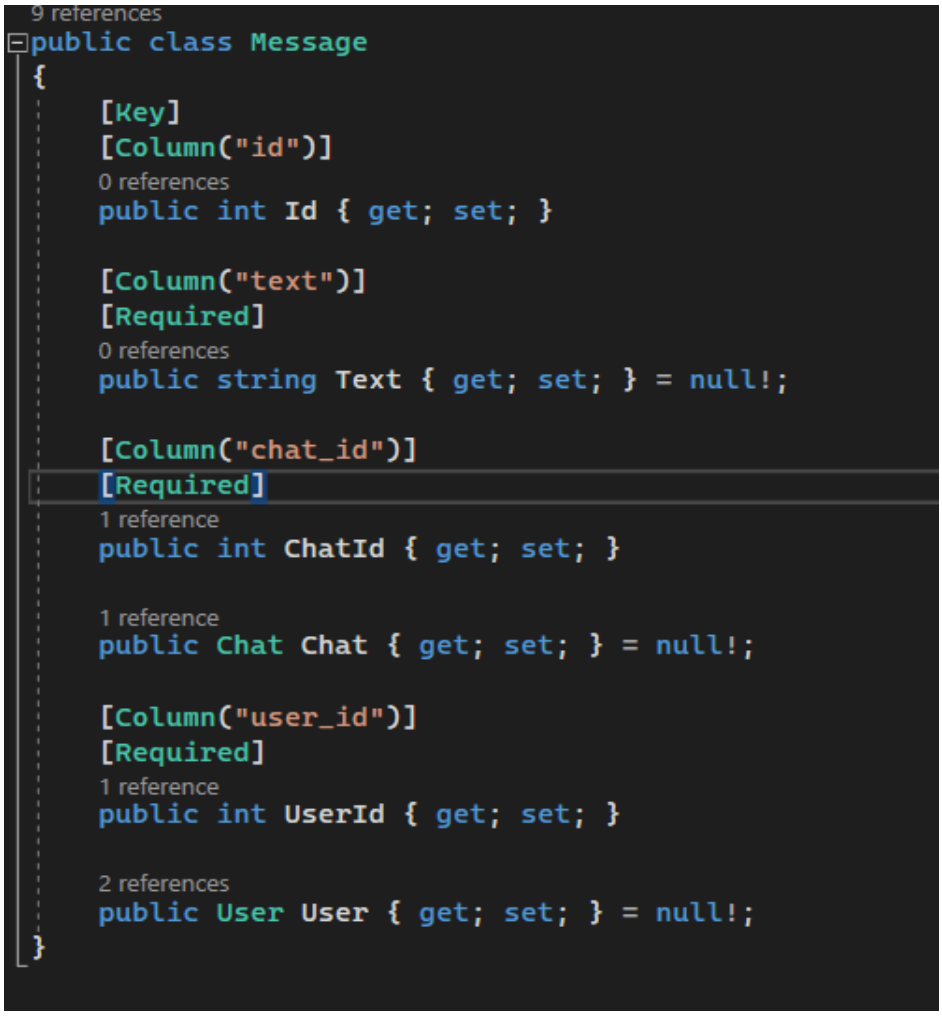


Рисунок 3.1 – Вигляд моделей в WebSocket

Зворотнім напрямком, коли сервер відправляє повідомлення до клієнта через вебсокети, він також може використовувати модель Message для упаковки даних повідомлення перед відправкою. Клієнт на своєму боці розпаковує отримане повідомлення в екземпляр моделі Message, що дозволяє йому отримати доступ до даних та відобразити повідомлення на користувацькому інтерфейсі.

Таким чином, модель Message в даному коді використовується для створення єдиного формату даних для обміну повідомленнями між сервером та клієнтами через вебсокети, спрощуючи обробку та передачу цих даних у режимі реального часу.

### 3.3 Контролери та їх реалізація

В контексті WebSocket, контролер відповідає за обробку вебсокет-запитів. Вебсокет-контролер є спеціальним класом, який може містити методи для обробки певних подій, пов’язаних з вебсокет-з’єднаннями, таких як встановлення з’єднання, отримання повідомлень і відправлення відповідей.

Основна мета контролерів вебсокетів (WebSocket controllers) полягає в обробці та керуванні вебсокет-з’єднаннями та їх подіями в рамках вебдодатка. Вони забезпечують взаємодію між клієнтами та сервером за допомогою двостороннього з’єднання WebSocket, що дозволяє передавати повідомлення в реальному часі без необхідності постійного встановлення нових HTTP-запитів.

Основні цілі використання контролерів вебсокетів включають наступні моменти.

*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.

Наприклад контролер UserController відповідає за обробку HTTP-запитів, пов’язаних з операціями користувачів в системі. Контролер використовує залежності ApplicationDbContext і IMapper, які він отримує через конструктор. Виконання логіки запитів здійснюється взаємодією з ApplicationDbContext для отримання даних з бази даних та IMapper для мапування об’єктів між різними типами даних (DTO, моделі). Також в цьому контролері реалізована реєстрація користувача в чат (див. рис. А.2).

### 3.4 Міграції та їх реалізація

Міграції в WebSocket використовуються для автоматичного керування структурою бази даних під час розгортання або оновлення додатку. Вони дозволяють визначати зміни в схемі бази даних, такі як створення нових таблиць, додавання або видалення стовпців, встановлення обмежень тощо, і застосовувати ці зміни до бази даних за допомогою коду.

В нашій реалізації представлений файл міграцій InitialTables. Цей файл визначає структуру таблиць і зв’язки між ними для створення початкової схеми бази даних. Він містить методи Up і Down, які відповідають за виконання змін в базі даних при розгортанні (Up) і відкату (Down) міграції.

Метод Up за допомогою об’єкта migrationBuilder виконуються декілька операцій, таких як створення таблиць Roles, Users, Chats і Messages, визначення стовпців і обмежень для кожної таблиці, а також встановлення зовнішніх ключів для забезпечення зв’язків між таблицями.

У методі Down виконується відкат змін, видаляються всі таблиці (Roles,

Users, Chats, Messages) і повертається база даних до початкового стану.

Міграції використовуються для забезпечення структурної цілісності бази даних і автоматизації процесу оновлення. При запуску додатку, система міграцій перевіряє поточний стан бази даних і автоматично застосовує потрібні зміни, якщо вони ще не були застосовані. Це дозволяє легко розгортати додаток на нових середовищах або оновлювати його без необхідності вручну втручатися в базу даних (див. рис. А.7).

### 3.5 Сигнал Р

SignalR – це бібліотека, яка надає функціональність реального часу в вебдодатках. Вона дозволяє забезпечити двосторонню комунікацію між сервером і клієнтом через різні протоколи, такі як вебсокети, Server-Sent Events (SSE) або Long Polling. Один з ключових компонентів SignalR – це сигнал Р (SignalR Hub).

Сигнал Р (SignalR Hub) – це клас на серверній стороні, який служить центральною точкою для обміну повідомленнями між сервером і клієнтами. Він дозволяє викликати методи на сервері з клієнта і навпаки, що дозволяє реалізувати багатокористувацькі інтерактивні додатки з миттєвим оновленням інформації.

У вебдодатках SignalR використовується для різних сценаріїв:

– реальний час оновлення: SignalR дозволяє надсилати повідомлення в реальному часі від сервера до клієнта і навпаки без необхідності оновлення сторінки;
– комунікація клієнта і сервера: сигнал Р дозволяє клієнтам викликати методи на сервері і навпаки, що дозволяє виконувати двосторонню комунікацію;
– розподілені додатки: SignalR підтримує масштабованість і розподілену обробку, що дозволяє побудувати додатки, які працюють в

розподіленому середовищі з багатьма серверами;

– групування клієнтів: SignalR дозволяє групувати клієнтів і надсилати повідомлення до цілих груп клієнтів. Це корисно для реалізації чат-подібних функціональностей або сповіщень для групи користувачів.

У вебдодатках SignalR може бути використаний для різних сценаріїв, таких як спілкування в реальному часі, сповіщення, оновлення стану додатка тощо. Використовуючи SignalR, розробники можуть побудувати більш інтерактивні і динамічні вебдодатки, які надають користувачам миттєву зворотну зв’язок та оновлення інформації в режимі реального часу.

## 3.6 Тестування онлайн консультанта

У цьому підрозділі проводиться функціональне тестування розробленого онлайн консультанта з метою перевірки його відповідності функціональним вимогам та очікуванням користувачів. Функціональне тестування дозволяє виявити потенційні проблеми та дефекти в роботі консультанта та переконатися, що всі функції працюють належним чином. Метою функціонального тестування є забезпечення роботи консультанта відповідно до очікувань користувачів, виявлення потенційних дефектів та покращення його якості. У даній роботі будуть використані наступні тестові сценарії:

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 рисунку 3.2 – 3.4 реалізовано тестування нашого онлайн консультанта.

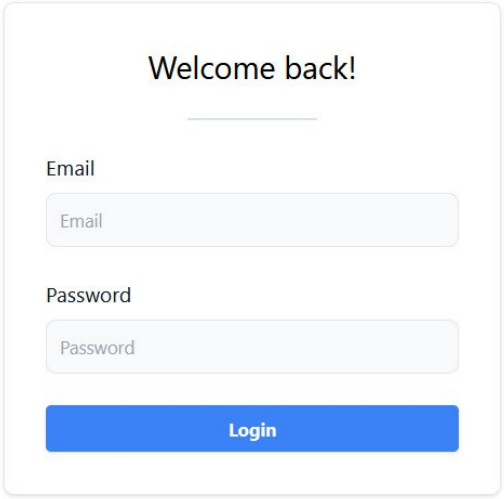


Рисунок 3.2 – Авторизація користувача

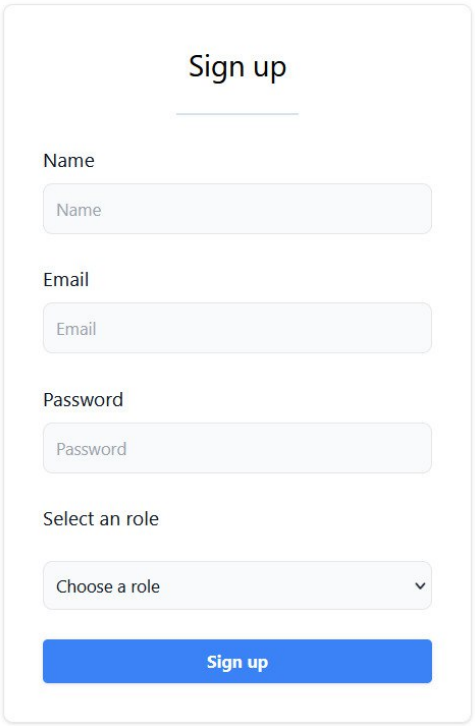


Рисунок 3.3 – Реєстрація користувача

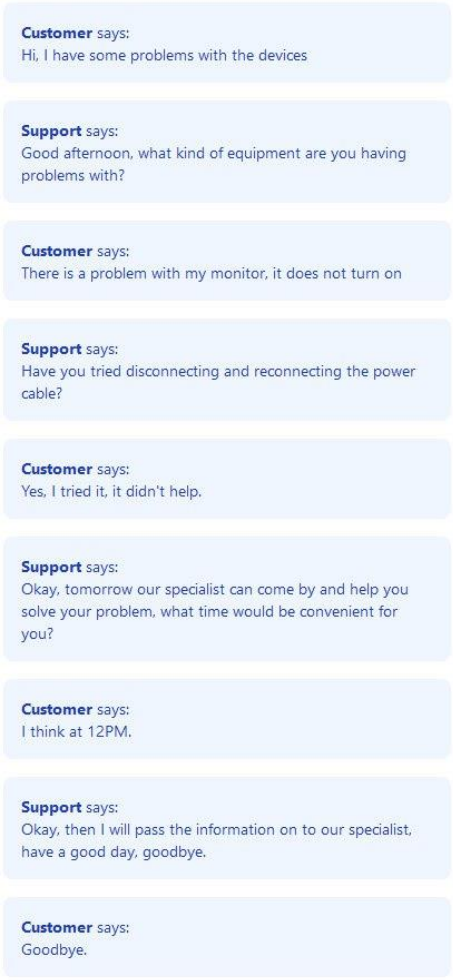


Рисунок 3.4 – Взаємодія користувача з онлайн консультантом

## 3.7 Висновки до розділу 3

У рамках розділу «Реалізація та тестування» 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

В рамках дипломної роботи була проведена розробка та реалізація онлайн консультанта для вебсайтів з використанням WebSocket. Робота включала аналіз вимог, проєктування системи, реалізацію функціональності, тестування та оцінку якості.

Основні результати:

– було розроблено функціональний онлайн консультант, який надає можливість спілкування користувачів з консультантами на вебсайтах через WebSocket з’єднання;
– вебдодаток був реалізований з використанням технологій, таких як WebSocket, MySQL, React та C#, що дозволяють забезпечити зручний та ефективний інтерфейс користувача;
– були використані різні технології та інструменти для досягнення поставлених цілей проєкту, включаючи використання WebSocket для забезпечення реального часу спілкування, бази даних MySQL для збереження інформації та фреймворку React для клієнтської частини.

Оцінка якості:

– в процесі тестування були виявлені та виправлені деякі помилки та проблеми, що сприяло поліпшенню якості системи;
– були застосовані функціональні тести для перевірки коректності роботи системи та взаємодії між її компонентами.

В цілому, дипломна робота була успішно завершена, мета проєкту була досягнута, а розроблений онлайн консультант готовий до використання. Результати роботи можуть бути використані для подальшого розгортання та використання системи в реальних умовах.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Документації WebSocket. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 03.04.2023).
2. Документація React. URL: https://vuejs.org/guide/introduction.html (дата звернення: 04.04.2023).
3. Документація C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 05.04.2023).
4. Документація MySQL. URL: https://dev.mysql.com/doc/ (дата звернення: 08.04.2023).
5. Документація ASP.NET. URL: https://jakeydocs.readthedocs.io/en/latest/ (дата звернення: 10.04.2023).
6. Додаткова інформація про сокети. URL: https://learn.javascript.ru/websockets (дата звернення: 11.04.2023).
7. Переваги і недоліки Вебсокетів. URL: https://what.com.ua/javascript-websocket-opis-pr/ (дата звернення: 14.04.2023).
8. Переваги і недоліки React. URL: https://www.affiliatebay.net/uk/advantages-and-disadvantages-of-react-js/ (дата звернення: 21.04.2023).

# ДОДАТОК А

## Додаткові приклади реалізаціїї окремих комопнентів консультанта

### А.1 Компонент запуску вебдодатка

У даній частині коду я реалізував конфігурацію та запуск вебдодатка з використанням фреймворка ASP.NET Core.

На початку програми ми отримуємо поточну директорію та шлях до файлу .env, який містить конфігураційні змінні. Завантажуємо ці змінні за допомогою бібліотеки DotEnv для зручного доступу до них.

```
public static class Program
{
   public static void Main(string[] args)
   {
      var root = Directory.GetCurrentDirectory();
      var dotenv = Path.Combine(root, ".env");
      DotEnv.Load(dotenv);
      WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

      builder.Services.AddControllersWithViews().AddJsonOptions(options =>
      {
         options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
      });
      builder.Services.AddSignalR();

      builder.Services.AddCors(options =>
      {
         options.AddPolicy("ClientPermission", policy =>
         {
            policy.AllowAnyHeader()
               .AllowAnyMethod()
               .WithOrigins(Environment.GetEnvironmentVariable("APP_URL")!)
               .AllowCredentials();
         });
      });

      builder.Services.AddAuthentication(opt =>
      {
         opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
         opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
      }).AddJwtBearer(options =>
      {
         options.TokenValidationParameters = new TokenValidationParameters
         {
```

```
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = Environment.GetEnvironmentVariable("APP_URL")!,
            ValidAudience = Environment.GetEnvironmentVariable("APP_URL")!,
            IssuerSigningKey = new SymmetricSecurityKey(
               Encoding.UTF8.GetBytes(Environment.GetEnvironmentVariable("JWT_SECRET_KEY")!)
            )
        };
    });

    var dbName = Environment.GetEnvironmentVariable("DB_NAME")!;
    var dbHostname = Environment.GetEnvironmentVariable("DB_HOSTNAME")!;
    var dbUser = Environment.GetEnvironmentVariable("DB_USER")!;
    var dbPassword = Environment.GetEnvironmentVariable("DB_PASSWORD")!;

    builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());
    builder.Services.AddDbContextPool<ApplicationDbContext>(options => options.UseMySql(
       $"server={dbHostname};user={dbUser};password={dbPassword};database={dbName};",
       new MySqlServerVersion(new Version(8, 0, 11))
    ));
    builder.Services.AddAutoMapper(typeof(MapperUser), typeof(MapperChat));

    WebApplication app = builder.Build();
    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseCors("ClientPermission");
    app.UseAuthentication();
    app.UseRouting();
    app.UseAuthorization();
    app.MapHub<ChatHub>("/hubs/chat");
    app.MapDefaultControllerRoute();

    app.MapFallbackToFile("index.html");

    app.Run();
  }
}
```

Рисунок А.1 – Конфігурація вебдодатку

Потім ми створюємо екземпляр WebApplicationBuilder, який дозволяє налаштувати та створити вебдодаток. Ми додаємо сервіси контролерів з представленнями та налаштовуємо серіалізацію JSON, щоб ігнорувати циклічні посилання. Також додаємо підтримку SignalR для роботи з вебсокетами.

Далі ми налаштовуємо політику CORS (Cross-Origin Resource Sharing), щоб дозволити доступ до нашого додатка з певних джерел. Ми дозволяємо будь-які заголовки та методи, вказуємо джерело (APP_URL) зі змінних середовища та дозволяємо передачу облікових даних.Потім йде налаштування

аутентифікації з використанням схеми JWT Bearer. Ми встановлюємо параметри перевірки токена, такі як перевірка видавця, аудиторії, часу життя та підпису. Далі йде налаштування підключення до бази даних MySQL. Ми отримуємо значення назви бази даних, хоста, користувача та пароля зі змінних середовища. Використовується пул з’єднань DbContextPool та налаштування версії MySQL 8.0.11.

Потім ми додаємо сервіс AutoMapper для мапування об’єктів між класами. Здесь также указаны конкретные классы MapperUser та MapperChat, необходимые для маппинга. Далі ми створюємо екземпляр вебдодатку та налаштовуємо його. Використовується роутинг, авторизація, аутентифікація, обробка статичних файлів та маршрутів контролерів. Накінець, за допомогою MapFallbackToFile ми налаштовуємо поведінку додатка для неіснуючих маршрутів, вказуючи файл index.html як резервний варіант.

### А.2 Компонент створення регістрації клієнат

На початку класу ми визначаємо приватні поля _applicationContext та _mapper, які будуть використовуватися для доступу до бази даних та мапування об’єктів.У конструкторі контролера ми отримуємо залежності ApplicationDbContext та IMapper і зберігаємо їх у відповідних полях для подальшого використання.

Метод [HttpPost(«register»)] відповідає за реєстрацію нового користувача. Ми спочатку перевіряємо валідність моделі UserRegisterDto. Якщо модель не є валідною, повертається статус помилки 400 (BadRequest).

Пароль користувача хешується за допомогою бібліотеки BCrypt.Net. Потім створюється новий користувач з використанням маппера _mapper та зберігається в базі даних. Після збереження користувача ми створюємо JWT-токен з використанням серкетного ключа, який також береться зі змінних середовища.

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

      user.Password = BCrypt.Net.BCrypt.HashPassword(user.Password);
      var userResult = _applicationContext.Users.Add(_mapper.Map<User>(user));
      await _applicationContext.SaveChangesAsync();
      var secretKey =
         new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Environment.GetEnvironmentVariable("JWT_SECRET_
KEY")!));
      var signinCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);
      var userData = _applicationContext.Users
         .Include(u => u.Role)
         .First(x => x.Id == userResult.Entity.Id);

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

Рисунок А.2 – Реєстрація користувача

### А.3 Компонент для авторизації користувача

Метод [HttpPost(«login»)] відповідає за авторизацію користувача. Ми також перевіряємо валідність моделі UserLoginDto. Якщо модель не є валідною, повертається статус помилки 400 (BadRequest).Ми перевіряємо валідність введених даних авторизації з допомогою методу IsLoginDataValid. Якщо дані неправильні, повертається статус помилки 401 (Unauthorized). У разі успішної авторизації ми створюємо JWT-токен, аналогічно до методу реєстрації, і повертаємо користувача разом з токеном.

Приватний метод IsLoginDataValid використовується для перевірки

валідності введених даних авторизації. Він здійснює перевірку хешованого пароля користувача з базою даних. Якщо паролі збігаються, метод повертає значення true, в іншому випадку – false.

Метод [Authorize] [HttpGet(«me»)] дозволяє отримати дані про авторизованого користувача. Ми отримуємо ідентифікатор користувача з клеймів JWT-токена, використовуючи HttpContext.User.Identity. Потім з бази даних витягується користувач за його ідентифікатором і мапується на об’єкт CleanUserDto, який містить необхідну інформацію про користувача. Якщо користувач не знайдений, повертається статус помилки 404 (NotFound).

```
[HttpPost("login")]
    public IActionResult Login(UserLoginDto loginData)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        if (!IsLoginDataValid(loginData))
        {
            return Unauthorized();
        }

        var userData = _applicationContext.Users
            .Include(u => u.Role)
            .First(x => x.Email == loginData.Email);
        var secretKey =
            new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Environment.GetEnvironmentVariable("JWT_SECRET_
KEY")!));
        var signinCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);

        JwtSecurityToken tokenOptions = new JwtSecurityToken(
            issuer: Environment.GetEnvironmentVariable("APP_URL"),
            audience: Environment.GetEnvironmentVariable("APP_URL"),
            claims: new List<Claim> { new("Id", userData.Id.ToString()), new("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 Ok(userWithToken);
    }

    private bool IsLoginDataValid(UserLoginDto loginData)
```

```
    {
        var userData = _applicationContext.Users.First(x => x.Email == loginData.Email);
        return BCrypt.Net.BCrypt.Verify(loginData.Password, userData.Password);
    }


    [Authorize]
    [HttpGet("me")]
    public ActionResult<CleanUserDto> GetLoggedInData()
    {
        int userId = 0;
        if (HttpContext.User.Identity is ClaimsIdentity identity)
        {
            userId = Convert.ToInt32(identity.FindFirst("Id")!.Value);
        }

        var user = _applicationContext.Users
            .Include(u => u.Role)
            .FirstOrDefault(x => x.Id == userId);

        if (user is null)
        {
            return NotFound();
        }

        return _mapper.Map<CleanUserDto>(user);
    }
}
```

Рисунок А.3 – Авторизація користувача

### А.4 Компонент створеня повідомленть в чаті

У цьому компоненті я реалізував контролер MessageController, який відповідає за створення повідомлень в чаті. На початку класу ми визначаємо приватні поля _dbContext та _mapper, які будуть використовуватися для доступу до бази даних та мапування об’єктів. У конструкторі контролера ми отримуємо залежності ApplicationDbContext та IMapper і зберігаємо їх у відповідних полях для подальшого використання.

Метод [Authorize] [HttpPost] відповідає за створення нового повідомлення в чаті. Ми спочатку перевіряємо валідність моделі CreateMessageDto. Якщо модель не є валідною, повертається статус помилки 400 (BadRequest).

Отримуємо ідентифікатор користувача з клеймів JWT-токена за

допомогою HttpContext.User.Identity. Потім присвоюємо ідентифікатор користувача UserId властивості message, яка містить дані про нове повідомлення.Далі ми отримуємо чат з бази даних за його ідентифікатором ChatId. Якщо чат вже закритий (Resolved), повертається статус помилки 400 (BadRequest).Також ми перевіряємо, чи користувач має дозвіл на додавання повідомлення до чату. Якщо користувач не є власником чату (CustomerId) і не має роль "employee", повертається статус помилки 400 (BadRequest).

Додаваємо нове повідомлення до бази даних за допомогою _dbContext.Messages.Add, після чого зберігаємо зміни за допомогою _dbContext.SaveChangesAsync.

```
public class MessageController : ControllerBase
{
    private readonly ApplicationDbContext _dbContext;
    private readonly IMapper _mapper;

    public MessageController(ApplicationDbContext context, IMapper mapper)
    {
        _dbContext = context;
        _mapper = mapper;
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<ChatDto>> CreateMessage(CreateMessageDto message)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        int userId = 0;
        if (HttpContext.User.Identity is ClaimsIdentity identity)
        {
            userId = Convert.ToInt32(identity.FindFirst("Id")!.Value);
        }

        message.UserId = userId;
        var chat = _dbContext.Chats.First(chat => chat.Id == message.ChatId);

        if (chat.Resolved)
        {
            return BadRequest();
        }

        var user = _dbContext.Users
            .Include(u => u.Role)
```

```
        .First(x => x.Id == userId);

    if (userId != chat.CustomerId && user.Role.Name != "employee")
    {
        return BadRequest();
    }

    _dbContext.Messages.Add(_mapper.Map<Message>(message));
    await _dbContext.SaveChangesAsync();

    var updatedChat = _dbContext.Chats
        .Include(c => c.Customer)
        .ThenInclude(u => u.Role)
        .Include(c => c.Messages)
        .ThenInclude(m => m.User)
        .First(c => c.Id == message.ChatId);

    return _mapper.Map<ChatDto>(updatedChat);
```

Рисунок А.4 – Авторизація користувача

### А.5 Компонент обробки запитів чату

Контролер ChatController дозволяє створювати нові чати, отримувати список чатів користувача та позначати чати як вирішені. Він використовує екземпляр ApplicationDbContext для взаємодії з базою даних і IMapper для виконання мапування об’єктів. Метод CreateChat створює новий чат і зберігає його у базі даних. Повертає об’єкт ChatDto з деталями нового чату. Метод GetAllUserChats повертає список всіх чатів користувача. Отримує ідентифікатор користувача з токена автентифікації і запитує базу даних про всі чати, пов’язані з цим користувачем. Повертає список об’єктів ChatDto, що містять деталі кожного чату. Метод MarkAsResolved позначає чат як вирішений. Приймає POST-запит з ідентифікатором чату і перевіряє, чи користувач, що надсилає запит, є власником чату. Якщо так, то встановлює прапорець Resolved у значення true. Після збереження змін у базі даних повертає об’єкт ChatDto з оновленими даними чату.

```
public class ChatController : ControllerBase
{
    private readonly ApplicationDbContext _applicationContext;
```

```
    private readonly IMapper _mapper;

    public ChatController(ApplicationDbContext context, IMapper mapper)
    {
        _applicationContext = context;
        _mapper = mapper;
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<ChatDto>> CreateChat()
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        int userId = 0;
        if (HttpContext.User.Identity is ClaimsIdentity identity)
        {
            userId = Convert.ToInt32(identity.FindFirst("Id")!.Value);
        }

        CreateChatDto createChat = new() { CustomerId = userId, Resolved = false };

        var createdChat = _applicationContext.Chats.Add(_mapper.Map<Chat>(createChat));
        await _applicationContext.SaveChangesAsync();
        var chat = _applicationContext.Chats
            .Include(c => c.Customer)
            .ThenInclude(u => u.Role)
            .Include(c => c.Messages)
            .First(c => c.Id == createdChat.Entity.Id);

        return _mapper.Map<ChatDto>(chat);
    }

    [Authorize]
    [HttpGet]
    public ActionResult<List<ChatDto>> GetAllUserChats()
    {
        int userId = 0;
        if (HttpContext.User.Identity is ClaimsIdentity identity)
        {
            userId = Convert.ToInt32(identity.FindFirst("Id")!.Value);
        }

        var chat = _applicationContext.Chats
            .Include(c => c.Customer)
            .ThenInclude(u => u.Role)
            .Include(c => c.Messages)
            .Where(u => u.CustomerId == userId);

        return _mapper.Map<List<ChatDto>>(chat.ToList());
    }

    [Authorize]
    [HttpPost("{id:int}")]
```

```
    public async Task<ActionResult<ChatDto>> MarkAsResolved(int id)
    {
        int userId = 0;
        if (HttpContext.User.Identity is ClaimsIdentity identity)
        {
            userId = Convert.ToInt32(identity.FindFirst("Id")!.Value);
        }

        var chat = _applicationContext.Chats
            .Include(c => c.Customer)
            .ThenInclude(u => u.Role)
            .Include(c => c.Messages)
            .First(c => c.Id == id);

        if (userId != chat.CustomerId)
        {
            return BadRequest();
        }

        chat.Resolved = true;
        await _applicationContext.SaveChangesAsync();

        return _mapper.Map<ChatDto>(chat);
    }
}
```

Рисунок А.5 – Керування чатами

### А.6 Компонент для зміни керування базою даних

ApplicationDbContext – це клас, який успадковує від DbContext в Entity Framework Core. Він використовується для взаємодії з базою даних і містить набір DbSet властивостей для кожної з таблиць бази даних: Users, Messages, Roles і Chats. У конструкторі ApplicationDbContext приймаються параметри DbContextOptions, які передаються до базового конструктора DbContext.

Метод OnModelCreating використовується для визначення взаємозв’язків між таблицями та конфігурації моделі даних. У цьому методі викликаються різні методи HasOne, WithMany, HasForeignKey та OnDelete, щоб встановити зв’язки між моделями. Наприклад, встановлюється зв’язок між таблицями User і Role, Chat і User, Chat і Message, а також Message і User.

Цей клас використовується для взаємодії з базою даних, виконання запитів і збереження змін у таблицях.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<User> Users { get; set; } = null!;
    public DbSet<Message> Messages { get; set; } = null!;
    public DbSet<Role> Roles { get; set; } = null!;
    public DbSet<Chat> Chats { get; set; } = null!;

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>()
            .HasOne(e => e.Role)
            .WithMany(e => e.Users)
            .HasForeignKey(e => e.RoleId)
            .IsRequired();

        modelBuilder.Entity<Chat>()
            .HasOne(chat => chat.Customer)
            .WithMany(user => user.Chats)
            .HasForeignKey(chat => chat.CustomerId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Chat>()
            .HasMany(e => e.Messages)
            .WithOne(e => e.Chat)
            .HasForeignKey(e => e.ChatId)
            .IsRequired();

        modelBuilder.Entity<Message>()
            .HasOne(msg => msg.User)
            .WithMany(usr => usr.Messages)
            .HasForeignKey(msg => msg.UserId)
            .OnDelete(DeleteBehavior.Restrict);
    }
}
```

Рисунок А.6 – Керування базою даних

Відображення коду міграції який створює таблиці «Roles», «Users», «Chats» та «Messages» з відповідними колонками у базі даних. Також встановлюється зв’язок між таблицями за допомогою зовнішніх ключів. Код міграції також містить деякі анотації, що вказують на використання UTF-8 кодування для деяких текстових полів у таблицях.

```
public partial class InitialTables : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
```

```
    {
        migrationBuilder.AlterDatabase()
            .Annotation("MySql:CharSet", "utf8mb4");

        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"),
                role_id = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Users", x => x.id);
                table.ForeignKey(
                    name: "FK_Users_Roles_role_id",
                    column: x => x.role_id,
                    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),
                customer_id = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
```

```
            {
                table.PrimaryKey("PK_Chats", x => x.id);
                table.ForeignKey(
                    name: "FK_Chats_Users_customer_id",
                    column: x => x.customer_id,
                    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"),
                chat_id = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Messages", x => x.id);
                table.ForeignKey(
                    name: "FK_Messages_Chats_chat_id",
                    column: x => x.chat_id,
                    principalTable: "Chats",
                    principalColumn: "id",
                    onDelete: ReferentialAction.Cascade);
            })
            .Annotation("MySql:CharSet", "utf8mb4");

        migrationBuilder.CreateIndex(
            name: "IX_Chats_customer_id",
            table: "Chats",
            column: "customer_id");

        migrationBuilder.CreateIndex(
            name: "IX_Messages_chat_id",
            table: "Messages",
            column: "chat_id");

        migrationBuilder.CreateIndex(
            name: "IX_Users_role_id",
            table: "Users",
            column: "role_id");
    }

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

        migrationBuilder.DropTable(
```

```
            name: "Chats");

        migrationBuilder.DropTable(
            name: "Users");

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

Рисунок А.7 – Міграції