МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СЕРВІСУ «ОНЛАЙН-КОНСУЛЬТАНТ» НА БАЗІ ПРОТОКОЛУ WEBSOCKET»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1-дн
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
О.Р. Рудьковський
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(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 – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

Рудьковському Олексію Руслановичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу «Онлайн-консультант» на базі протоколу WebSocket

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 26.02.2024 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. Проєктування системи.
3. Реалізація та тестування системи.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 03.05.2023 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.	08.05.2023	
2.	Збір вихідних даних.	09.06.2023	
3.	Обробка методичних та теоретичних	31.08.2023	
	джерел.		
4.	Розробка першого та другого розділу.	15.12.2023	
5.	Розробка третього розділу.	05.02.2024	
6.	Оформлення та нормоконтроль	19.02.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.03.2024	

Студент _____ О.Р. Рудьковський
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка сервісу «Онлайн-консультант» на базі протоколу WebSocket»: 53 с., 24 рис., 6 табл., 15 джерел, 1 додаток.

ВЕБДОДАТОК, ПРОГРАМУВАННЯ, JAVASCRIPT, NODEJS, VUEJS, WEBSOCKET.

Об’єкт дослідження – процес розробки сервісу «Онлайн-консультант» на базі протоколу WebSocket.

Мета роботи: розробити систему «Онлайн-консультант» на базі протоколу WebSocket.

Методи дослідження – методи об’єктно-орієнтовного програмування, методи програмної інженерії.

Проведено огляд існуючих схожих проєктів, а також опис інструментів і технологій для розробки системи, аналіз використаних технологій. Розроблено структуру системи, UML-діаграми, схеми бази даних, що проілюстровано ER-діаграмою, та діаграму класів.

Кінцевий вебдодаток було реалізовано використовуючи для обох частин Frontend та Backend, а також окремого сервісу для реалізації протоколу WebSocket. При реалізації використовувались такі мови програмування як PHP, JavaScript.

Результати роботи можуть бути використані при організації системи допомоги на онлайн-сервісах, зокрема у інтернет магазинах, блогах або складних вебдодатків.

SUMMARY

Master’s qualifying paper «Development of the "Online Consultant" Service Based on the WebSocket Protocol»: 53 pages, 24 figures, 6 tables, 15 references, 1 supplement.

WEBAPP, PROGRAMMING, JAVASCRIPT, NODEJS, VUEJS, WEBSOCKET.

The object of the study is the process of developing the “Online consultant” service based on the WebSocket protocol.

The aim of the study is to develop the “Online consultant” system based on the WebSocket protocol.

The method of research is methods of object-oriented programming, methods of software engineering.

An overview of existing similar projects was carried out, as well as a description of tools and technologies for system development, an analysis of the technologies used. Developed system structure, UML diagrams, database schema illustrated by ER diagram, and class diagram.

The final web application was implemented both Frontend and Backend parts, as well as a separate service for implementing the WebSocket protocol. Programming languages such as PHP and JavaScript were used in the implementation.

The results of the work can be used in the organization of the help system on online services, in particular in online stores, blogs or complex web applications.

ЗМІСТ

ВСТУП

У сучасному світі через велику кількість користувач мережі Інтернет стає актуальним питання підтримки користувачів на вебсайтах. Користувачі можуть стикатись з різними проблемами під час користування продуктом, будь то звичайний блог, інтернет-магазин чи великий і складний сервіс з безліччю функцій. Користувач продукту може стискатись з проблемами, для вирішення яких необхідно звертатись до служби підтримки проєкту. Оскільки кількість користувачів зазвичай перевищує кількість співробітників, то постає питання для вчасного обслуговування запитів.

Одним із можливих варіантів вирішення проблеми є використання окремного сервісу для надання допомоги користувачам у вигляді чату, де один агент підтримки зможе одночасно допомагати декільком користувачам продукту.

Таке програмне забезпечення має надавати можливість звернутись за допомогою у будь-який час і надати можливість власникам проєкту допомогти якомога більшій кількості користувачів.

Метою роботи є розробка сервісу «Онлайн-консультант» на базі протоколу WebSocket з використанням мов програмування PHP та JavaScript.

Завданнями дослідження є:

– проаналізувати існуючі сервіси для надання допомоги користувачам;
– розробити сервіс з використанням мов програмування PHP та JavaScript;
– протестувати розроблену системи.

Об’єктом дослідження є процес розробки сервісу «Онлайн-консультант» на базі протоколу WebSocket.

Методами дослідження є методи об’єктно-орієнтовного програмування, методи програмної інженерії.

У ході вирішення поставлених завдань було отримано такі результати:

проаналізовано ряд подібних існуючих проєктів, спроєктовано структуру застосунку , описано основні принципи роботи програмного забезпечення, реалізовано клієнтську та серверну частину і протестовано розроблену систему.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань (15 найменувань) та одного додатку. Перший розділ присвячений аналізу існуючих сервісів та технологій реалізації сервісу «Онлайн-консультанта», вибір мови програмування та супутніх технологій зберігання та обміну інформації. Другий розділ містить процес проєктування сервісу, розробки UML діаграм, діаграм класів та схеми бази даних. У третьому розділі наведено основні етапи розробки програмної реалізації системи «Онлайн-консультант», тестування коду та перевірка роботоспроможності системи.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Передумови та загальний опис проєкту

Зі збільшенням кількості користувачів мережі Інтернет і збільшенням розповсюдженням між людьми різних професій і навичок у тій чи іншій сфері, при користуванні деякими вебдодатками та вебсайтами у них можуть з’являтися питання, відповіді на які вони не зможуть дати самі собі. Для вирішення таких питань і для збереження користувачів постає питання надання допомоги таким користувачам.

Для надання допомоги можна використовувати різні канали зв’язку, а саме: телефонні дзвінки, електрону пошту, чат на сайті. Розглянемо детальніше переваги і недоліки кожного з цих каналів.

При використанні телефонних дзвінків уся увага агента підтримки зосереджена на вирішенні однієї проблеми одного користувача. Користувач отримує всю увагу агента і рішення буде знайдено значно швидше. Недоліком є необхідність мати великої кількості агентів підтримки, бо зі зростанням кількості користувачів треба пропорційно збільшувати штат служби підтримки. Також до недоліків можна віднести складність реалізації автоматичної допомоги користувачам без залучання агентів підтримки. Також неможливим є обмін файлами, що у деяких випадках може бути критичним для вирішення проблеми, особливо коли необхідно вирішити технічну проблему і треба дивитись якісь звіти або фотографії.

Під час використання електронної пошти питання буде вирішено повільніше, проте один агент підтримки може працювати над питаннями багатьох користувачів і є можливість обмінюватись файлами. Іншою перевагою є відсутність необхідності інтеграції якихось додаткових систем і достатньо лише створити відповідний поштовий ящик для всіх агентів підтримки.

Останнім розглянутим каналом зв’язку був чат на сайті. Чат поєднує переваги та недоліки обох систем. До переваг чату відноситься можливість працювати над декількома питаннями одному агенту (та декільком агентам над одним питанням за потреби), швидкість відповіді та рівень залучення агента підтримки у вирішення проблеми, можливість обмінюватись файлами. Недоліком є необхідність додаткової реалізації чату або підключення стороннього сервісу [1].

Враховуючі різні вподобання людей та різну критичність запитів, найкращим варіантом є підтримка усіх трьох каналів зв’язку.

Використання сучасних технологій, таких як PHP, JavaScript, WebSockets та SQL дозволяє створювати складні і швидкі додатки. Ці технології і мови програмування є широко розповсюдженими, мають велику кількість готових рішень, тому пропонується для реалізації системи використати саме їх.

PHP є однією з найпопулярніших мов програмування, яка використовується в широкому спектрі розробок. JavaScript зайняв своє місце у застосунках, де необхідна швидкість у комунікації між клієнтом і сервером, а також для створення динамічних сайтів. SQL є мовою програмування реляційних баз даних, яка дозволяє працювати з великою кількістю даних. WebSocket є технологією обміну інформації між клієнтом та сервером, яка значно пришвидшує обмін повідомленнями та запитами.

Використовуючи переваги наведених мов програмування, можна створити швидку, безпечну та надійну систему обміну повідомленнями. Сервіс онлайн-консультанту, з використанням сучасних і швидких технологій, може стати надійним інструментом для компаній для покращення якості надання власних послуг.

Таким чином, метою роботи є створення системи онлайн-консультанта з використанням технології WebSocket. Ця система розроблена з використанням фреймворку Laravel мови програмування PHP, фреймворку

VueJS для розробки користувацького інтерфейсу, а також NodeJS для сервера обміну повідомленнями між клієнтом і сервером за технологією WebSocket.

1.2 Огляд подібних проєктів

Існує багато готових сервісів для реалізації функції інтернет-консультанта, націлених на різні групи проєктів та клієнтів. Всі вони мають як свої переваги, так і недоліки. Далі наведено і розглянуто найбільш популярні існуючі системи.

«SendPulse» це українська платформа з інструментами для маркетингу та автоматизації продажів. Крім чату для сайту, у SendPulse є email-сервіс, безплатна CRM, конструктор лендінгів, платформа EDU для розробки та монетизації власних онлайн-курсів, SMS-розсилки, web push-повідомлення, чат-боти для Telegram, Instagram, Facebook Messenger, WhatsApp, Viber. До основних функцій і переваг цієї системи належать можливість підключення штучного інтелекту, збереження історії листування з клієнтом, прийом платежів у чаті, можливість надсилати фото та файли. Для клієнтів SendPulse є цілодобова служба підтримки [2].

«LexxChat» – частина платформи з розвитку інтернет-продажу Lexx.me. Lexx.me – це повноцінний сервіс для створення власного інтернет-магазина, з яким ви можете швидко завантажувати свої товари на Rozetka, Prom, Google Shopping, Facebook Shop тощо. Основними перевагами цього сервісу є можливість інтеграції з Telegram та Viber, пересилання файлів, сповіщення операторів на пошту, база знань для самообслуговування клієнтів, яка вбудована у вікно онлайн-чату [3].

«LiveChat» – це програмне забезпечення для онлайн-обслуговування клієнтів з онлайн-чатом, програмним забезпеченням служби підтримки та можливостями вебаналітики. Вперше він був запущений в 2002 році, а зараз розробляється і пропонується в рамках бізнес-моделі SaaS компанією LiveChat

Software S.A. До переваг можна віднести мобільний додаток для iOS та Android, наявність бази знань для клієнтів. До недоліків варто віднести те, що сервіс має лише англійську версію [4].

До популярних систем варто додати ZenDesk. Zendesk – це онлайн-платформа для клієнтської підтримки, яка гнучко адаптується до потреб різних бізнесів – від малих стартапів до корпоративних гігантів. Саме тому вже понад 100 000 компаній світу, серед яких Uber, Siemens, Vimeo, GCash, обрали її.

Zendesk спрощує комунікацію з користувачами, допомагаючи швидко обробляти запити і таким чином підвищуючи лояльність клієнтів. Це дійсно важливо, оскільки репутація компанії залежить від оцінок та відгуків покупців.

Цей багатофункціональний додаток має всі необхідні інструменти для покращення якості обслуговування клієнтів – безперервний зв'язок з можливістю інтеграції понад 50 каналів комунікації, автоматизовану звітність і статистику, точні показники для швидкої організації та впорядкування бази даних клієнтів.

Платформа дозволяє одночасно вирішувати низку питань:

– централізовано обробляти всі згенеровані тікети;
– управляти всіма вхідними лідами для відділу продажу;
– оцінювати ефективність менеджерів в режимі реального часу;
– зберігати файли, що надсилаються клієнтами;
– створювати інтелектуальні бази знань з готовими рішеннями;
– автоматично опрацьовувати більшість клієнтських звернень [5].

Також дуже популярним сервісом є LiveAgent. Однією з переваг є його здатність сприяти оперативному обслуговуванню по всіх каналах – від чату та соціальних мереж до електронної пошти та телефону. Перевага LiveAgent полягає в його хмарній функціональності чату та широкій функціональності служби підтримки.

У LiveAgent можна створювати власні вікна чату, інтелектуально

спрямовувати питання через чат, автоматизувати основні фітчі та багато іншого. За допомогою LiveChat ви можете налаштовувати вікно чату, керувати трафіком чату за допомогою інтелектуальної маршрутизації, надсилати автоматичні вітання та багато іншого. Отже, якщо ви шукаєте рішення для відстеження проблем через чат, варто звернути увагу на LiveAgent. Серед переваг є наявність API, користувацькі поля відкритих питань, що дозволяє більш точніше вказувати питання користувачам, та можливість налаштування повідомлень у Slack [6].

Наведені платформи є найбільш популярними та використовуються частіше за інших. Кожна з них є як базові функції для таких систем, наприклад можливість вбудови чату, підтримка декількох агентів підтримки, доступність, простота використання.

1.3 Засоби розробки інформаційної системи

Розробка системи складається з декількох етапів, які впливають на створення функціонального і надійного програмного забезпечення.

Першим етапом є формування списку вимог. На цьому етапі визначаються цілі, завдання та етапи розробки. Також обираються методи та інструменти. Ці рішення є ключовими для формування процесу розвитку системи.

Наступною стадією є розробка дизайну системи. Існує два аспекти дизайну: дизайн інтерфейсу та дизайн структури:

– діаграми класів;
– діаграми послідовності;
– діаграми розгортання;
– діаграми компонентів.

Зв’язок між дизайном інтерфейсу та дизайном структури є дуже важливим для загального успіху системи. Поки дизайн інтерфейсу створює

зручний досвід користування додатком або вебсайтом, дизайн структури гарантує швидкість роботи, відмовостійкість та простоту розширення функцій і підтримки готового продукту.

Третім етапом є власне розробка системи. Цей етап включає в себе реалізацію користувацького інтерфейсу, спираючись на результати попереднього етапу, а також реалізацію серверної частини, що включає налаштування необхідного програмного забезпечення і сервісів, наприклад бази даних. Під час цього етапу основний фокус робиться на написанні коду, що є основою програмного забезпечення.

Це часто передбачає використання таких технологій, як JavaScript, універсальна мова програмування, яка широко використовується в сучасній веброзробці, для збагачення взаємодії користувачів динамічними та адаптованими компонентами. JavaScript, який функціонує як мова сценаріїв на стороні клієнта, має важливе значення для надання миттєвих оновлень, асинхронного зв’язку та інтерактивних функцій для інтерфейсу користувача при впровадженні технології WebSocket. Ця мова програмування надає можливість створювати динамічні вебпрограми, які обходять потребу повного оновлення сторінки. Крім того, широкий спектр бібліотек і фреймворків, таких як React, Angular або VueJS, покращує робочий процес розробки, пропонуючи повторно використовувані елементи та спрощені підходи до обробки стану програми.

На серверній стороні фреймворк Laravel забезпечує швидку розробку та чистий код. Фреймворк надає готові рішення для роботи з базою даних, створення API, роботою з файловою системою та може легко використовуватись у поєднанні з іншими технологіями, наприклад NodeJS та Redis. Фреймворк включає систему ORM (Object-Relational Model), яка спрощує взаємодію з базою даних. Для побудови бази даних, фреймворк підтримує систему міграцій – внесення змін у базу даних за допомогою скриптів, що додає зберігання структури бази даних на різних етапах розробки [7].

SQL (Structured Query Language) використовується для взаємодії з реляційними базами даних та ефективного керування зберіганням і пошуком даних. Це дозволяє розробникам визначати, маніпулювати та запитувати дані з точністю, забезпечуючи цілісність та узгодженість рівня даних програми [8].

Redis (Remote Dictionary Service) – це сервер баз даних типу ключ-значення з відкритим кодом. Redis – це сервер структур даних. Унікальні особливості сервера Redis стали основною причиною його популярності та того, що він застосовується у багатьох реальних проєктах. Спочатку Redis використовували практично так само, як Memcached, але з розвитком Redis цю систему управління базами даних знайшла застосування й багатьох інших ситуаціях. Зокрема – у реалізаціях механізму видавець/передплатник, завдання потокової обробки даних, системи, де потрібно працювати з чергами [9].

Node.js – це програмне середовище виконання з відкритим вихідним кодом та бібліотека, яка використовується для запуску вебдодатків, написаних на мові JavaScript, поза браузером клієнта. В легкому форматі, Node.js – це середовище, яке дозволяє запускати програми, створені за допомогою JavaScript, поза браузером. Раніше програми на JavaScript, на відміну від інших мов програмування, можна було запускати лише у веббраузерах з відповідним движком. Однак з появою Node.js це змінилося. В основу Node.js був взятий двигун виконання JavaScript, відомий як V8, створений компанією Google для браузера Google Chrome. Це дозволяє виконувати JavaScript-код у різних середовищах, включаючи сервери. Тепер за допомогою Node.js можна писати як клієнтську, так і серверну частину вебдодатків [10].

WebSocket – це просунута технологія, що дозволяє відкрити постійне двонаправлене мережне з'єднання між браузером користувача та сервером. За допомогою його API можливо відправити повідомлення на сервер та отримати відповідь без виконання http-запиту, причому цей процес буде подієво-керованим [11].

Крім WebSocket є й інші способи досягнення двонаправленого з’єднання

з сервером, наприклад технологія Long Polling. Також існують сервіси та бібліотеки, які надають схожі можливості, як і WebSocket, наприклад Socket.IO та Pusher. Проте, вони є надбудовою над іншими протоколами та є окремими сервісами, а отже мають затримку при передачі даних та може виникати залежність від сторонніх сервісів [12].

Етап розробки також включає в себе етап модульного тестування системи під час розробки. Модульне тестування допомагає виявляти помилки під час розробки та вчасно усувати їх.

Заключним етапом є тестування готової системи. Під час тестування можуть бути виявленні проблеми або частини, які необхідно змінити, виходячи з досвіду взаємодії з системою. Радше за все, під час цього етапу буде необхідно повертатись до попереднього етапу для вирішення проблем.

Ці етапи можуть бути не суворо послідовними, а накладатись один на одного або повторюватись, щоб забезпечити кінцевий продукт найвищими стандартами та реалізувати всі поставленні задачі та цілі.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Аналіз вимог

Аналіз вимог це головний етап розробки будь-якої системи. Цей етап визначає основні характеристики, функціональні та нефункціональні вимоги.

У цьому розділі кваліфікаційної роботи наводиться аналіз основних вимог до системи інтернет-консультанта, яка включає в себе: можливість реєстрації сайту, додавання агентів підтримки, чат з користувачами клієнтського сайту.

Чат має вирішувати такі питання:

– можливість вирішення питань з одним або декількома агентами підтримки;
– можливість обмінюватись файлами між агентом та користувачем.

Аналізуючи систему та аналогічні рішення, визначаємо наявність наступних підсистем:

– підсистема зв’язку між клієнтом на агентом;
– підсистема адміністрування;
– підсистема чату.

Можливість реєстрації сайту є дуже важливим пунктом для системи, оскільки з цього починається робота з системою. Система має мати спосіб реєстрації через заповнення відповідної форми, автентифікацію. Дані користувача та сайту повинні зберігатись у базі даних.

Візуальне оформлення має бути простим та зрозумілим. Система має бути такою, щоб нею було просто користуватись людям з будь-якими навичками та досвідом роботи за комп’ютером. Оскільки вона має допомагати користувачам, а не ставати на заваді у комунікації між агентами підтримки та користувачами. В цей самий час, агенти підтримки повинні мати простий і зручний спосіб роботи з запитами користувачів.

На основі перелічених задач отримуємо наступні функціональні вимоги (табл. 2.1).

Таблиця 2.1 – Функціональні вимоги ПЗ

Вимога до ПЗ	Опис вимоги
Передача даних між агентом і користувачем	Повідомлення між агентом підтримки та користувачем і у зворотному напрямку повинні виконуватись без перезавантаження сторінки і з найменшою затримкою
Можливість роботи над декількома зверненнями	Один агент має можливість працювати над декількома зверненнями користувачів одночасно
Передача файлів у системі	Агент повинні мати можливість відправляти файли користувачеві. Файли повинні зберігатись на сервері і у будь-який момент бути доступними для перегляду агентами

Аналізуючи сторонні системи можемо виділити наступні вимоги до інтерфейсу (табл. 2.2).

Таблиця 2.2 – Вимоги до інтерфейсу

Вимога до інтерфейсу	Опис вимоги
Наявність віджета	Власник вебсайту повинен мати можливість інтегрувати чат на свій сайт за допомогою віджету
Мінімалістичність вікна чату	Вікно чату має бути простим і функціональним
Перегляд чатів і їх статус агентом підтримки	Агент підтримки повинен мати можливість бачити свої чати і їх статус (наявність нових повідомлень або інших активностей клієнта)
Простота користування	Система має бути простою і зрозумілою у використанні

2.2 Діаграма прецедентів та діаграма послідовностей

Цільовою аудиторією системи є власники вебсайтів, які прагнуть покращити зв’язок зі своїми користувачами шляхом додавання функції інтернет-консультанта на свій сайт.

Визначивши функціональні вимоги до системи, розробимо діаграму прецедентів для визначення функцій усіх користувачів системи. Діаграма прецедентів дозволяє проводити аналіз та візуалізацію різних ролей у системі та їх взаємодії.

Для розробки діаграми прецедентів визначив які групи користувачів можуть бути у системі:

– кінцевий користувач (користувач) – людина, яка користується якимось вебсайтом та потребує допомоги;
– агент – користувач системи, який має доступ до панелі керування на рівні перегляду чатів та взаємодії з ними та який повинен опрацьовувати звернення користувачів;
– власник – власник вебсайту, який прагне додати функцію консультування у свій продукт.

Виходячи з цього можемо виділити різні функції користувачів та створити діаграму прецедентів (див. рис. 2.1).

Система повинна надавати можливість користувачам, які заходять на сайт клієнтів, звернутись до агентів підтримки для отримання консультації з будь-якого питання. Користувач повинен мати можливість надсилати повідомлення агентам підтримки та отримувати їх відповіді без необхідності перезавантаження сторінки та з мінімальною затримкою.

Агенти підтримки повинні мати доступ до повідомлень лише тих користувачів, які звертаються з сайтів власника. Вони не повинні бачити повідомлення з інших сайтів та не повинні мати можливості відповідати на них.

Власники, при потраплянні на сайт системи, повинні мати можливість

або створити новий обліковий запис для підключення нового сайту, або використати свої імейл та пароль для переходу у панель керування системою.

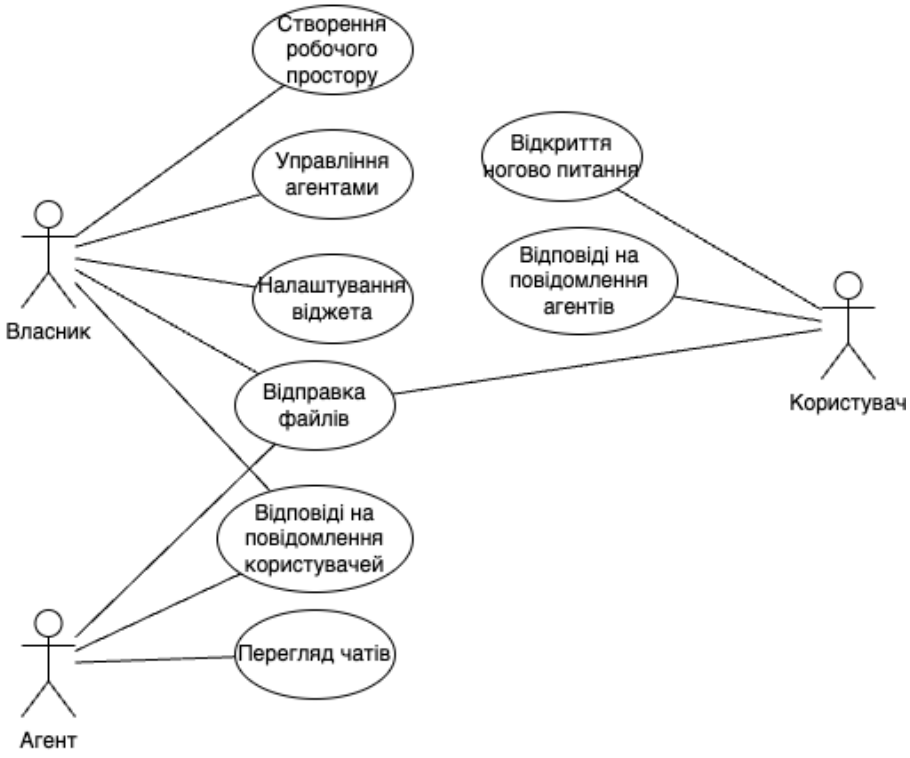

Рисунок 2.1 – Діаграма прецедентів

Після авторизації власник повинен мати можливість отримати код підключення свого сайту до системи. Також власник повинен мати можливість додавання нових агентів підтримки у випадку якщо йому необхідна допомога з наданням консультацій та допомоги користувачам.

Окрім діаграми прецедентів було розроблено діаграму послідовностей дій між об’єктами (рис. 2.2).

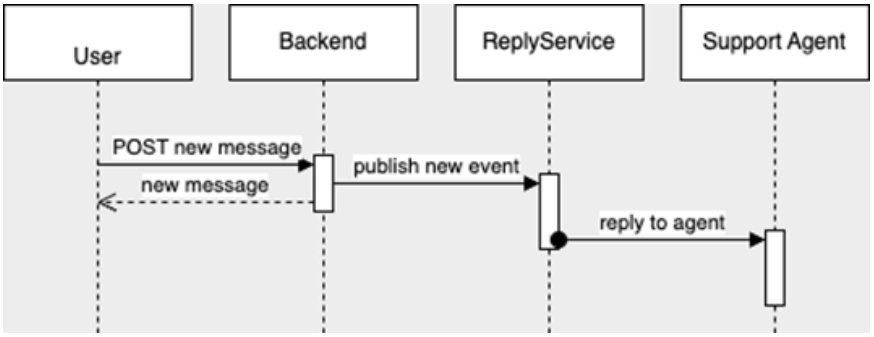

Рисунок 2.2 – Діаграма послідовностей

Діаграма послідовностей має візуально показати процес взаємодії між користувачами сайту та агентами підтримки з моменту натискання кнопки відправки повідомлення і до моменту отримання агентом повідомлення у своїй панелі.

При натисканні кнопки відправки повідомлень, робиться запит за допомогою API до системи. Цей запит містить унікальний ідентифікатор діалогу та саме повідомлення. Отримавши запит, система повинна додати повідомлення у базу даних та передати інформацію агентам підтримки.

Оскільки передати дані користувачу системи у браузер не є тривіальною задачею, оскільки здебільшого інтернет працює за принципом «запит-відповідь». Для досягнення постійної комунікації необхідно використовувати будь-яку систему зв’язку, побудовану за іншим принципом.

Оскільки мова програмування PHP є серверною і орієнтована на додатки по типу «запит-відповідь», необхідно використати NodeJS, який дозволяє реалізувати постійну комунікацію. Так як PHP та NodeJS це дві різні платформи, необхідно налаштувати зв’язок між ними, використавши базу даних Redis, яка є посередником.

За допомогою NodeJS маємо можливість відправляти повідомлення з сервера користувачам не очікуючи на запити зі сторони користувача.

2.3 Діаграма класів та схема бази даних

Для будь-якого сучасного вебдодатку важливою частиною є база даних, відповідно проєктування бази даних і подальше створення структури у обраній системі керування базами даних є одним із важливих етапів розробки. Перед тим, як почати створювати структуру бази даних, необхідно визначитись які дані необхідно зберігати, які зв’язки та тип бази даних необхідно використовувати.

Для проєктування бази даних можливо використати різні підходи, але

найбільш простим для розуміння є візуально схема. Розроблена схема бази даних зображена на рисунку 2.3.

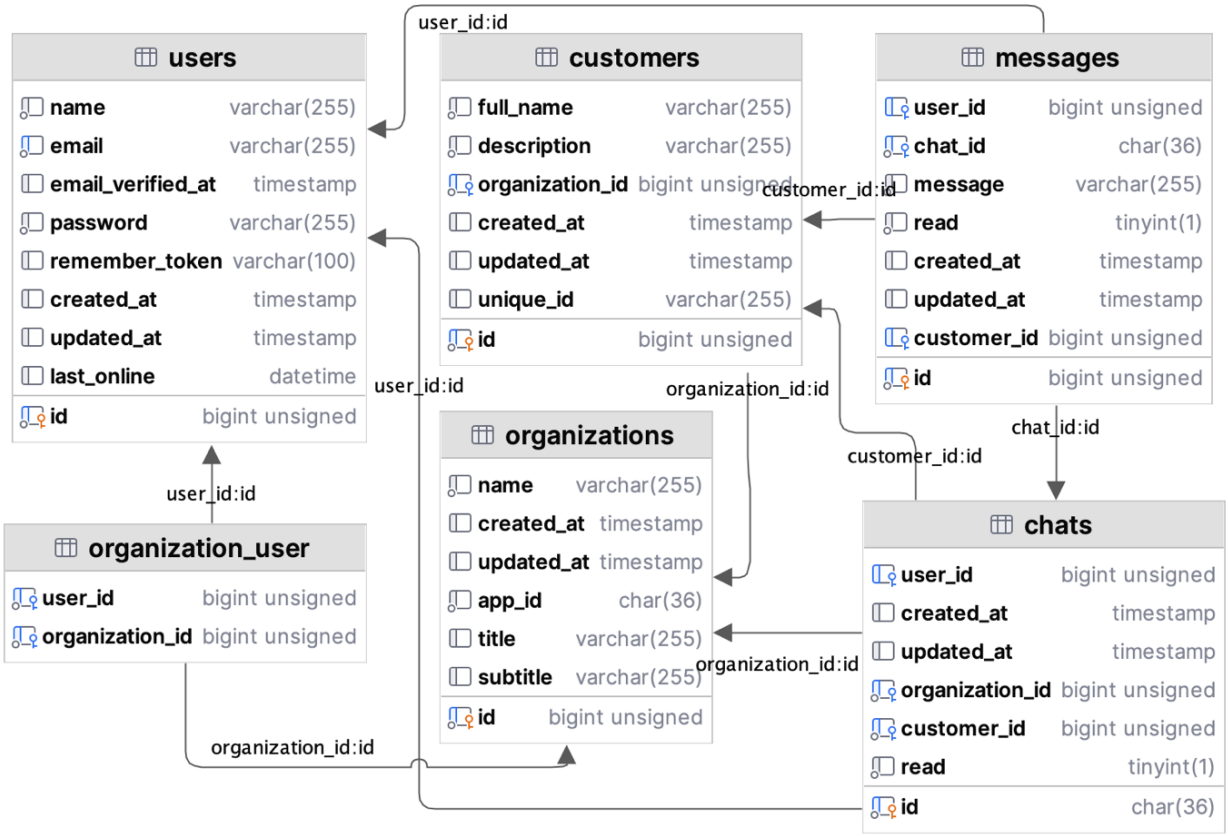

Рисунок 2.3 – Схема бази даних

Для створення бази даних буде використано MySQL 8. Оскільки для реалізації системи інтернет-консультанта необхідно розробити декілька зв’язаних між собою таблиць, то використання NoSQL баз даних не є доцільним. Використання реляційної бази даних дозволяє створити гнучку структуру, а правильно налаштовані зв’язку – цілісність даних.

Всього необхідно декілька ключових таблиць:

– organizations;

– users;

– customers;

– chats;

– messages.

Таблиця organizations є ключовою, оскільки вона створює окрему групу

користувачів, клієнтів та чатів. Це також дозволяє у майбутньому розширити функціонал системи дозволивши працювати агентам у різних організаціях, тим самим надаючи можливість залучення інших людей для підтримки сайту. Таблиця унікальний ідентифікатор організації, назву та опис. Також важливим полем є ідентифікатор додатка, який використовується при роботі з API.

Таблиця users необхідна для збереження інформації про тих користувачів, які можуть мати доступ до панелі керування. Такими користувачами є адміністратори сайтів та їх агенти. Ця таблиця містить такі поля як «name» – повне ім’я користувача, «password» – пароль у хешованому виді, «email» – імейл користувача, «last_online» – дата та час останнього запиту до сервера від імені цього користувача. Також маємо поля «createad_at», «updated_at» та «email_verified_at». Кожне з цих полів має дату та час відповідних даних, а саме: створення користувача, оновлення даних, верифікація імейлу.

Таблиця customers необхідна для збереження інформації стосовно клієнтів, які звертались за допомогою. Таблиця містить поля «full_name» – повне ім’я користувача, яке задає клієнтський сайт або агент підтримки; «description» – опис користувача. Додатково таблиця містить такі службові поля як «id», «created_at», «updated_at», «unique_id».

Таблиця chats необхідна для відображення чату як окремої одиниці у системі. Ця таблиця зв’язує таблиці organizations, users, customers, messages. Вона містить поле «read», яке вказує чи був чат переглянутий агентом підтримки. Також, оскільки всі дії з чатом виконуються за допомогою його ідентифікатора, використання звичайного лічильника було б неправильним з точки зору безпеки даних, оскільки користувачі мали б можливість створювати хибні записи у базі даних підібравши правильний ідентифікатор. Щоб уникнути цього, замість лічильника використовується текстове поле і формат UUID.

Universally Unique Identifier – це 128-бітний ідентифікатор, який призначений для унікальної ідентифікації об'єктів у різних системах без

потреби в централізованому координуванні. UUID складається з 32 шістнадцяткових символів, розділених дефісами на п'ять груп: 8-4-4-4-12. Наприклад, такий ідентифікатор виглядає так: 550e8400-e29b-41d4-a716-446655440000.

До основних переваг UUID можна віднести:

– UUID гарантує унікальність ідентифікатора навіть при його генерації на різних системах в однаковий момент часу (це дозволяє уникнути конфліктів при ідентифікації об'єктів);
– UUID може бути згенерований без потреби в централізованій системі управління ідентифікаторами (це робить їх особливо корисними для розподілених систем);
– деякі типи UUID базуються на криптографічних функціях хешування, що робить їх важкими для передбачення (це може бути корисним для застосувань, де потрібна висока стійкість до атак);
– UUID можна легко створити у багатьох мовах програмування та середовищах, використовуючи вбудовані функції або бібліотеки;
– UUID може бути розширений з використанням різних видів ідентифікаторів, які відповідають певним вимогам системи чи додатка [13].

Останньою таблицею є messages. Записи у цій таблиці є відображенням відправленого повідомлення. Таблиця містить поля «user_id» – ідентифікатор агента підтримки, «customer_id» – ідентифікатор користувача, «message» – саме повідомлення, «read» – чи було повідомлення прочитане.

Крім наведених таблиць є й декілька інших, наприклад «files», яка необхідна для прив’язки відправлених файлів певним повідомленням і чатам. Наведені і описані таблиці є мінімально необхідними для роботи системи інтернет-консультанта, проте за необхідністю можливо розширювати функціонал системи, додаючи нові.

Також слід розглянути системні таблиці, а саме «migrations», «organization_user», «jobs».

Таблиця migrations необхідна для правильного використання міграцій. Міграція – механізм оновлення бази даних у фреймворку Laravel. Цей механізм дозволяє описувати зміни, які необхідно внести у структуру бази даних, за допомогою коду написаного мовою програмування PHP. Додатково механізм міграцій дозволяє зберігати історію змін у структуру бази даних у системах контролю версій. Відповідно, щоб дізнатись які міграції вже було виконано, а які міграції є новими – необхідна окрема таблиця, у якій зберігається назва міграції та черга її виконання.

Таблиця organization_user є сполучною таблицею для таблиць organizations та users. Таблиця organization_user необхідна для реалізації зв’язку «багато-до-багатьох», оскільки структурою бази даних закладена можливість участі одного агента у декількох організаціях.

Таблиця jobs є необхідною для додавання задач у чергу виконання і використовується фреймворком Laravel у фоновому режимі.

Визначившись із структурою бази даних, можемо переходити до проєктування структури класів системи.

Оскільки система складається з двох основних частин – бекенду системи керування та систему обміну повідомленнями, розглянемо їх окремо.

Система обміну повідомленнями є додатком на NodeJS, яка працює із протоколом WebSocket та базою даних для пересилання повідомлень від користувача і агента підтримки іншому учаснику чата. Відповідно до ролі додатка у системі, необхідно спроєктувати класи таким чином, аби спросити подальше реалізацію функціоналу. Для цього треба розробити наступні класи: Client, Message, Chat.

Клас Client є відображенням підключеного клієнта до протоколу WebSocket. Клас повинен мати можливість відправляти дані агенту або користувачу, реагувати на повідомлення від інших учасників чату.

Клас Message є відображенням повідомлення. У цьому класі необхідно мати інформацію та методи для роботи з відправником повідомлення та самим повідомленням.

Клас Chat є відображенням чату і повинен мати методи роботи з учасниками чату та повідомленнями. Необхідно мати можливість додавати нових учасників, видаляти непотрібних та опрацьовувати повідомлення.

Додатково, для спрощення роботи з усіма трьома класами, необхідно розробити клас ChatContainer. Цей клас необхідний для спрощення роботи з чатами і виконує роль бази даних.

Діаграму класів, необхідних для системи обміну повідомленнями системи інтернет-консультанта, наведено на рисунку 2.4.

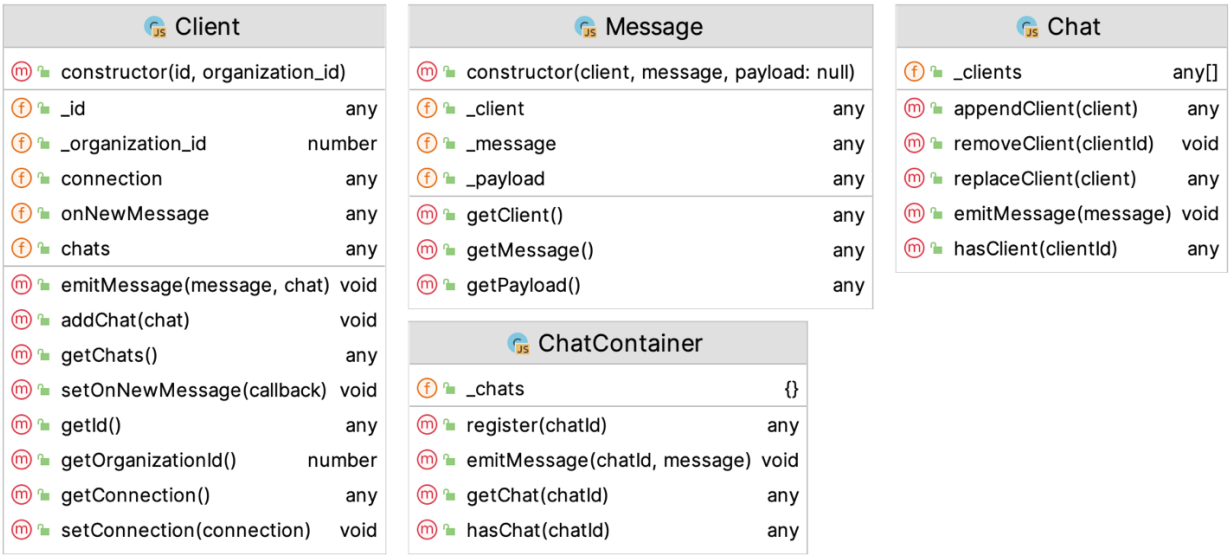

Рисунок 2.4 – Діаграма класів сервісу на NodeJS

Оскільки маємо різні джерела даних, а саме вебсервер та запит від користувача, необхідно передбачити загальний інтерфейс взаємодії з однаковими типами даних, які надходять у різних форматах і у різній структурі. Для цього необхідно розробити певні рівні абстракції і працювати з самими даними, а не з їх структурою. Для цього у системі пересилання повідомлень необхідно розробити такі класи як Message та Chat. Ці класи мають зберігати інформацію про повідомлення та чат. Для простоти майбутньої розробки необхідно закласти методи взаємодії з іншими залежними об’єктами, наприклад з підключеними клієнтами до чату.

Наявність класу ChatContainer зумовлена абстрагуванням чатів та джерел повідомлень, оскільки саме цей клас має відповідати за створення чатів (як об’єкта у системі) та за переправлення вхідних повідомлень необхідним чатам.

Іншою важливою системою є система керування. Ця система виконує роль панелі керування системою та API для роботи з системою. Діаграма класів системи керування наведено на рисунку 2.5.

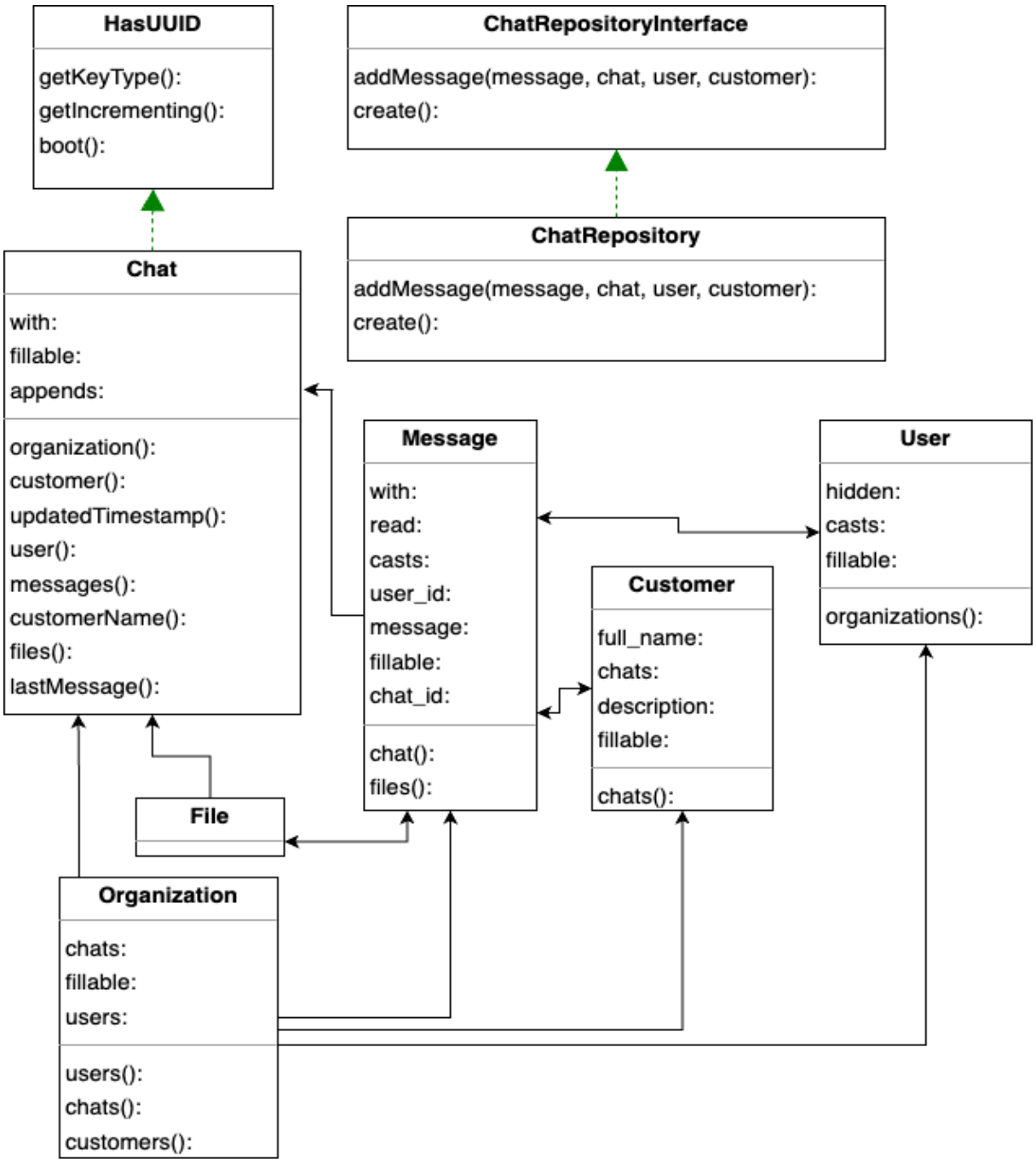

Рисунок 2.5 – Діаграма класів серверної частини системи

Розглянемо систему керування детальніше.

У системі передбачається використання шаблону проєктування репозиторій для роботи з чатами, тому необхідно розробити інтерфейс репозиторію та саму реалізацію. Репозиторій повинен надавати можливість додавати нові повідомлення до існуючого чата (addMessage) та створювати нові чати (create).

Клас HasUUID необхідний для реалізації формату UUID та заміни лічильника на ідентифікатор у цьому форматі. Цей клас є типом Trait і використовується класом Chat.

Класи Organization, Chat, Message, User, File та Customer є класами, які відображають відповідні таблиці бази даних. Ці класи є частиною ORM.

Під час розробки системи буде задіяно значно більше класів, оскільки фреймворк Laravel містить дуже багато сторонніх рішень і повністю написаний у об’єктно-орієнтовному стилі.

2.4 Діаграма компонентів та діаграма розгортання

Оскільки маємо декілька сервісів, то постає питання у взаємодії цих сервісів.

Враховуючи що реалізація протоколу WebSocket працюватиме у окремому додатку, виникає проблема з прослуховуванням TCP порту. Два різні додатки не можуть прослуховувати один і той самий порт, тому необхідно використовувати проксування запитів. Для вирішення цієї задачі підходить вебсервер NGINX, який дозволяє створити необхідне налаштування [14].

При використанні проксування необхідно розділити різні додатки за адресами (доменними іменами) або відносними шляхами запиту. Було обрано розділення додатків за доменними іменами, оскільки це є найбільш простішим для розуміння і архітектури додатків.

Враховуючи все це маємо наступні задіяні адреси (див. табл. 2.3).

Таблиця 2.3 – Використані адреси

Адреса	Призначення
websocket.helpdesk.test	Сервіс обміну повідомленнями
app.helpdesk.test	Сервіс панелі керування і чату

Враховуючі задіяні сервіси можемо побудувати схему (див. рис. 2.6).

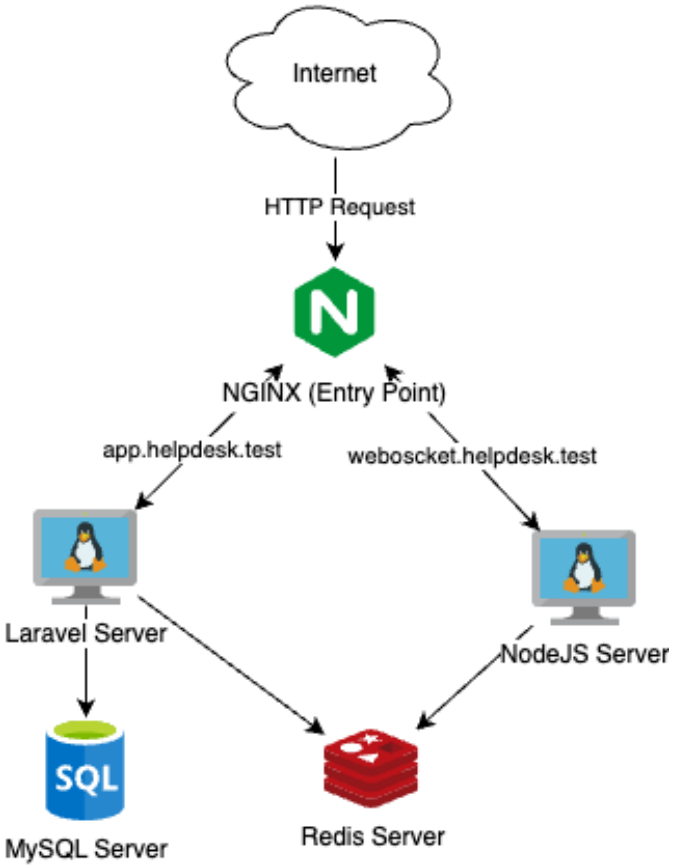

Рисунок 2.6 – Схема задіяних сервісів

Всі сервіси можуть бути розміщені на окремих серверах, що дає можливість налаштувати гнучкий проєкт у хмарі і досягти максимальної стійкості та надійності системи. Як видно зі схеми сервісів, з базою даних взаємодіє лише сервер Laravel. Сервер взаємодії з клієнтами (NodeJS Server) не використовую базу даних, а отримує повідомлення від іншого сервісу за допомогою бази даних Redis.

Система може бути розгорнута як на одному сервері, так і на різних

серверах. Для наглядного відображення розміщення частин на різних серверах існують схеми розгортання.

Розробка схеми розгортання є критичним етапом у будь-якому проєкті програмного забезпечення чи інформаційної системи. Перш за все, ця схема надає загальний огляд та розуміння того, як програмне забезпечення чи система будуть встановлюватися та використовуватися. Це дозволяє команді проєкту зрозуміти потрібні кроки для успішного впровадження та уникнути можливих проблем.

Другий аспект полягає в тому, що схема розгортання служить важливим комунікаційним інструментом між різними членами команди проєкту, включаючи розробників, тестувальників та адміністраторів систем. Вона допомагає уточнити ролі та відповідальності кожного учасника процесу розгортання, а також забезпечує чітке розуміння послідовності дій.

На рисунку 2.7 зображена схема розгортання на одному фізичному сервері.

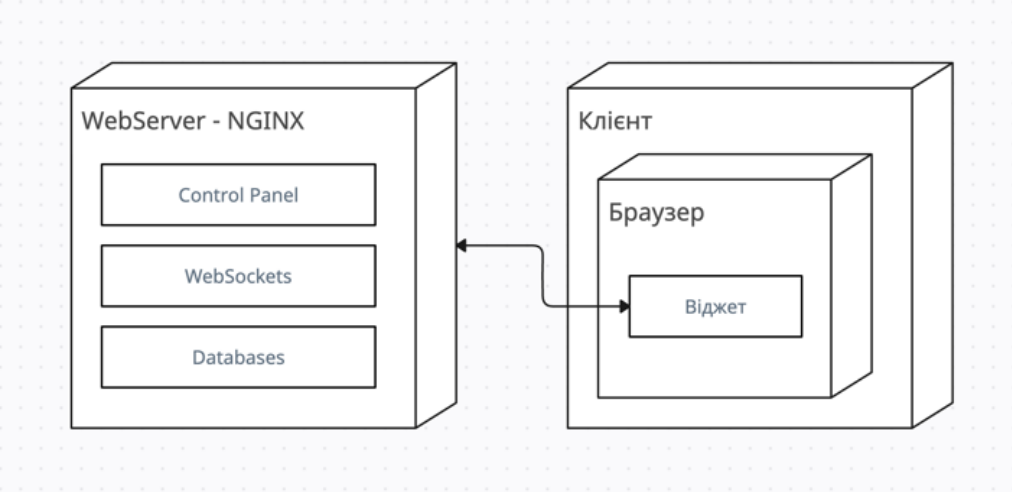

Рисунок 2.7 – Діаграма розгортання на одному сервері

Оскільки у проєкті передбачається модульна структура, схема розгортання на різних проєктах та копіях програмного засобу можуть сильно відрізнятись один від одного, оскільки кінцевий користувач може як додати нові сервіси та доповнити код, так і прибрати або замінити якісь компоненти

системи, що прямо вплине на кількість та типи задіяних сервісів і змінить схему розгортання.

Побудувавши всі необхідні вимоги, UML діаграми та схеми класів, можемо перейти до розробки програмної реалізації системи.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація системи інтернет-консультанта

Для розробки системи «Онлайн-консультант» було обрано наступні технології:

- Laravel у якості фреймворку для реалізації backend і панелі керування підключеними сайтами;
- VueJS е якості фреймворку для побудови фронтенду;
- NodeJS для реалізації протоколу WebSocket;
- MySQL як основну базу даних для зберігання історії звернень, користувачів та налаштувань сайтів;
- Redis як базу даних для передачі повідомлень між Laravel та NodeJS.

Обраний стек технологій є дуже гнучким та дозволяє створювати додатки будь-якої складності, проте він також і вносить певні обмеження.

Фреймворк Laravel написаний на мові програмування PHP і працює за клієнт-серверною архітектурою. Він дозволяє опрацьовувати HTTP запити, але не може підтримувати асинхронні з’єднання, що є необхідним для побудови системи. Додавання асинхронної роботи і реалізації WebSocket на мові програмування PHP є можливим, проте неоптимальним з точку зору швидкості роботи.

Таким чином було обрано мову програмування NodeJS для побудови асинхронної частини системи. NodeJS підтримує асинхронну роботу без будь-яких додаткових налаштувань, тому реалізація WebSocket на ньому є найбільш доцільною.

Оскільки маємо дві різні мови програмування і дві різні частини системи, то виникає питання для поєднання їх у одне ціле. Для цих цілей використано базу даних Redis та її особливість Pub/Sub [15], коли існує можливість створювати події у одному додатку, та реагувати на них у іншому.

Система інтернет-консультанта складається з трьох основних частин: панель керування, сервіс пересилання повідомлень та клієнтський віджет.

Для розробки панелі керування обрано мову програмування PHP та фреймворк Laravel. Для створення клієнтської частини панелі керування обрано фреймворк VueJS, оскільки він є дуже простим у використанні та швидким.

Клієнтський віджет розроблено з використанням фреймворку VueJS, оскільки це дозволяє використовувати повторно існуючий код панелі керування.

Сервіс пересилання повідомлень розроблено з використанням мови програмування NodeJS.

Оскільки панель керування і клієнтський віджет мають однакову кодову базу, але різні способи виконання коду, у проєкті використано різні підходи для побудови окремих модулів. Так для побудови панелі керування використовується сучасну систему збірки Vite, тоді як для віджета – Webpack. Це дозволяє певною мірою розділити код, але і додає додаткові кроки під час побудови всього користувацького інтерфейсу.

Як вже згадувалось у розділі проєктування, побудова структури бази даних відбувається за допомогою міграцій. Міграціями у фреймворку Laravel є окремі класи, написані мовою програмування PHP, які описуються які дії зі структурою бази даних необхідно виконати при виконанні міграція, та які при відкаті міграції. Ці два методи допомагають розробити структуру бази даних таким чином, що у будь-який момент часу можна відновити певний стан структури бази до необхідної версії. Приклад міграції зображено на рисунку 3.1.

Як видно з прикладу міграції, необхідні поля та їх налаштування вказуються кодом за допомогою виклику певних методів. За допомогою міграцій можемо створити базу даних не використовуючи жодної команди мови SQL, що надає дуже суттєву перевагу, оскільки відсутній зв’язок з певним рушієм бази даних та типом самої БД.

```php
Schema::create( table: 'messages', function (Blueprint $table) {
    $table->id();

    $table->foreignIdFor( model: \App\Models\User::class,  column: 'user_id')
        ->nullable()
        ->constrained();
    $table->foreignIdFor( model: \App\Models\Chat::class,  column: 'chat_id')
        ->constrained()
        ->onDelete( action: 'cascade');

    $table->string( column: 'message');
    $table->boolean( column: 'read')->default( value: false);

    $table->timestamps();
});
```

Рисунок 3.1 – Міграція бази даних для створення таблиці повідомлень

Всю роботи з перетворення викликів методів класів на SQL команди виконує драйвер бази даних. Так само працюємо й з базою даних для виконання якихось дій – створення, видалення чи зміни даних. Це дозволяє уникнути будь-якої залежності від технологій бази даних та обмежень і пришвидшити процес написання коду. При роботі з базою даних у фреймворку Laravel, програміст абстрагується від бази даних чи інших сервісів і працює з даними переважно як зі звичайними об’єктами (рис. 3.2).

```
AlexeyRudkovskiy
public function create(): Chat
{
    $chat = new Chat();
    $chat->save();

    return $chat;
}
```

Рисунок 3.2 – Створення нового чату у базі даних

Іншою важливою частиною фреймворку Laravel є використання Middleware. Middleware – спеціальні класи, які виконуються перед виконанням самого коду додатка та після початку обробки. За допомогою Middleware

можливо як налаштувати певні права доступу до якихось частин системи, так і виконувати інші дії, наприклад виконувати логування запитів чи оновлювати системну інформацію.

За допомогою Middleware у системі розроблено функціонал відслідковування статусу агентів. При кожному запиті до API, які виконуються агентами, автоматично оновлюється відповідне поле у базі даних, яке вказує дату та час останнього запиту. Далі, там де необхідно отримати активних агентів, можемо відфільтрувати агентів за датою (див. рис. 3.3 та 3.4).

```
AlexeyRudkovskiy
public function handle(Request $request, Closure $next): Response
{
    $user = auth()->user();
    $user->last_online = now();
    $user->save();

    return $next($request);
}
```

Рисунок 3.3 – Оновлення дати та часу останнього запиту

```
no usages  new *
public function apply(Builder $builder, Model $model)
{
    $builder->where( column: 'last_online',  operator: '>', now()->subMinutes( value: 30));
}
```

Рисунок 3.4 – Фільтрація користувачів за датою останнього запиту

Для побудови користувацького інтерфейсу у системі інтернет-консультанта використано фреймворк VueJS, який збирається за допомогою системи збірки Vite.

Додаток на VueJS побудовано з використанням віртуального роутера, тобто за принципом SPA. Для панелі керування розроблено спеціальний підхід з шаблонами, який дозволяє концентруватись на написанні коду сторінок, а не на загальній структурі. Існує два основних шаблона – Dashboard та Empty.

Шаблон Dashboard завантажується автоматично маршрутизатором і містить такі елементи інтерфейсу користувача як основне навігаційне меню, загальна розмітка сторінки та інформація про користувача. Шаблон Empty містить лише заготовку для відображення сторінок авторизації та реєстрації.

Для прискорення розробки було використано бібліотеку стилів Tailwind. Вона дозволяє зосередитись на самих компонентах, а не на написанні стилів до них. Стилі додаються шляхом додавання класів до тегів. Процес додавання стилів до компонента можна побачити на рисунку 3.5.

```
<template>
    <div class="rounded-xl shadow bg-white">

        <div class="rounded-t-xl px-6 py-6 bg-gray-50 border-0 border-b-2 border-gray-100" v-if="!noHeader">
            <div class="text-2xl text-bold ">{{ title }}</div>
            <div class="mt-2 text-gray-600">{{ description }}</div>
        </div>

        <div :class="contentClasses" v-if="!noContentStyles">
            <slot></slot>
        </div>
        <slot v-else></slot>

        <div class="flex w-full rounded-b-xl bg-gray-50 border-0 border-t-2 border-gray-100 p-6" v-if="!noFooter">
            <slot name="footer"></slot>
            <span v-if="!$slots.footer">{{ footer }}</span>
        </div>

    </div>
</template>
```

Рисунок 3.5 – Приклад додавання стилів до компонента

Важливою частиною панелі керування є підключення до сервісу пересилання повідомлень. Для цього одразу після завантаження усіх скриптів та після початку побудови інтерфейсу користувача виконується підключення за допомогою протоколу WebSocket. Підключення зберігається протягом всього знаходження користувача на сторінці і доступно з будь-якої сторінки чи файлу. Одразу після підключення до сервера, додаємо обробник нових повідомлень, який створює глобальні події при надходженні нових повідомлень від сервісу. Весь процес підключення та додавання обробника повідомлень зображені на рисунку 3.6.

```
const socket : WebSocket  = new WebSocket( url: 'wss://ws.helpster.pics');
 AlexeyRudkovskiy
socket.addEventListener( type: 'open',  listener: function () : void  {
  socket.addEventListener( type: 'message',  listener: (e : MessageEvent<any> ) : void  => {
    let data = e.data;
    data = JSON.parse(data);
    const { event, payload } = data;

    emitter.emit( event: 'socket:event', event, payload);
  });
});


globalState.socket = socket;
```

Рисунок 3.6 – Підключення до WebSocket

Під час створення шаблона Dashboard, який є основним для всієї панелі керування, виконується додатковий етап підключення – реєстрація користувача. Для реєстрації необхідно відправити повідомлення у систему за допомогою WebSocket, у якому необхідно вказати ідентифікатор користувача та ідентифікатор організації (рис. 3.7).

```
emit( event: 'register',  payload: {
  id: 'agent_' + authorizedUser.id,
  organization_id: user.value.organization_id
});
```

Рисунок 3.7 – Реєстрація користувача у сервісі обміну повідомлень

Останнім етапом підключення до сервісу є реєстрація користувача у певному чаті. Для цього на сторінці чата при виборі якогось чату необхідно відправити нове повідомлення сервісу, у якому достатньо вказати ідентифікатор чата. Однак, якщо виконується перехід до іншого чату, необхідно також відправити повідомлення щоб покинути його, оскільки якщо не зробити цього – агент буде отримувати хибні повідомлення. Процес підключення до чату зображено на рисунку 3.8.

```
axios.get( url: `/api/chats/${chat.id}`)
  .then(response => response.data)
  .then(response => {
    chat.read = true;
    selectedChat.value = chat
    activeChat.value = response
    container.init(response.messages)
    groups.value = container.getGroups()

    customerName.value = response.customer.full_name;

    emit( event: 'join',  payload: { chatId: response.id })

    setTimeout( handler: () => {
      const scrollHeight = messagesContainer.value.scrollHeight + 100;
      messagesContainer.value.scrollTo(0, scrollHeight);
    },  timeout: 25);

    if (details.value !== null) {
      showDetails()
    }
  })
```

Рисунок 3.8 – Підключення до чата

Після підключення до чата і у разі додавання нового повідомлення, буде створено нову глобальну подію, відреагувавши на яку можемо підтримувати історію листування у актуальному стані.

Окрім повідомлень з відкритого чату агент підтримки також отримує всі повідомлення у рамках своєї організації. Це необхідно для оновлення інших чатів, а саме стану чата (чи є нові повідомлення) та останнього повідомлення.

Процес додавання нового повідомлення до чату дуже просте. Необхідно відправити HTTP запит по API і передати ідентифікатор чата та саме повідомлення. Сервер панелі керування, отримавши запит, створює новий запис у базі даних та систему подію (див. рис. 3.9).

На сервері панелі керування існує окрема черга виконання завдань, яка працює у фоновому режимі і не затримує обробку HTTP запитів. У системі інтернет-консультанта маємо одну системну задачу, а саме відправка нового повідомлення.

```
AlexeyRudkovskiy
public function addMessage(Chat $chat, Request $request)
{
    $customer = $chat->customer;

    $message = $this->chatRepository->addMessage($request->get( key: 'message'), $chat,  user: null, $customer);
    NewMessage::dispatch($message);

    return $message;
}
```

Рисунок 3.9 – Додавання нового повідомлення до чата

Фреймворк Laravel має дуже гнучку систему подій і дозволяє автоматично перенаправляти події по різним каналам. Фреймворк здатен перенаправляти події у різні системи, наприклад Pusher або Redis. Це дозволяє реалізувати реагування на події у інших додатках, а не лише всередині самого фреймворку. Саме цей принцип і використано у системі, а саме пересилка подій з фреймворку у Redis і подальша їх обробка сервером NodeJS.

Оскільки процес відправки повідомлення у систему обробки повідомлень може займати багато часу, задля підвищення швидкості і вірогідності успішної відправки повідомлень, доцільним є використання черги системних завдань для відправки повідомлень.

Цей функціонал вже є вбудованим у фреймворк, тому не потребує ніяких додаткових налаштувань. Процес додавання нової події у такому підході є наступним: фреймворк додає нову задачу у чергу, а окремий обробник черги отримує нове завдання та виконує код додавання нової події. Цей підхід є найбезпечнішим, оскільки можемо повторювати та відслідковувати невдалі спроби, приховуючи це від користувача та не впливаючи на обробку самого HTTP запиту.

Код події нового повідомлення зображено на рисунку 3.10.

На рисунку 3.11 зображено процес обробки системної черги задач.

```php
AlexeyRudkovskiy
class NewMessage implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public int $organization_id = -1;

    /**
     * Create a new event instance.
     */
    AlexeyRudkovskiy
    public function __construct(public Message $message)
    {
        $this->organization_id = $this->message->chat->organization_id;
    }

    /** Get the channels the event should broadcast on. ...*/
    no usages  AlexeyRudkovskiy
    public function broadcastOn(): array{...}
}
```

Рисунок 3.10 – Код події нового повідомлення

```
alexey@alexeys-MacBook-Pro ws-backend.test % php artisan queue:work

  INFO  Processing jobs from the [default] queue.

  2024-03-05 21:29:02 App\Events\NewMessage ..... RUNNING
  2024-03-05 21:29:02 App\Events\NewMessage ..... 36.96ms DONE
```

Рисунок 3.11 – Обробка системної черги задач

3.2 Тестування роботи системи інтернет-консультант

В сучасному інформаційному середовищі тестування програмного забезпечення визнається невід'ємною складовою кожного процесу розробки програм. Незважаючи на те, що цей етап часто відображається як завершальний у циклі розробки, його важливість розповсюджується на всі стадії роботи над продуктом. Від тестування залежить якість програмного продукту, його надійність та безпека.

Тестування дозволяє виявити та усунути помилки на ранніх етапах розробки, що значно зменшує ризики появи проблем у роботі програми після випуску. Це сприяє покращенню якості продукту і позитивному сприйняттю його користувачами. Окрім цього, тестування допомагає виявити потенційні ризики та вразливості програмного забезпечення, що є важливим для забезпечення безпеки та захисту конфіденційності даних користувачів.

Ще однією важливою складовою тестування є оцінка продуктивності програми. Тестування продуктивності дозволяє переконатися, що програмне забезпечення працює ефективно під різними умовами і зберігає високу продуктивність навіть при великому навантаженні. Це дозволяє забезпечити зручне та безперебійне використання програми для кінцевих користувачів.

Тестування також спрямоване на покращення зручності використання програмного забезпечення. Інтерфейс користувача та загальний досвід взаємодії з програмою грають важливу роль у задоволенні потреб користувачів. Тому виявлення та виправлення проблем зручності використання на етапі тестування є критично важливим для успіху програмного продукту.

У підсумку, тестування програмного забезпечення є невід'ємною складовою розробки програм та гарантує їх високу якість, надійність та безпеку. Важливо вкласти достатньо уваги та ресурсів у цей етап процесу розробки, щоб забезпечити успішне впровадження програми та задоволення потреб користувачів.

Використання тестування на всіх етапах розробки програмного забезпечення дозволяє запобігти появі помилок та вчасно виявити вже існуючі помилки. Аби не виконувати постійно тестування вручну і не перевіряти безліч можливих варіантів роботи програмного засобу, існує автоматичне тестування.

При написанні коду мовою програмування PHP, можемо підключити PHPUnit та описувати необхідні сценарії тестування кодом.

PHPUnit – це популярний інструмент для автоматизованого тестування

в PHP. Він призначений для написання і виконання тестів для PHP-коду, забезпечуючи високу якість та надійність програм. PHPUnit надає розширений набір функцій для розробки тестів, включаючи підтримку юніт-тестування, функціональне тестування та тестування інтеграції.

Одним із ключових переваг PHPUnit є його здатність виконувати тести автоматично, що дозволяє швидко та ефективно перевіряти функціональність програмного забезпечення при внесенні змін. Це особливо корисно в контексті неперервної інтеграції та неперервної поставки, де автоматизоване тестування виконується на кожній ітерації розробки, щоб впевнитися у стабільності програми.

PHPUnit дозволяє створювати тести для перевірки різних аспектів програмного забезпечення, включаючи перевірку правильності поведінки окремих методів та класів, перевірку відповідності вихідних даних очікуваним результатам, а також тестування взаємодії з базою даних та іншими зовнішніми сервісами. Це дозволяє забезпечити повноту та коректність функціональності програми під час розробки та після внесення змін.

Існує спеціальний підхід до розробки програмного забезпечення, який ставить тестування на перше місце – розробка через тестування. Суть підходу полягає у тому, що спочатку пишеться тест, а вже потім реалізацію необхідних класів та функціоналу.

У кваліфікаційній роботі також використовується автоматичне тестування за допомогою PHPUnit. Оглянемо декілька сценаріїв тестування.

Першим сценарієм є перевірка роботоспроможності авторизації. У таблиця 3.1 наведено необхідні кроки сценарію.

Таблиця 3.1 – Тесткейс «Авторизація існуючого користувача»

№	**Крок**	**Очікуваний результат**
1	Створити організацію	Поява нової організації у базі даних
2	Створити користувача	Поява нового користувача у базі даних
3	Відправити HTTP запит	Користувач успішно авторизований

Окремо необхідно протестувати валідацію даних, тому додамо тест перевірки правильності вказаних імейлу та пароля (табл. 3.2).

Таблиця 3.2 – Тесткейс «Перевірка валідації»

№	**Крок**	**Очікуваний результат**
1	Створити організацію	Поява нової організації у базі даних
2	Створити користувача	Поява нового користувача у базі даних
3	Відправити HTTP запит з хибними даними	Користувач не авторизований

Також необхідно перевірити можливість реєстрації (див. табл. 3.3).

Таблиця 3.3 – Тесткейс «Реєстрація»

№	**Крок**	**Очікуваний результат**
1	Відправити HTTP запит з коректними даними	Організація успішно створена
2	Перевірка статусу відповіді	Отримати правильну відповідь відповідно до стандарту REST
3	Перевірка наявності користувача в базі даних	База даних містить нового користувача
4	Перевірка наявності організації в базі даних	База даних містить нову організацію

Інші компоненти системи, а саме робота з чатами та повідомленнями, протестована аналогічним способом.

Повний код тестування модуля авторизації наведено у Додатку А.

Використання автоматичного тестування дозволяє запобігти появі нових помилок у коді та впевнитись у працездатності вже існуючого коду, оскільки при змінах коду тести можуть перестати виконуватись успішно.

Важливим зауваженням при написанні автоматичних тестів є необхідність розробки якомога більше сценаріїв тестування з різними наборами даними, включаючи такі, які не будуть використовуватись у системі або призведуть до передбачуваних помилок, наприклад тестування валідації та вхідних параметрів функцій і методів класів.

Незважаючи на важливу роль у використанні автоматичного тестування, слід не забувати про інші види тестування – перевірка працездатності вручну, інтеграційні тести, оскільки навіть при успішному проходженні модульних тестів, можлива помилка у не протестованому фрагменті коду.

3.3 Огляд готової системи

Розроблена система складається з двох основних частин – панелі керування та клієнтського віджета.

Будь-який користувач системи, який хоче підключити систему інтернет-консультанта на свій сайт, починає роботу з системою з реєстрації або авторизації, якщо він вже має свій профіль (рис. 3.12).

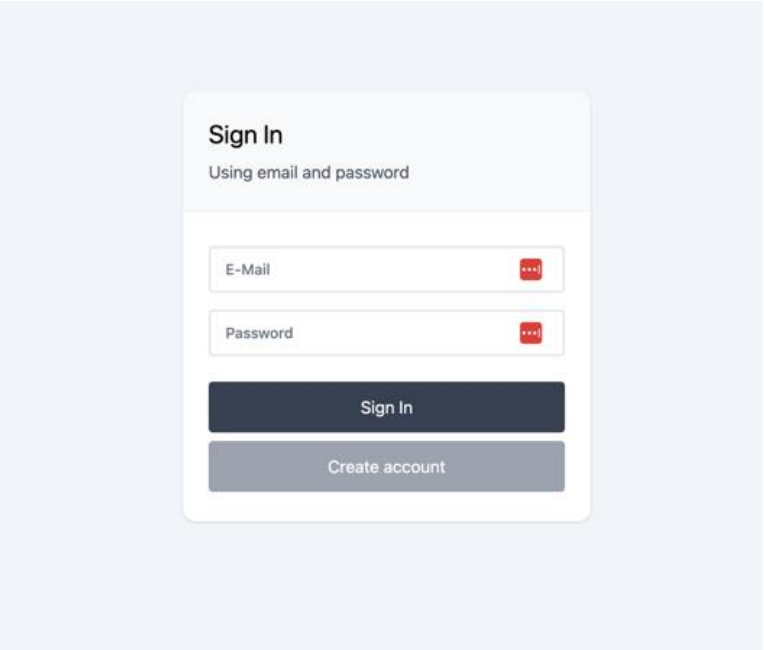

Рисунок 3.12 – Сторінка авторизації

Після авторизації або реєстрації, адміністратор потрапляє до панелі керування (рис. 3.13). Тут він може побачити статистику по своїй організації, а саме кількість агентів у системі, кількість агентів онлайн та кількість відкритих чатів. Чат вважається відкритим, якщо існує хоча б одне непереглянуте повідомлення від клієнта.

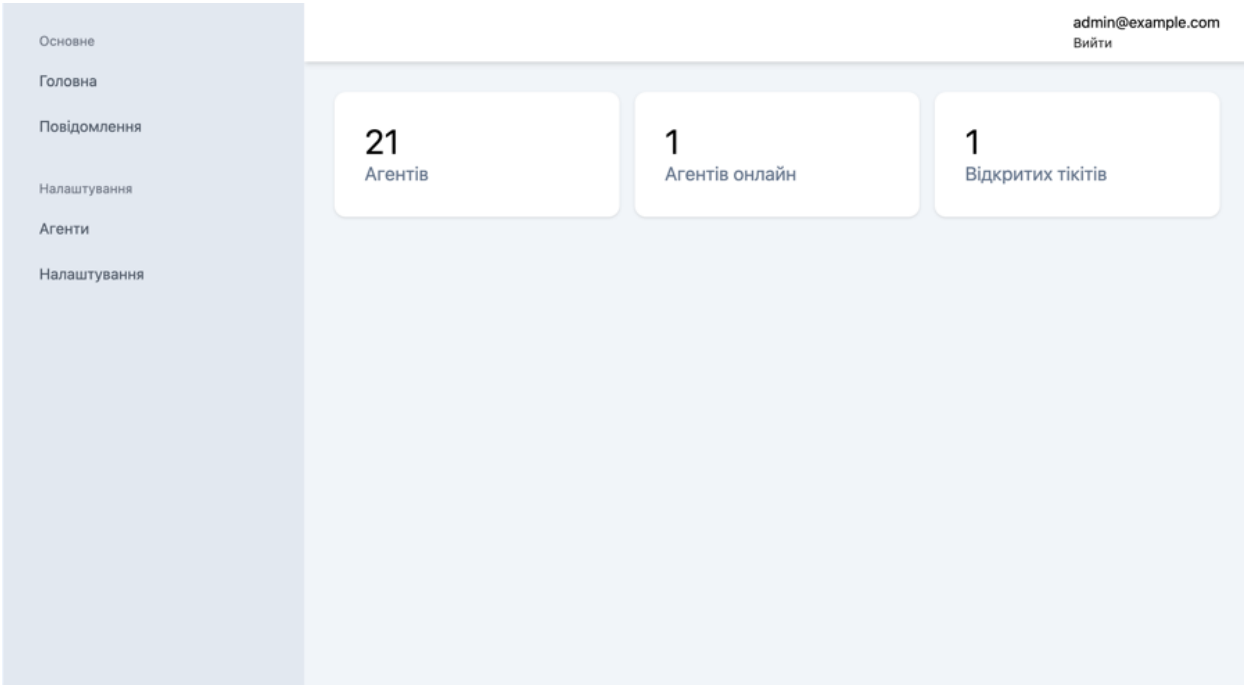

Рисунок 3.13 – Головна панелі керування

Оскільки система розповсюджується з відкритим кодом, будь хто може налаштувати власну копію системи та додати нові елементи як на головну сторінку, так і на будь-яку іншу чи навіть створити нові сторінки системи.

Важливою сторінкою для підключення сайтів є сторінка налаштувань (див. рис. 3.14). На цій сторінці можна побачити інструкцію для підключення сайту та код підключення.

На сторінці налаштувань код підключення є стандартним і може використовуватись будь-яким сайтом, але він все одно потребує змін. Так, власник сайту повинен змінити об’єкт params, оскільки він задає інформацію про клієнта. Цей об’єкт можливо повністю видалити, тоді запит підтримки буде анонімним.

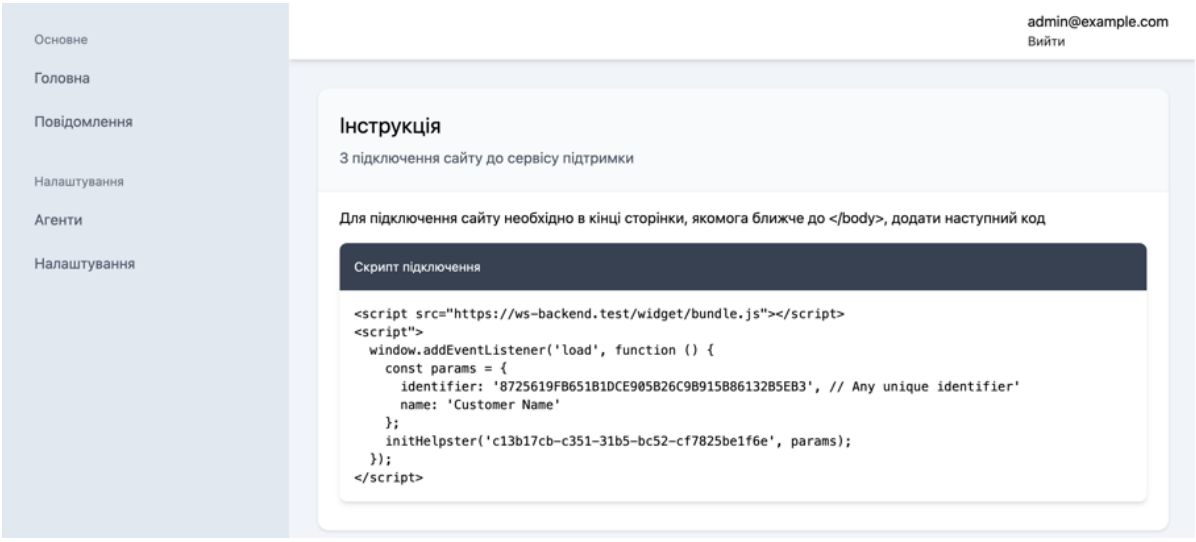

Рисунок 3.14 – Сторінка налаштування підключення

Після реєстрації організації, адміністратор може додати нових агентів, перейшовши на сторінку «Агенти» (рис. 3.15). Після додавання агента, він буде автоматично доданий до поточної організації та зможе авторизуватись за допомогою вказаних імейла та пароля.

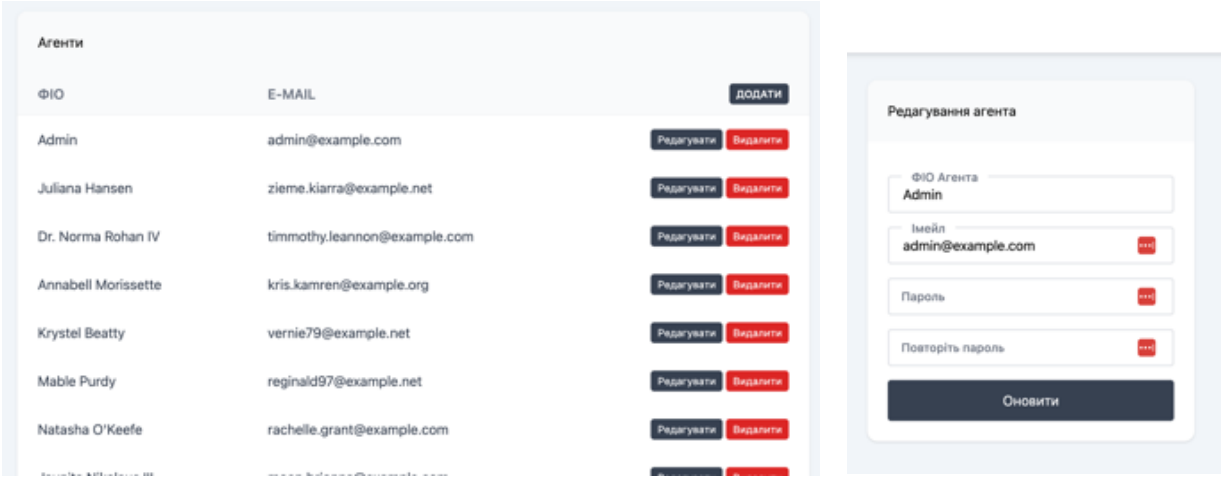

Рисунок 3.15 – Сторінки агентів та форма редагування окремого агента

Організація може мати необмежену кількість агентів підтримки, система не містить обмежень.

Після реєстрації кабінету, налаштування агентів (за потреби), власник сайту може додати віджет на свій сайт. Для тестування віджету було розроблено окрему сторінку (див. рис. 3.16).

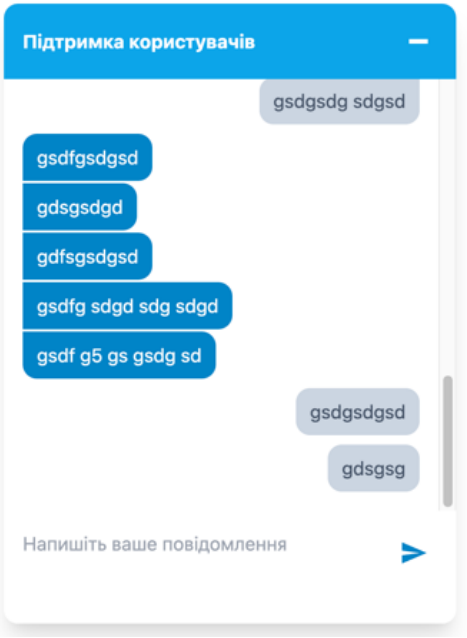

Рисунок 3.16 – Сторінка демонстрації роботи віджета

На цій сторінці можна протестувати роботу системи, оскільки можна одразу перевірити дві ролі – агента та клієнта.

Завдяки використанню протоколу WebSocket, повідомлення, надіслані агентом або клієнтом, потрапляють до співрозмовника майже миттєво.

Сторінка агента підтримки має звичайний вид і містить такі елементи: список чатів, повідомлення обраного чату, додаткова інформація по чату.

Кожен чат на сторінці чатів відображається як окремий рядок у лівій колонці. Кожен чат має наступну інформацію: ім’я клієнта, останнє відправлене повідомлення та статус чата. Чат може бути прочитаним або відкритим. Відкритий чат позначається трохи синім кольором на фоні і кружечком синього кольору.

Натиснувши на чат, історія листування буде завантажена і відображена у правій колонці. Крім повідомлень, область листування містить ім’я клієнта зверху та область для написання нового повідомлення знизу.

Також існує область з інформацією про чат, яка за замовченням прихована, але може бути показана на екрані натисканням відповідної кнопки. Ця область містить таку корисну інформацію як надіслані файли у цьому чаті та попередні звернення клієнта. Попередні звернення містять початкове

повідомлення та ім’я агента, який першим взяв на себе це листування. У області додаткової інформації агент може змінити ім’я клієнта, наприклад якщо запит був анонімним, та у процесі листування ім’я користувача стало відомим. Зовнішній вид сторінки чатів зображено на рисунку 3.17.

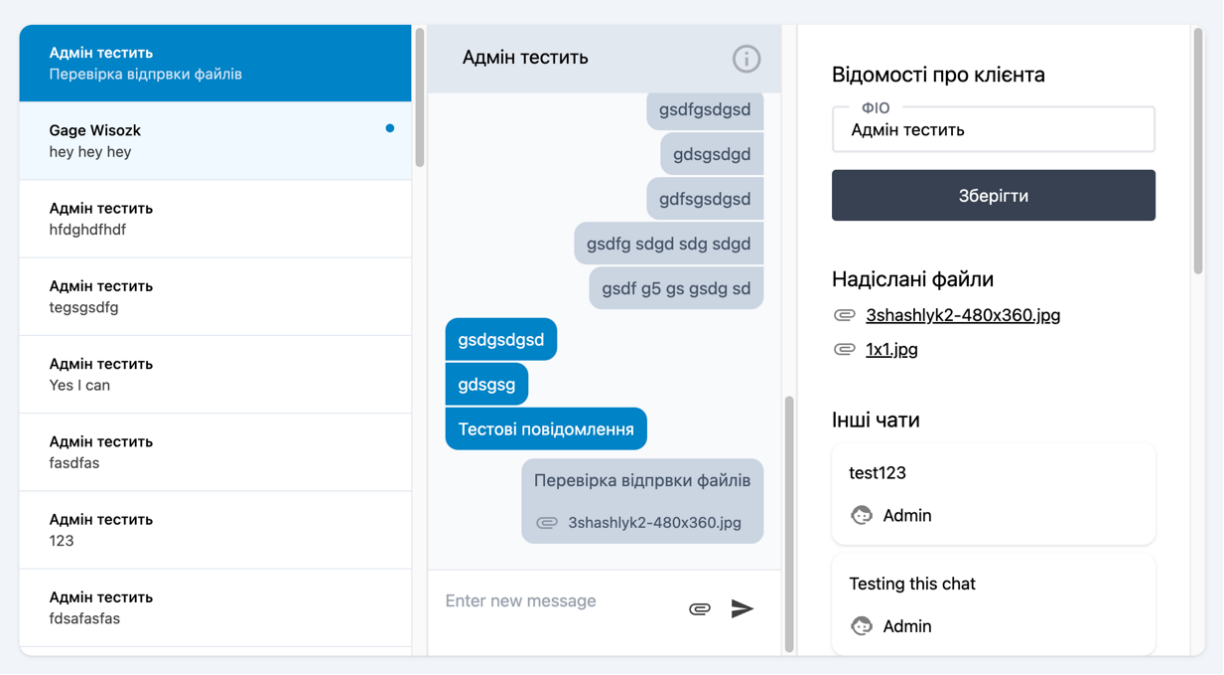

Рисунок 3.17 – Зовнішній вид сторінки чатів

Інтерфейс користування у розробленій системі є дуже простим та зрозумілим, а використання звичних елементів у порівнянні з іншими системами робить її вкрай простою для розуміння. Використання сучасних інструментів та технологій дозволило побудувати сучасну і швидку системи, яка може конкурувати з іншими аналогічними системами. Крім цього, розроблена система має ряд переваг, наприклад відкритий код та можливість запуску власної копії системи на власному сервері.

ВИСНОВКИ

У ходні виконання дипломного проєкту отримано успішну реалізацію системи інтернет-консультанта, розробленої мовою програмування PHP, JavaScript, бази даних MySQL, фреймворків Laravel, VueJS та TailWind, а також протоколу WebSocket. Під час проєктування системи було розроблено структуру майбутньої системи, структуру бази даних, схеми класів та UML діаграми, які допомогли розробити систему і визначити функції користувачів.

Для реалізації системи було використано мову програмування PHP та фреймворк Laravel. Для розробки інтерфейсу користувача використано фреймворк VueJS. Для обміну повідомленнями між сервером та клієнтами використано протокол WebSocket та окремий сервіс, розроблений з використанням мови програмування JavaScript.

У розробленій системі, власники вебсайтів можуть зареєструвати особисті кабінети та підключити систему інтернет-консультанта у вигляді плаваючого віджету, що дозволить їм покращити або налагодити комунікацію з власними користувачами і допомогти їм вирішувати будь-які питання. Власники сайтів можуть додавати агентів підтримки для перенесення функцій допомоги користувачам найманим робітникам.

За допомогою VueJS було створено швидкий динамічний сайт. У поєднанні з іншими технологіями і підходами до реалізації отримана система є дуже простою у використанні, має звичний і простий інтерфейс, проте виконує функції обміну повідомленнями з мінімальними затримками.

На фінальному етапі кваліфікаційної роботи було виконано тестування системи інтернет-консультанта. За результатами тестування було виявлено помилки у налаштуванні системи, які було вдало вирішено. Повторне тестування не виявило проблем. Система завантажена на вебсервер і доступна з будь-якого комп’ютера для будь-якого користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. What Are the Most Effective Channels for Customer Support? URL: https://www.groovehq.com/blog/customer-service-channels (дата звернення: 26.11.2023).

2. Онлайн чат для сайту – 13 кращих онлайн-консультантів, безкоштовні vs платні | Блог HOSTiQ.ua. URL: https://hostiq.ua/blog/ukr/online-consulting/ (дата звернення: 27.11.2023).

3. Платформа розвитку інтернет-продажів. URL: https://lexx.me (дата звернення: 27.11.2023).

4. What is live chat? URL: https://www.mindmesh.com/glossary/what-is-live-chat (дата звернення: 27.11.2023).

5. Zendesk – те, що допоможе вашому бізнесу. URL: https://www.partner-way.com/blog-ua/zendesk-how-this-service-can-help-your-business-ua (дата звернення: 29.11.2023).

6. Вебсайт «LiveAgent». URL: https://www.liveagent.com/ (дата звернення: 29.11.2023).

7. Laravel. URL: https://en.wikipedia.org/wiki/Laravel (дата звернення: 04.12.2023).

8. Що таке SQL та Бази даних? URL: https://acode.com.ua/sql-intro/ (дата звернення: 07.12.203).

9. Everything You Need to Know About Redis. URL: https://medium.com/codex/7-redis-features-you-might-not-know-bab8c9beb2c (дата звернення: 07.12.2023).

10. Що таке Node JS простими словами. URL: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami (дата звернення: 07.12.2023).

11. A Simple Explanation Of What A WebSocket Is. URL: https://medium.com/@yg17381/a-simple-explanation-of-what-a-websocket-is-

74b9bb14e85b (дата звернення: 07.12.2023).

12. Long Polling vs WebSockets – which to use in 2024. URL: https://ably.com/blog/websockets-vs-long-polling (дата звернення: 07.12.2023).

13. UUID – об’єкти UUID відповідно до RFC 4122. URL: https://docs.python.org/uk/3/library/uuid.html (дата звернення: 10.12.2023).

14. NGINX Reverse Proxy. URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (дата звернення: 11.12.2023).

15. Pub/Sub. URL: https://redis.com/glossary/pub-sub/ (дата звернення: 20.01.2023).

ДОДАТОК А

Тестування модуля авторизації

```
<?php

namespace Tests\Unit;

use App\Models\Organization;
use App\Models\User;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Http\Response;
use Tests\TestCase;


class AuthTest extends TestCase
{
    use RefreshDatabase;

    public function setUp(): void
    {
        parent::setUp();

        // 1. Create new organization
        $organization = new Organization();
        $organization->name = 'Test Organization';
        $organization->app_id = 'app-id';
        $organization->save();

        // 2. Create new user
        $user = new User();
        $user->email = 'test@example.com';
        $user->name = 'Test';
        $user->password = 'password';
        $organization->users()->save($user);
    }

    public function test_login()
    {
        // 1. Make http request
        $response = $this->post('/api/login', [
            'email' => 'test@example.com',
            'password' => 'password'
        ]);

        $response->assertStatus(Response::HTTP_OK);
    }

    public function test_wrong_email()
    {
        // 1. Make http request
        $response = $this->post('/api/login', [
            'email' => 'test2@example.com',
            'password' => 'password'
        ]);
```

```
        $response->assertStatus(302);
    }

    public function test_registration()
    {
        $response = $this->post('/api/register', [
            'email' => 'test2@example.com',
            'password' => 'password',
            'name' => 'Test2 User',
            'organization_name' => 'Test2 Organization'
        ]);

        $response->assertStatus(Response::HTTP_CREATED);
        $this->assertDatabaseHas('users', [
            'email' => 'test2@example.com'
        ]);
        $this->assertDatabaseHas('organizations', [
            'name' => 'Test2 Organization'
        ]);
    }

    public function test_registration_with_existing_email()
    {
        $response = $this->post('/api/register', [
            'email' => 'test@example.com',
            'password' => 'password',
            'name' => 'Test2 User',
            'organization_name' => 'Test2 Organization'
        ]);

        $response->assertStatus(302);
        $this->assertDatabaseMissing('organizations', [
            'name' => 'Test2 Organization'
        ]);
    }

}
```