ДИПЛОМНА РОБОТА
БАКАЛАВРА
ДРБ. ІП - 01.00.00.000 ПЗ
Група ІП-19-1К
Деренько Василь
2022

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерія програмного забезпечення

УДК _____004.4_____

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: "Розробка backengine бізнес логіки та WEB- сервісу системи комунікації з клієнтами засобами JS"

(назва згідно з наказом ректора)

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 00.00.00.001 ПЗ

(позначення)

Студент

ІП-19-1К _____ /В.В.Деренько /
(шифр групи) (підпис) (дата)(розшифровка підпису)

Керівник роботи

доц.,к.т.н. _____ / Бандура В. В./
(посада) (підпис) (дата) (розшифровка підпису)

Консультанти:

з питань нормоконтролю

доц., к.т.н_____ /В.Я. Піх/
(посада) (підпис) (дата)(розшифровка підпису)

Допускається до захисту
Завідувач кафедри

проф., д.т.н. _____ /В.І.Шекета/
(посада) (підпис) (дата) (розшифровка підпису)

2022
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
проф. В.І. Шекета
“ ____ ” _____2022 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Дереньку Василю Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) " Розробка backengine бізнес логіки та WEB- сервісу системи комунікації з клієнтами засобами JS "

керівник проекту (роботи) Бандура Вікторія Валеріївна

затверджені наказом вищого навчального закладу від “ 25 ” травня 2022 р. № _____

2. Строк подання студентом проекту (роботи) 17 червня 2022 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Огляд існуючих програмних рішень та сервісів в предметній області

2 Опис інструментів та середовища розробки

3 Власне оригінальне алгоритмічне та програмне рішення задачі

4. Тестування програмного продукту

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

1 Схема Бази Даних (рис. 3.1, ст. 40)

2 Таблиця users. (рис. 3.2, ст. 41)

3 Таблиця Chats (рис. 3.5, ст. 42)

4 Back-end, Структура проекту (рис. 3.9, ст. 46)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **25 травня 2022 р.**

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та огляд подібних програмних рішень	15.05.2021	виконано
2	Розробка структури бази даних та вибір сервера бази даних	20.05.2021	виконано
3	Розробка API	25.05.2021	виконано
5	Виконання тестування розробленого програмного продукту	06.06.2021	виконано
6	Оформлення пояснювальної записки дипломної роботи бакалавра	08.06.2021	виконано
7	Затвердження пояснювальної записки дипломної роботи бакалавра завідувачем кафедри	13.06.2021	виконано

Студент _____

(підпис)

Керівник _____

(підпис)

АНОТАЦІЯ

Темою дипломної роботи є Розробка backengine бізнес логіки та WEB-сервісу системи комунікації з клієнтами засобами JS. У роботі розглянуто подібні веб-сервіси та проаналізовано їх переваги та недоліки на основі яких розроблено програмний продукт. Був вибраний стек технологій для кожної складової програмного продукту та проаналізовані їх переваги та недоліки.

КЛЮЧОВІ СЛОВА: СЕРВІС, ОНЛАЙН ЧАТ, ОНЛАЙН ПІДТРИМКА, ВЕБ-САЙТ, NODE.JS, ORM-PRISM, DOCKER, POSTGRES, SOCKET.IO, NEST.JS.

ANNOTATION

The theme of the work is the development of backengine representations of business logic and WEB-service of communication system with clients by means of JS. The paper considers similar web services and analyzes their advantages and disadvantages on the basis of which the software product was developed. A technology stack was selected for each component of the software product and their advantages and disadvantages were analyzed.

KEY WORDS: SERVICE, ONLINE CHAT, ONLINE SUPPORT, WEBSITE, NODE.JS, ORM-PRISM, DOCKER, POSTGRES, SOCKET.IO, NEST.JS.

ЗМІСТ

ЗМІСТ **5**
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ **6**
ВСТУП **7**
1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ **8**
1.1 Back-End розробка 8
1.2 Веб-сервіси та API 10
1.3 Огляд існуючих програмних рішень на ринку 13
1.4 Постановка задачі 21
Висновок до розділу 1 23
2 ОПИС ІНСТРУМЕНТІВ ТА СЕРЕДОВИЩА РОЗРОБКИ **24**
2.1 Вибір серверної мови програмування 24
2.2 Вибір Node.JS фреймворка 28
2.3 Вибір системи управління базами даних 34
2.4 Додаткове програмне забезпечення 37
Висновок до розділу 2 39
3 ВЛАСНЕ ОРИГІНАЛЬНЕ АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ РІШЕННЯ ЗАДАЧІ **40**
3.1 Складові програмного рішення 40
3.1Розбір функціоналу програмного рішення 56
3.3 Подальший розвиток продукту 76
Висновок до розділу 3 77
ВИСНОВКИ **78**
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА **79**
БІБЛІОГРАФІЧНА ДОВІДКА **80**

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ТЗ – Технічне завдання

ПЗ – Програмне забезпечення

БД – База даних

SVG – Scalable Vector Graphics – Масштабована векторна графіка

MVP – Minimal Value Project – Мінімально життєздатний продукт

CRM – Customer relationship management – Управління відносинами з клієнтами

API - Application Programming Interface - інтерфейсів програмного забезпечення

MVC - Model-View-Controller - Модель–представлення–контролер

СУБД – система управління базами даних

ORM - Object-relational mapping - Об'єктно-реляційна проєкція

ВСТУП

У сучасному світі більшість послуг надається через всесвітню мережу. Клієнти вибирають інтернет-магазин замість звичайного магазину, оскільки це економить свій час і пропонує широкий спектр послуг або товарів. Завдяки своїм можливостям інтернет-магазини можуть обслуговувати більше клієнтів, ніж офлайн-магазини.

Обслуговування клієнтів завжди відігравало важливу роль у бізнесі, заснованому на торгівлі. Якість обслуговування прямо пропорційна успіху бізнесу та його подальшому розвитку. Хороший сервіс може збільшити кількість лояльних клієнтів і таким чином збільшити продажі тих самих послуг і товарів. Погане обслуговування може призвести до відтоку наявних і потенційних клієнтів, втрати продажів послуг або товарів і переміщення ваших клієнтів до конкурентів. Слід розуміти, що головне очікування клієнта від обслуговування полягає в тому, щоб швидко та професійно задовольнити його потреби.

Аналізуючи вищесказане, ми розуміємо, що для того, щоб отримати якісний сервіс, потрібно безпосередньо зв’язуватися зі своїми клієнтами. Бо якщо немає зв’язку із замовником, то немає і прибутку. Існує багато сервісів, які надають послуги для спілкування з клієнтами за допомогою засобів комунікації. Але не всі з них зручні у використанні. Метою цієї дипломної програми є створення сервісу, який допомагає створити зв’язок між операторами підтримки та клієнтами.

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Back-End розробка

Back-End розробка відноситься до серверної частини розробки, де ви в першу чергу зосереджені на тому, як працює сайт. Внесення оновлень та змін на додаток до моніторингу функціональності сайту буде вашим основним обов'язком. Цей тип веб-розробки зазвичай складається з трьох частин: сервера, програми та бази даних. Код, написаний розробниками серверної частини, є тим, що передає інформацію бази даних браузеру. Все, що ви не можете легко побачити оком, як-от бази даних і сервери, є роботою серверного розробника

Багато розробників серверного інтерфейсу знають мови інтерфейсу, такі як HTML і CSS, але їм потрібно використовувати такі мови, як Java, PHP, Ruby on Rails, Python і .Net, щоб виконати роботу з серверної частини. Розробники серверної частини найбільше зосереджені на швидкості реагування сайту. Ці мови використовуються для створення динамічних сайтів, які відрізняються від статичних сайтів тим, що ці типи веб-сайтів зберігають інформацію бази даних. Вміст на сайті постійно змінюється та оновлюється. Прикладами динамічних сайтів є Facebook, Twitter і Google Maps.

«Традиційний» бек-енд — це поєднання сервера, баз даних, API та операційних систем, які живлять інтерфейс програми. Щоб детально ознайомитися з програмним забезпеченням, яке пов'язує все це разом, прочитайте нашу статтю про сценарії на стороні сервера . Внутрішня частина програм може виглядати дуже по-різному від програми до програми, будь то використання хмарних серверів і сховищ даних, контейнеризація за допомогою таких служб, як Docker, постачальники Backend-as-a-Service (BaaS) або API для заміни. більш складна обробка.

Back-End розробка може бути набагато різноманітнішою, ніж розробка інтерфейсу, яка значною мірою керується JavaScript, HTML, CSS та різними інтерфейсними фреймворками, що використовують ці мови.

Щоб спростити речі, ми розділимо серверну частину на чотири основні компоненти «стеку програмного забезпечення»: сервер, базу даних, операційну систему та програмне забезпечення.

Сервери. З чотирьох компонентів вашого внутрішнього стека, незалежно від того, чи знаходиться він локально чи в хмарі, сервер діє як джерело життя мережі. Ці потужні комп'ютери забезпечують спільні ресурси, необхідні для роботи мереж, зокрема зберігання файлів, безпеку та шифрування, бази даних, електронну пошту та веб-служби.

Бази даних. Бази даних у контексті веб-сайту — це мозок, який робить веб-сайти динамічними. Кожного разу, коли ви запитуєте щось із веб-сайту, незалежно від того, чи шукаєте ви продукт в інтернет-магазині чи шукаєте розташування готелів у певному місті, база даних відповідає за прийняття цього запиту, отримання даних та повернення їх на веб-сайт. Бази даних також можуть приймати нові та відредаговані дані, коли користувачі веб-сайту чи програми взаємодіють з ними. Клієнт може змінювати інформацію в базі даних з браузера, незалежно від того, публікує користувач статті в CMS, завантажує фотографії в профіль соціальних мереж або оновлює інформацію про клієнта.

Проміжне програмне забезпечення. Проміжне програмне забезпечення, по суті, описує будь-яке програмне забезпечення на сервері, яке з'єднує фронт-енд частину програми з її бек-енд частиною. Подумайте про проміжне програмне забезпечення як про систему для вашого сайту — воно передає будь-яку комунікацію, як-от запити та відповіді, між вашою програмою та вашим сервером/базою даних. Ви не бачите проміжного програмного забезпечення, але воно є, і воно має бути надійним і завжди робити те, що від нього очікують.

Проміжне програмне забезпечення (програмне забезпечення на стороні сервера) полегшує підключення клієнт-сервер, утворюючи проміжний рівень

між програмою і мережею: сервером, базою даних, операційною системою тощо. Проміжне програмне забезпечення може бути багаторівневим, організованим у різні шари сайту, будь то рівень презентації чи бізнес-рівень. Тут також можуть використовуватися веб-API, створюючи міст між бізнес-рівнем і рівнем презентації.

Проміжне програмне забезпечення також дозволяє хмарним додаткам і локальним програмам «розмовляти», а також надає такі послуги, як інтеграція даних та обробка помилок.

API. Сьогодні ви не можете говорити про внутрішню частину програми, не торкаючись API і того, як вони легко з'єднують програмне забезпечення, програми, бази даних і служби. API відіграють невід'ємну роль у створенні більшості архітектур програмного забезпечення на стороні сервера, часто замінюючи більш складне програмування, щоб дозволити програмному забезпеченню обмінюватися даними та передавати дані.

1.2 Веб-сервіси та API

Веб-сервіси та API — це два з тих технічних термінів, які перетинаються, які регулярно плутають. Можливо, ви навіть чули, як ці слова використовуються як синоніми, але чи це одне й те саме?

Незважаючи на те, що веб-сервіси та API є двома різними поняттями, вони мають певний збіг.

Що таке API?

Термін «API» означає «Application Programming Interface» Інтерфейс програмування прикладних програм . Якщо розкласти це слово за словом, можна досить добре зрозуміти, що воно означає. API – це інтерфейс, який можна використовувати для програмування програмного забезпечення, яке взаємодіє з наявною програмою. На практиці API — це «набір функцій і процедур», які дозволяють отримати доступ до даних і функціональних можливостей існуючої програми та розширити їх.

API існують протягом багатьох років. Досить скоро після того, як було створено першу комп'ютерну програму, кмітливі розробники зрозуміли, що ви можете отримати набагато більше від різних програм, об'єднавши їх разом — і так народився API.

Зараз, коли ми говоримо про API, ми зазвичай маємо на увазі веб-API , які надають дані та функціональні можливості програми через Інтернет. Якщо ви придивитесь, ви побачите, що веб-API забезпечують наше повсякденне життя:

- Коли ви входите на веб-сайт за допомогою свого профілю Facebook
- Коли ви увімкнете Netflix і побачите десятки нових фільмів, на екрані випливають десятки нових фільмів
- Коли ви шукаєте авіарейси в Google

Технічно кажучи, веб-API зазвичай надсилають дані назад і вперед за допомогою HTTP-запитів. Ці запити часто повертають текстові дані у вигляді відповіді JSON або XML.

Що таке веб-сервіс?

Веб-сервіс — це програмна система, розроблена для підтримки взаємодії між машинами через мережу. Він має інтерфейс, описаний у форматі, який обробляється машиною (зокрема, WSDL). Інші системи взаємодіють із веб-сервісом у спосіб, визначений його описом, використовуючи SOAP-повідомлення, які зазвичай передаються за допомогою HTTP із серіалізацією XML у поєднанні з іншими стандартами, пов'язаними з Інтернетом.

Веб-сервіси зазвичай асоціюються з сервіс-орієнтованою архітектурою . Архітектура, орієнтована на обслуговування, або SOA, — це архітектурний зразок для проектування програмних додатків, де функції розділені й доступні як служби в мережі.

API проти веб-сервісів

На відміну від того, що ви думаєте, API та веб-сервіси не є взаємовиключними. Фактично, одне є підмножиною іншого: кожна веб-служба є API — оскільки вона розкриває дані та/або функціональні можливості

програми, — але не кожен API є веб-сервісом. Це пов’язано з тим, що визначення веб-сервісу є досить обмеженим, коли йдеться про реалізацію:

- Для веб-сервісів потрібна мережа. Хоча API-інтерфейси можуть бути в режимі онлайн або офлайн, веб-служби повинні використовувати мережу.
- API не залежать від протоколу . Хоча API можуть використовувати будь-які протоколи або стилі дизайну, веб-сервіси зазвичай використовують SOAP (але іноді REST, UDDI і XML-RPC).

Є ще одна велика відмінність. Багато публічних API є прозорими, з відкритою документацією та порталами самообслуговування для швидкої адаптації розробників. Це тому, що сенс багатьох сучасних API, зрештою, полягає в тому, щоб полегшити взаємодію з програмою. З іншого боку, веб-сервіси не користувалися такою відкритою історією: натомість вони, як правило, пропонують конкретні дані та/або функціональні можливості конкретним партнерам — їх не потрібно зламувати .

Для розробників, для яких веб-сервіси обов’язково означають SOA, є одна велика проблема з веб-сервісами. У сервіс-орієнтованій архітектурі служби мають бути розроблені з урахуванням того, як вони в кінцевому підсумку будуть використовуватися і хто буде їх використовувати. Якщо не вистачає належного планування, або якщо потреби змінюються з часом, розробники можуть швидко виявити, що змішують мішанину сервісів для створення правильної реалізації.

По суті, веб-сервіси не мають цієї проблеми. Ці проблеми можуть виникати лише тоді, коли веб-служби використовуються в сервіс-орієнтованій архітектурі.

API — це інтерфейс, який дозволяє розбудовувати дані та функціональні можливості іншої програми, а веб-сервіс — це мережевий ресурс, який виконує конкретне завдання. Так, між ними є збіг: усі веб-сервіси є API, але не всі API є веб-сервісами.

І веб-сервіси, і API — за своєю суттю — дуже корисні та дуже широко використовуються сьогодні. Однак саме веб-сервіси, пов’язані з SOAP та/або сервісно-орієнтованою архітектурою, втрачають популярність.

1.3 Огляд існуючих програмних рішень на ринку

Smartsupp – це недороге програмне забезпечення для підтримки чату, яке дозволяє відстежувати та записувати сеанси користувачів на вашому веб-сайті. Це допоможе вам зрозуміти, як потенційні клієнти використовують сайт і що заважає їм зробити покупку. (Рисунок - 1.1).

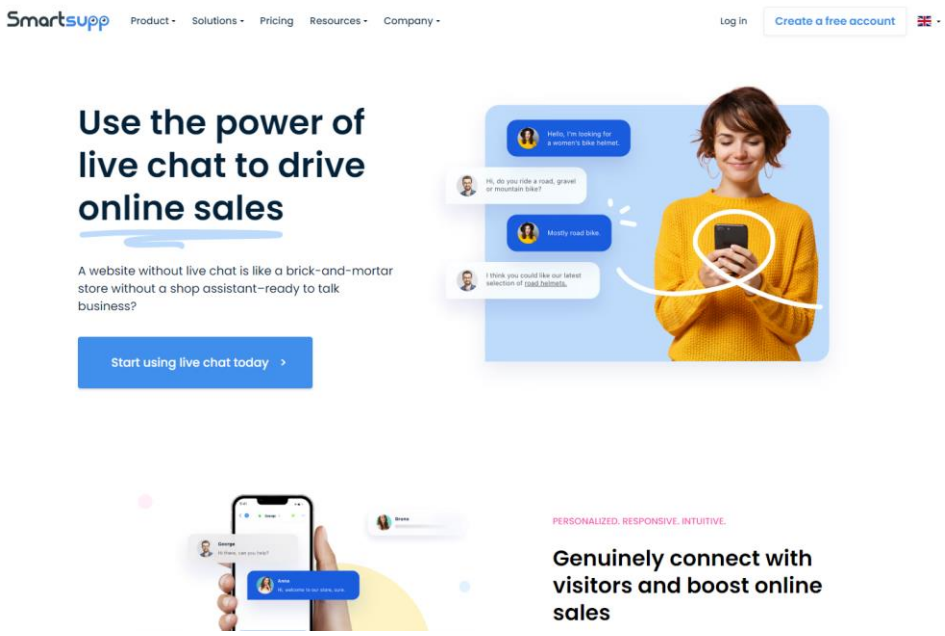


Рисунок 1.1 - Smartsupp, Головна сторінка

На головній сторінці сайту ми можемо бачити навігаційне меню, кнопки входу у свій акаунт та кнопку «create for free account». Для того щоб ми могли переглянути функціонал даного веб-сервісу, необхідно зареєструватись натиснувши кнопку «create for free account». Після активації даної кнопки ми потрапляємо на сторінку реєстрації(Рисунок 1.2)

Як ми бачимо, даний сервіс дає можливість реєстрації за допомогою акаунта Google або Apple. Це є досить корисно в зв'язку з тим що, полегшує та пришвидшує етап реєстрації нового акаунта.

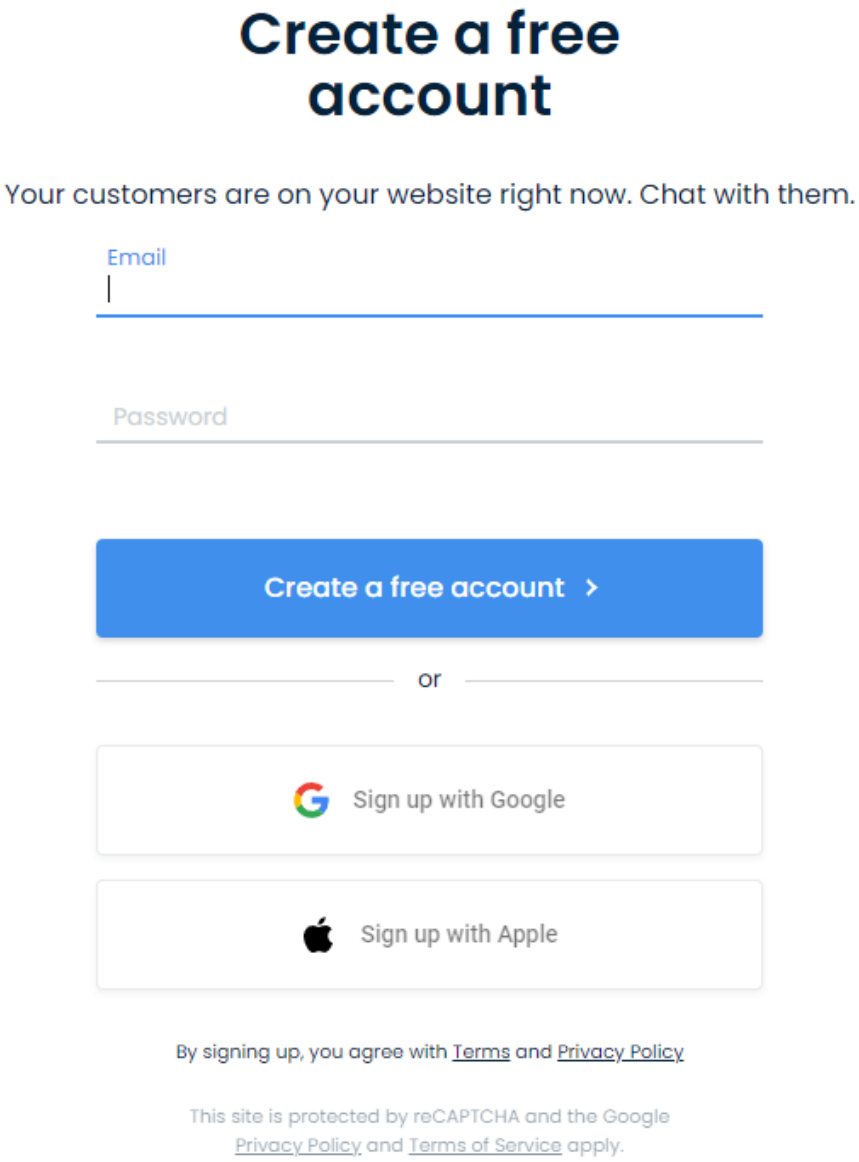


Рисунок 1.2 – Форма реєстрації.

Після реєстрації ми потрапляємо на основну сторінку (Рисунок - 1.3). На даній сторінці ми можемо відразу виділити для себе основні функції а саме:

- Чат
- Статистику
- Список відвідувачів
- Автоматизація
- Налаштування

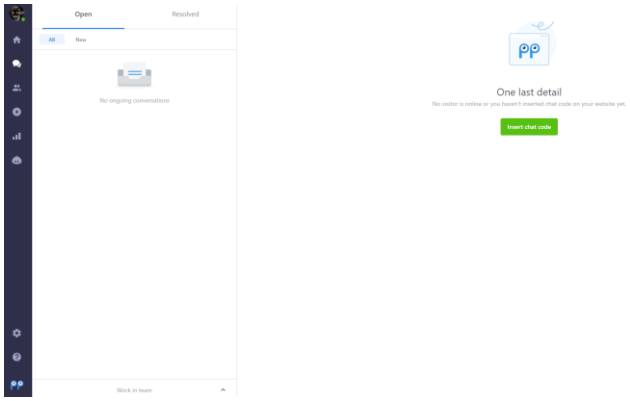


Рисунок 1.3 – головна сторінка

За кожну функцію відповідає відповідна кнопка у панелі навігації яка знаходиться ліворуч екрану. Самою основною функцією є функція чату. За допомогою даного чату клієнт може зв’язатися з оператором та отримати відповіді на свої запитання. Вигляд чату зі сторони оператора можемо побачити на рисунку 1.4.

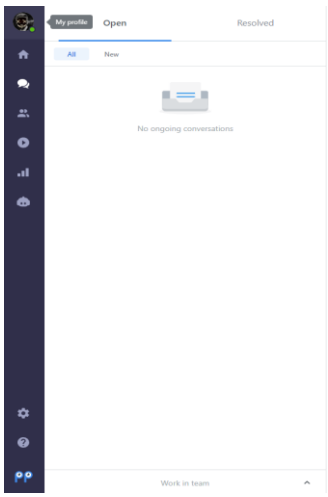


Рисунок 1.4 - Вигляд чату зі сторони оператора

При детальному розборі даного чату ми бачимо, що чат ділитт діалоги на 2 категорії, а саме:

- Відкриті
- Завершені

Як ми розуміємо у категорію «завершені» потрапляють уже опрацьовані чати. А от категорія «відкриті» ділиться на дві підкатегорії:

«Усі» та «Нові».

У категорію «Усі» як можемо зрозуміти по назві, потрапляють усі відкриті чати. А у категорію «Нові» потрапляють чати які ще не прийняли оператори на обробку.

У розділі про статистику (Рисунок 1.5) ми маємо змогу переглядати різного роду статистичні дані, такі як кількість нових чатів, кількість відповідей на чати, Кількість закритих чатів, середній рейтинг, сумарний час обробки чату, та кількість робочих годин.

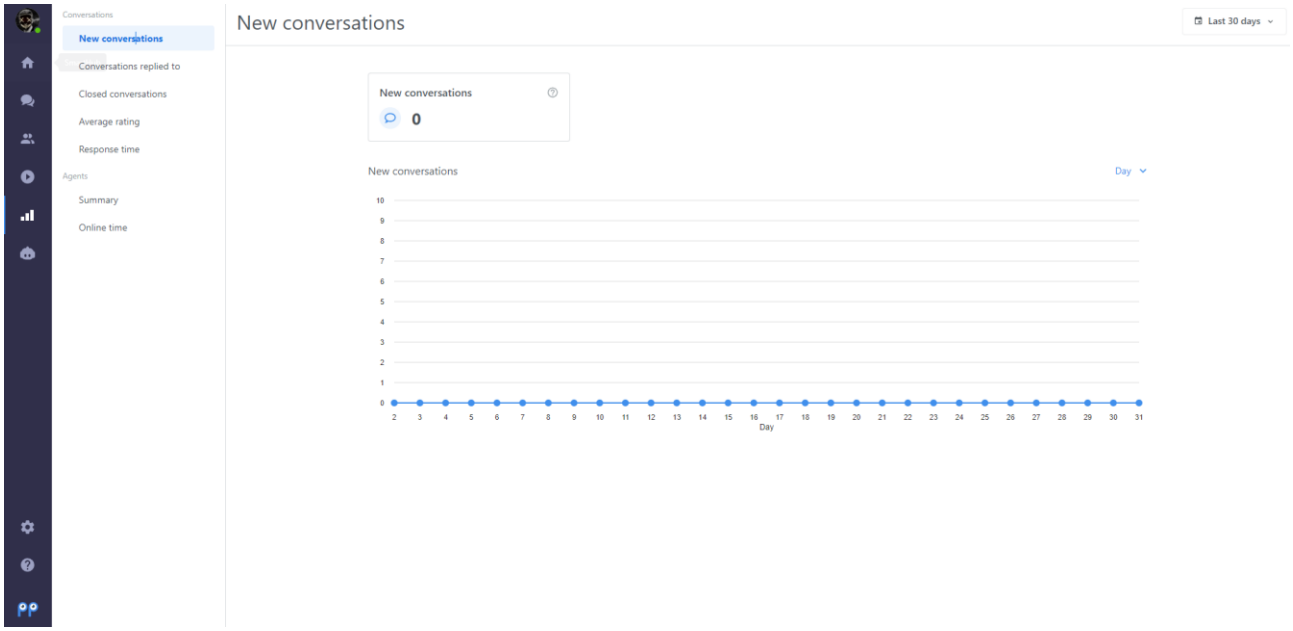


Рисунок 1.5 – Категорія статистики

Ще один розділ який ми розглянемо це «Налаштування».(рисунок 1.6)

У даному розділі користувач має змогу конфігурувати власні налаштування акаунту, створювати акаунти для нових операторів, створювати чи видаляти групи для операторів, проводити кастомізацію чат-боксу який буде на стороні клієнта, налаштовувати швидкі відповіді, та налаштовувати сповіщення.

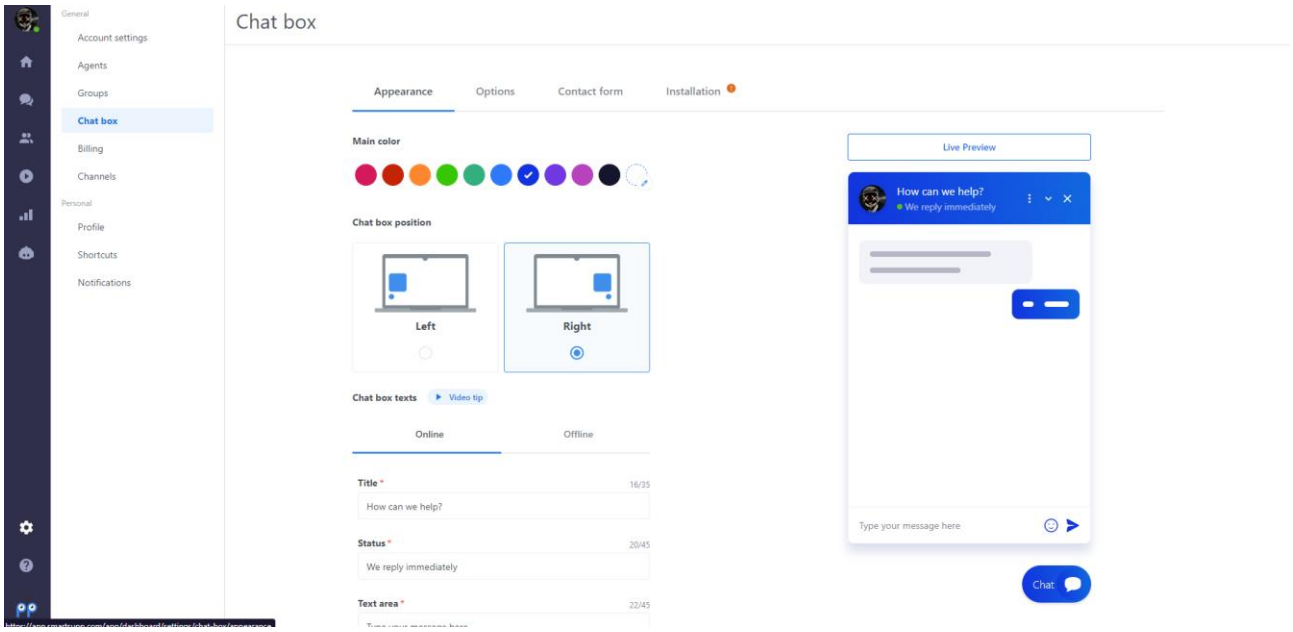


Рисунок 1.6 - Вигляд розділу налаштувань

Провівши аналіз даного веб-сервісу, уже можна сформувати початкові дані для власного проекту. Ми уже можемо визначитись із приблизними функціями які будуть присутні у нашому проекті.

Chaport – ще один сервіс який надає можливість комунікації з клієнтами безпосередньо з власного сайту.(Рисунок 1.7) Основна відмінність даного сервісу з попереднім є те, що цей сервіс призначений для сайтів на платформі WordPress.

На даному сайті як і в попередньому для аналізу функцій необхідно зареєструватись, для цього необхідно натиснути на кнопку «Try for free».

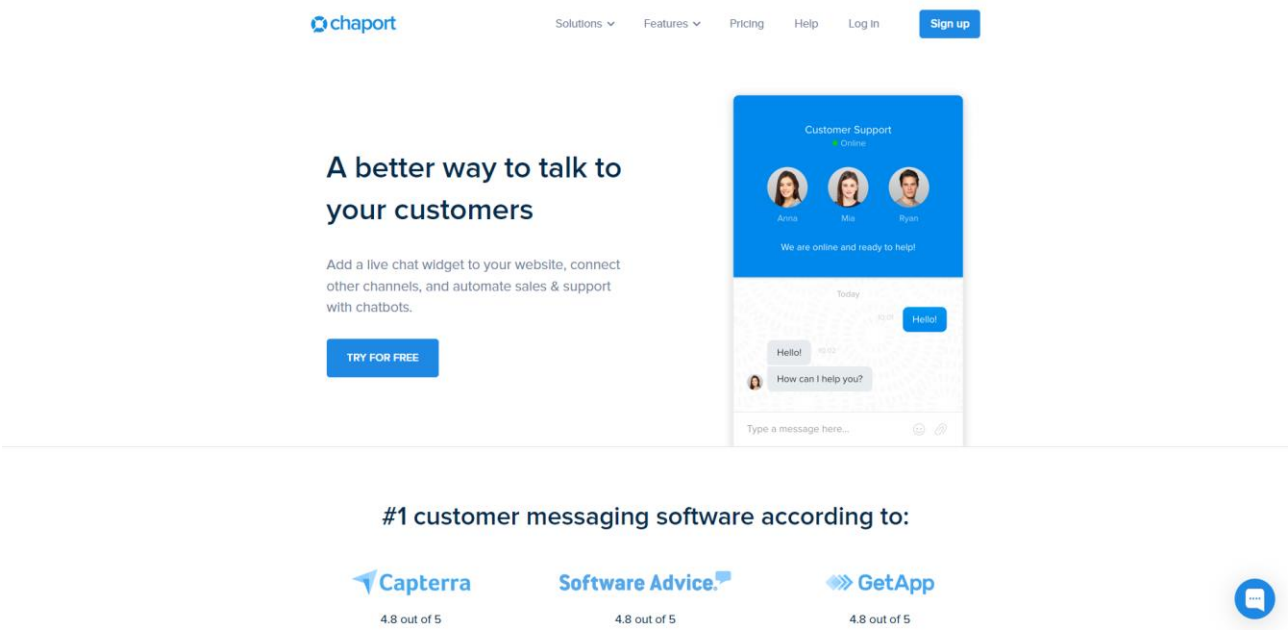


Рисунок 1.7 – Chaport головна сторінка

Для реєстрації необхідно ввести дані електронної пошти та придумати пароль. Вікно реєстрації ми можемо побачити на рисунку 1.8.

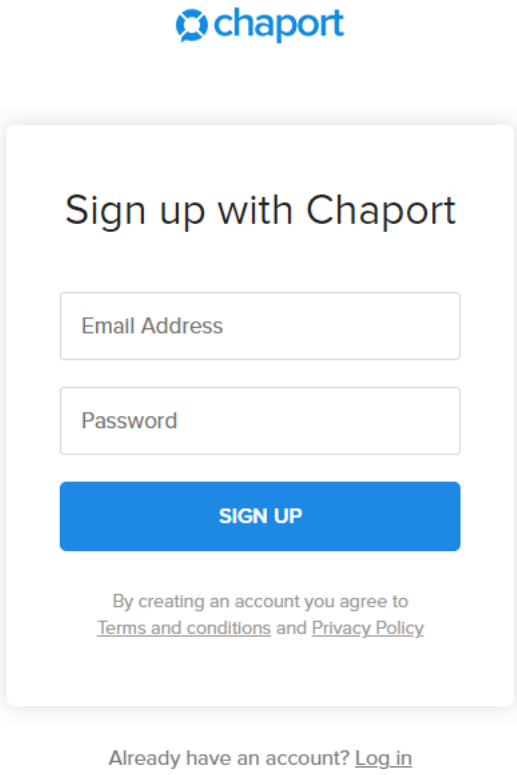


Рисунок 1.8 – форма реєстрації

Після реєстрації ми потрапляємо у особистий кабінет де нас відразу зустрічає сторінка на якій відбувається листування з клієнтами. (Рисунок 1.9)

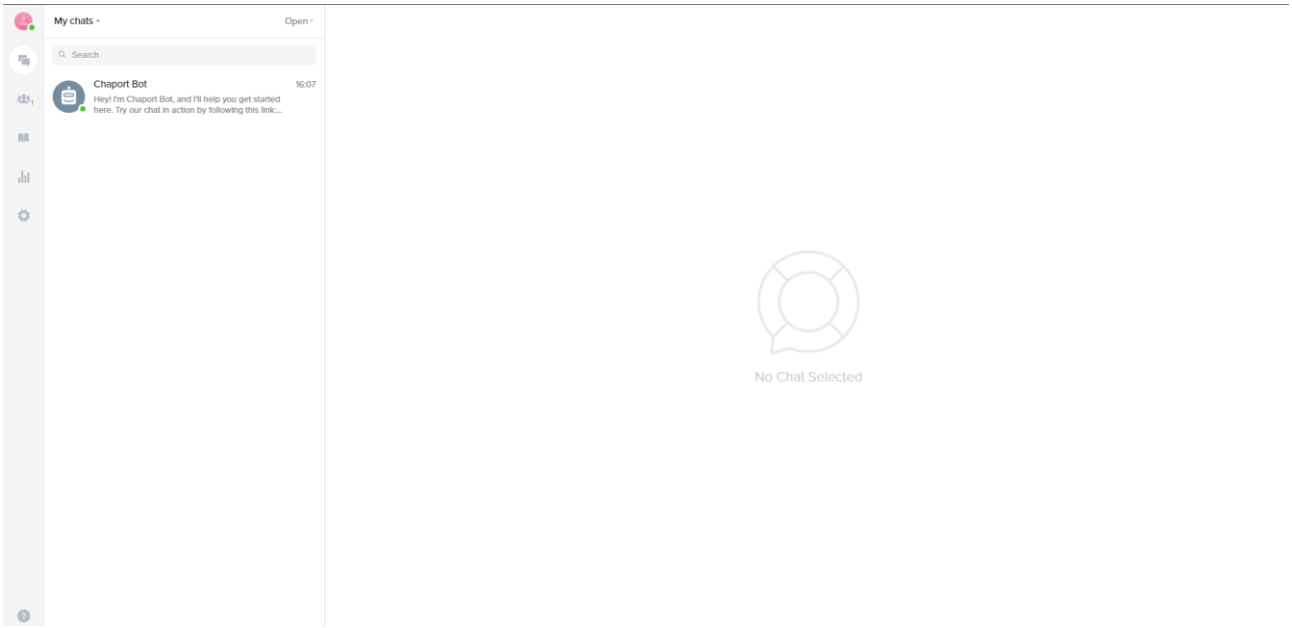


Рисунок 1.9 – Особистий кабінет

Ліворуч ми можемо бачити бічну панель навігації, яка розміщує у собі наступні кнопки. Chats – активована за замовчуванням та відкриває вікно з чатами. Кнопка Operators – відкриває розділ з операторами.(Рисунок 1.10)

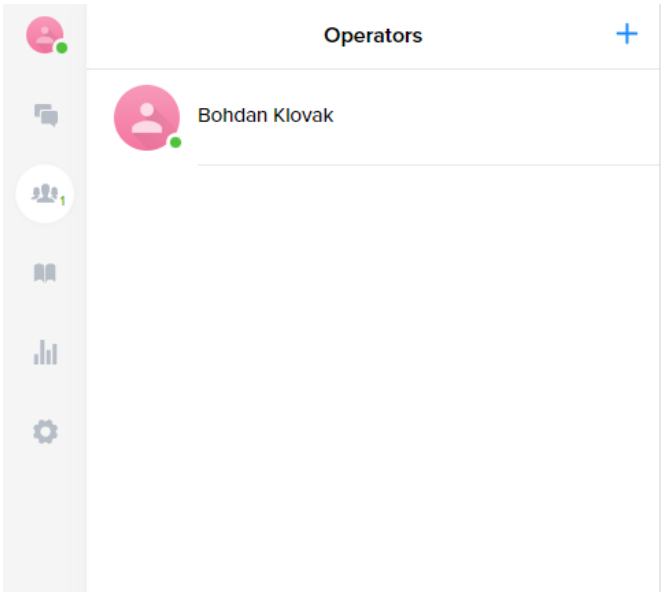


Рисунок 1.10 – розділ Operators

У даному розділі користувач має змогу створити нового оператора або ж видалити існуючого. Також натиснувши на існуючого оператора можна переглянути інформацію про нього. (Отсунок 1.11)

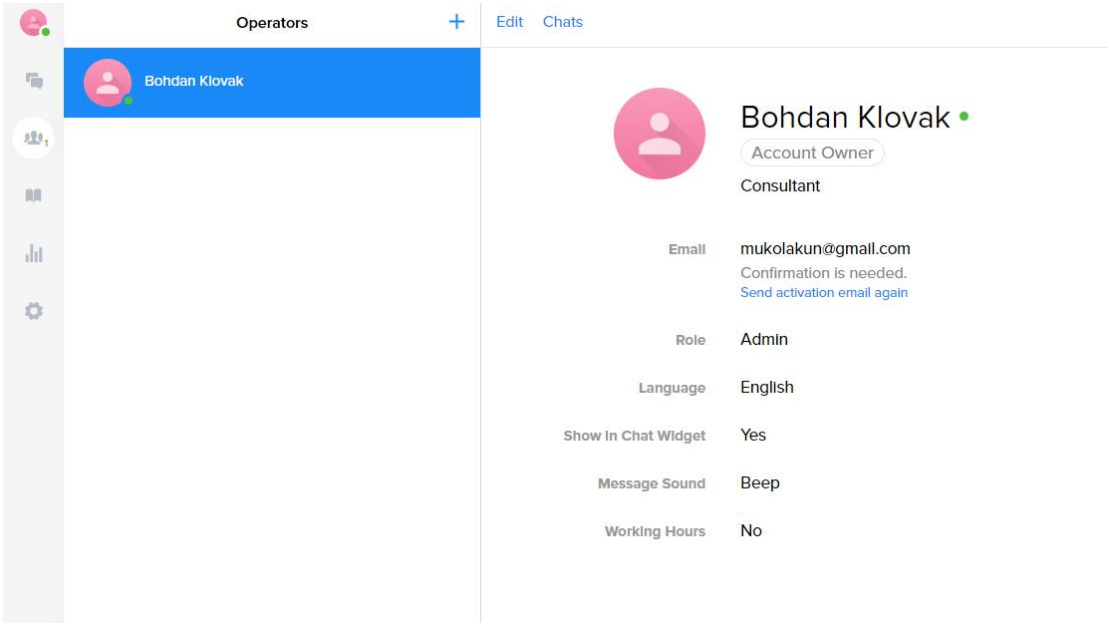


Рисунок 1.11 – інформація про оператора

Наступна кнопка «FAQ» - відкриває розділ з довідником у якому адміністратор може описати для операторів усі важливі нюанси роботи чи якусь інформацію яку він забажає показати операторам. (Рисунок 1.12)

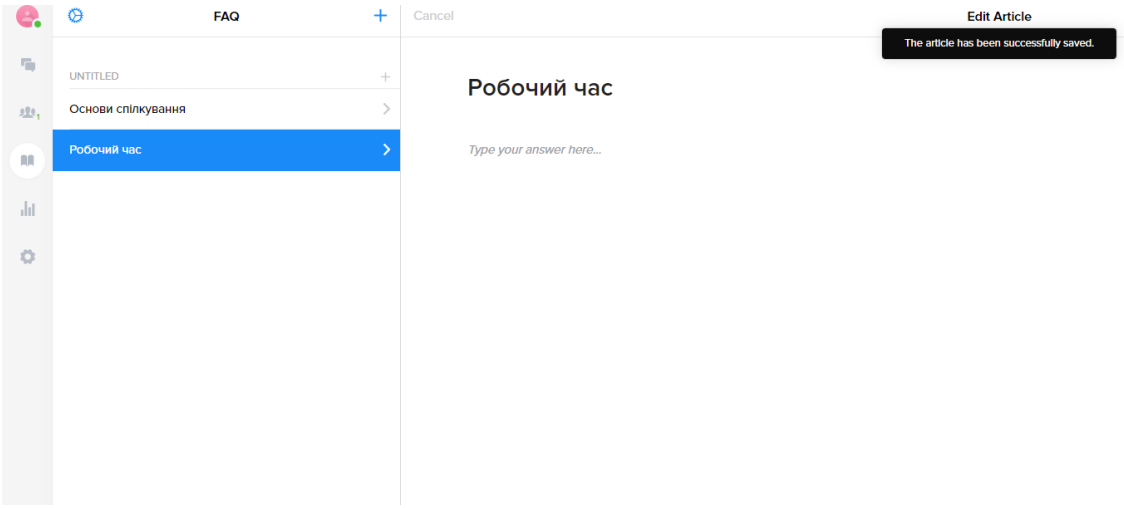


Рисунок 1.12 – розділ довідник

Останній розділ який ми розглянемо це розділ «Reports»(Рисунок 1.13) у якому знаходяться усі статистичні дані такі як:

- Загальна кількість чатів
- Кількість активних чатів
- Статистика роботи чат-ботів
- Рейтинг чатів
- Час листування
- Робочий час оператора

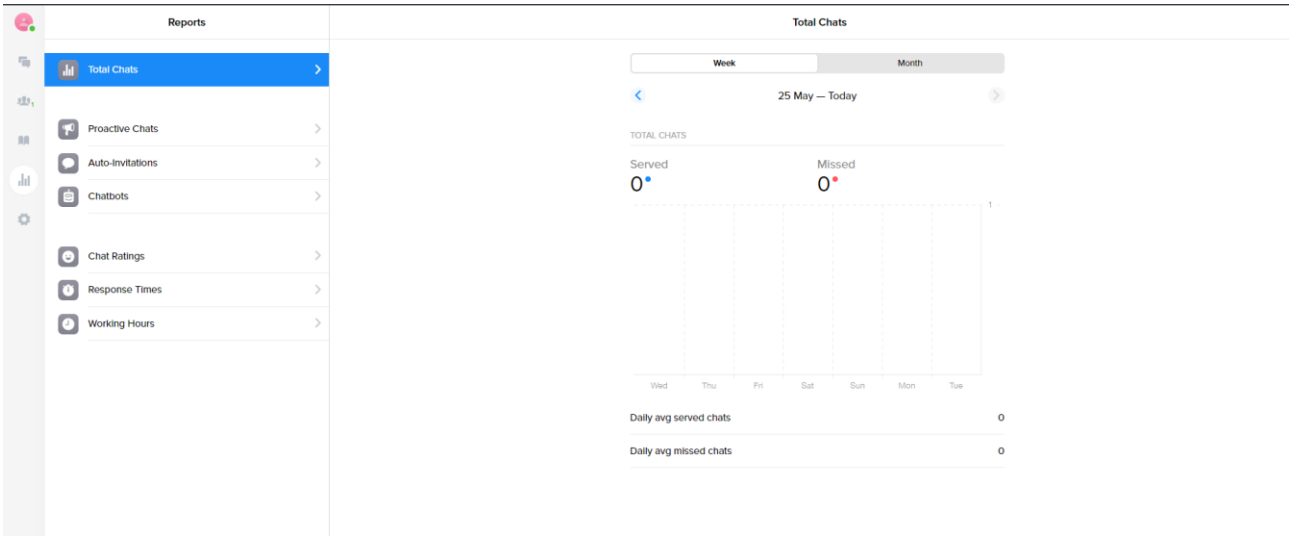


Рисунок 1.13 – розділ статистика

Уся статистика представлена у вигляді зручних графіків з можливістю вибору періодів графіка та іншої кастомізації.

Провівши аналіз даного веб-сервісу, та опираючись на попередній аналіз ми уже можемо повністю постановити ТЗ для даного дипломного проекту.

1.4 Постановка задачі

Завдяки цим проаналізованим онлайн-сервісам ми вже можемо створити технічне завдання для нашого веб-додатка. Кожен користувач буде переміщений на сторінку входу, де він зможе увійти в існуючий обліковий запис або

зареєструвати новий. Реєстрація на наш сервіс необхідна. Після авторизації адміністратор переходить на сторінку панелі керування, де розташована інтерактивна панель.

Першою вкладкою буде вкладка «Агенти», де адміністратор зможе переглянути існуючих операторів, стан їх мережі, рейтинги та кількість оброблених чатів, або, натиснувши кнопку «Створити» у спливаючому вікні, він зможе створити обліковий запис для нового оператора.

У вкладці «Клієнти» адміністратор зможе переглянути інформацію про клієнта за допомогою онлайн-чату підтримки, а саме: ім’я, адресу електронної пошти, номер телефону, призначений клієнтом, та дату зв’язку з оператором. Дані можна сортувати за датою чату .

У вкладці «Статистика» для адміністратора буде доступно два графіки зі статистикою кількості чатів за останній тиждень, та кількості нових клієнтів за тиждень.

Вкладка «Профіль» буде надавати доступ для редагування власних даних, таких як ім’я та адреса власного сайту. Також для перегляду буде доступно адреса електронної скриньки та дату створення акаунту.

Останньою вкладкою буде «Довідник» у якому будуть зберігатися відповіді на часті запитання від клієнтів. Довідник існуватиме для пришвидшення відповіді клієнтам операторами. Адміністратор зможе вказувати відповіді на часті запитання і ці відповіді будуть доступні для оператора в якості шпаргалки.

Для входу на сайт в якості оператора, потрібно буде авторизуватися за допомогою електронної пошти і паролю який вам повинен надати адміністратор. Після авторизації оператор потрапляє на сторінку чатування де також буде декілька вкладок.

Вкладка «Чати» у якій оператор бачитиме фільтр чатів, де у фільтри «усі» будуть добавлятись усі чати, у фільтр «активні» будуть добавлятись активні чати з якими зараз працює оператор, та фільтр «нові» куди будуть приходити нові

повідомлення від клієнтів і для початку діалогу з ними оператор повинен буде підтвердити початок чату з цими клієнтами. Також у оператора буде можливість завершити чат.

Вкладка «Довідник» надає можливість перегляду відповідей на часті запитання від клієнтів які добавив адміністратор сайту.

У вкладці «Шаблони» оператор зможе створювати шаблонні повідомлення які згодом зможе використовувати у спілкуванні з клієнтом задля швидшого обслуговування.

Також буде вкладка «Профіль» у якій оператор зможе переглянути кількість опрацьованих чатів, свій рейтинг, час початку роботи, та свою адресу електронної скриньки а також зможе відредагувати своє ім’я за необхідності

У клієнтській частині повинна бути кнопка після натиску на яку буде відкрито чат в якому клієнт повинен вказати своє ім’я, адресу електронної скриньки і за бажанням номер мобільного телефону, після чого дочекатись під’єднання оператора. Після завершення чату клієнту буде запропоновано оцінити роботу оператора в форматі вибору кількості зірочок.

Висновок до розділу 1

В даному розділі було розглянуто декілька схожих по суті сайтів, розібрано їх функціонал та дизайн, та на основі цього складені основні вимоги до функціональної частини майбутнього програмного продукту. Також був проаналізований процес створення веб-сайтів, розглянуто кожен етап та його особливості. Було визначено що таке веб-сервіси, якими вони бувають, так яке відношення до них має така річ як API. На основі усього вище переліченого були складені та скореговані вимоги до майбутнього програмного продукту.

2 ОПИС ІНСТРУМЕНТІВ ТА СЕРЕДОВИЩА РОЗРОБКИ

2.1 Вибір серверної мови програмування

Node.js, PHP і Python — популярні технології бекенда, які використовуються для створення веб-додатків. Кожна технологія має свої унікальні переваги та обмеження.

Перш ніж перейти до поглибленого порівняння Node.js проти PHP та Python, давайте спочатку поглянемо на короткий огляд кожної технології. Це допоможе вам зрозуміти ключові відмінності між кожною технологією.

Node.js — це кросплатформне середовище виконання з відкритим вихідним кодом і серверним кодом JavaScript для створення серверних і мережевих програм. Він надає велику бібліотеку численних модулів JavaScript, що спрощує процес розробки веб-додатків.

Ось основні особливості Node.js:

- Асинхронна й неблокуюча система вводу-виводу, яка забезпечує масштабованість та високу продуктивність
- Підтримка фронтенду та бекенда в JavaScript
- Кросплатформний фреймворк для розробки додатків для Windows, Mac і Linux
- Кастомізованість
- Усуває необхідність повторного виконання коду за допомогою його засобів кешування

PHP або Hypertext Preprocessor — це мова сценаріїв на стороні сервера. Ця загальна мова програмування сценаріїв, яку можна легко вбудувати в коди HTML, ідеально підходить для створення динамічних веб-сайтів і веб-додатків.

Це мова з відкритим вихідним кодом і кросплатформна. Таким чином, програми, створені за допомогою PHP, можуть працювати на різних платформах, включаючи Windows, Linux тощо.

Ось основні функції PHP:

- Містить попередньо написані та готові до використання коди, які прискорюють розробку
- Підтримка багатьох служб баз даних, включаючи MySQL, Oracle, Sybase тощо
- Постачається з вбудованими функціями безпеки для захисту від поширених загроз безпеки, таких як ін'єкції SQL, підробка даних тощо
- Програми на основі PHP підтримуються численними хмарними сервісами, такими як AWS

Python — це універсальна високорівнева об’єктно-орієнтована мова програмування, яка часто використовується для розробки веб-сайтів і програмного забезпечення, аналізу даних та автоматизації завдань. Таким чином, він не обмежується лише розробкою веб-сайтів.

Коли справа доходить до веб-розробки, роль Python може включати, але не обмежувати, обробку даних, взаємодію з базами даних, надсилання даних на/з серверів, маршрутизацію URL-адрес тощо.

Ось основні особливості Python:

- Простий синтаксис, що полегшує розуміння та вивчення Python
- Постачається з великою кількістю вбудованих модулів і бібліотек
- Незалежний від платформи – означає, що програми, створені за допомогою Python, працюють на різних платформах, включаючи Windows, Mac і Linux.
- Динамічно типізована мова – на відміну від Java.

Тепер давайте детальніше порівняємо всі ці технології бекенда на основі ключових факторів

Звичайні випадки використання Node.js, PHP і Python як бекенд-технології для веб-розробки та деякі широко відомі програми, створені за допомогою цих технологій.

Існують широко популярні програми, створені за допомогою Node.js, зокрема, Netflix, PayPal, Uber, Groupon, NASA та LinkedIn .

Тому Node.js як бекенд-технологія ідеально підходить для створення:

- Додатки для чату та співпраці в режимі реального часу
- Програми для соціальних мереж
- Програми для спільного використання поїздок
- Програми Інтернету речей
- Мікросервіси

Відомі програми та веб-сайти, створені за допомогою PHP, зокрема, Facebook, Tumblr, Slack, WordPress, Etsy тощо.

Тому PHP як бекенд-технологія ідеально підходить для створення:

- Системи управління контентом
- Програми на основі графічного інтерфейсу
- Веб-сайти та веб-додатки
- Програми для електронної комерції
- Представлення даних

Python є кращою мовою програмування для розробки серверної частини веб-сайтів і веб-програм для багатьох провідних компаній, таких як Google, Spotify, Netflix, Facebook тощо.

Враховуючи це, Python як бекенд-технологія ідеально підходить для створення:

- Розробка MVP
- Корпоративні програми
- Розробка гри
- Веб-сайти та веб-додатки

Вибираючи технологію бекенда, важливо подивитися, який тип архітектури підтримує технологія, оскільки вона впливає на загальну продуктивність і масштабованість веб-додатка.

Node.js

Node.js використовує архітектуру однопотокового циклу подій для керування кількома одночасними запитами без будь-яких обмежень. Крім того, це дозволяє розробникам використовувати архітектурну модель MVC або MVP.

PHP

Коли справа доходить до PHP, на вибір є дуже багато фреймворків, включаючи Laravel, Codeigniter, CakePHP, і більшість з них підтримують архітектури MVC.

Python

На відміну від Node.js, Python не підтримує багатопоточність. Як і PHP, існує багато фреймворків Python, найпопулярнішим є Django, який відповідає архітектурі Model View Template (MVT).

Працюючи з базою даних, ви дійсно хотіли б знати, чи підтримується вибрана вами технологія бекенда знайомими вам базами даних. Чи не так? Отже, давайте подивимося на підтримку баз даних для кожної технології.

Node.js забезпечує підтримку всіх типів баз даних, незалежно від реляційної бази даних або бази даних NoSQL. Який з них вибрати, залежить від мети та складності вашого проекту веб-розробки.

PHP забезпечує підтримку багатьох баз даних, включаючи MySQL, Oracle, Db2, MariaDB, SQLite, MongoDB і PostgreSQL.

Python підтримує численні бази даних, включаючи, SQLite, Oracle, Sybase, MySQL, PostgreSQL тощо.

Усі згадані тут бекенд-технології використовуються лідерами галузі. Отже, одне можна з упевненістю сказати, що всі ці технології є чудовими варіантами, коли справа доходить до серверної частини веб-розробки. І який з них ідеально підходить для вашого конкретного проекту? Це залежить від особливостей вашого проекту – особливо від типу веб-додатка, який ви хочете розробити.

Враховуючи попередні випадки використання, кожною технологією захоплюються підприємства різних вертикалей. Таким чином, незалежно від того, який ваш бізнес-жанр, ви можете подумати про всі ці бекенд-технології.

Вибір мови програмування

Для розробки дипломного проекту було вирішено використовувати Node.js, тому що основною функцією даного проекту є чат між оператором та клієнтом. А чат є найбільш типовим багатокористувацьким додатком у режимі реального часу. Від IRC (у той час) через безліч власних і відкритих протоколів, що працюють на нестандартних портах, до можливості реалізувати все сьогодні в Node.js за допомогою веб-сокетів, що працюють через стандартний порт 80.

На мою думку Node.js найкраще зможе реагувати на події, обробляти багато одночасних з'єднань та підтримувати плинність у користуванні.

2.2 Вибір Node.JS фреймворка

Робота з фреймворком складніше, ніж з готовим движком, але результат буде отримано в рази швидше, ніж при написанні коду. Також фреймворки забезпечують:

- високу швидкість створення додатків, сервісів
- можливість постійного розвитку продукту за рахунок інтеграції в структуру нових розширень;
- простий зручний супровід;
- застосування фреймворка для сайту забезпечує йому швидкість реагування, можливість витримувати значні навантаження.

Тому для розробки було вирішено використовувати Node.js Фреймворк.

Розглянемо популярні фреймворки NodeJs:

Hapi.js

Його представив Еран Хаммер з Walmart, намагаючись впоратися з трафіком у чорну п'ятницю. Це потужна та надійна найкраща JS-фреймворк вузла з відкритим кодом для розробки JSON API. Сервери інтерфейсу

прикладного програмування (API), веб-сайти та проксі-додатки HTTP створені за допомогою hapi.js. Різноманітні ключові функції, такі як перевірка введених даних, реалізація кешування, функціональні можливості на основі конфігурації, обробка помилок, ведення журналів тощо, а також добре розвинена система плагінів, роблять Hapi однією з найбільш бажаних платформ. Він використовується для створення корисних програм і надання технологічних рішень кількома великими веб-сайтами, такими як PayPal, Disney.

Особливості:

- Можливість повторного використання коду
- Ніяких зовнішніх залежностей
- Безпека
- Інтегрована архітектура: комплексний API авторизації та аутентифікації, доступний у структурі вузла.

Express.js

Створений TJ Holowaychuk, Express.js — це гнучкий і мінімальний фреймворк додатків Node.js, спеціально розроблений для створення односторінкових, багатосторінкових і гібридних додатків, які забезпечують надійний набір функцій для веб- та мобільних додатків.

Express не має готового механізму об’єктно-реляційного відображення. Express не ґрунтується на конкретних компонентах, тому що «не має думки» щодо технологій, які ви до нього підключаєте. Ця свобода в поєднанні з блискавично швидким налаштуванням і чистим середовищем JavaScript Node робить Express сильним кандидатом для швидкої розробки та швидкого створення прототипів. Express найбільш популярний серед стартапів, які хочуть створити продукт якомога швидше і не мають дуже багато застарілого коду.

Перевагою фреймворка є постійні оновлення та реформи всіх основних функцій. Це мінімалістичний фреймворк, який використовується для створення кількох мобільних додатків та API.

Koa.js

Розроблений та підтримуваний творцями широко використовуваного фреймворку Node.js — Express.js, Koa, міжплатформне середовище виконання на стороні сервера, є об’єктом, що містить набір функцій проміжного програмного забезпечення, які складаються та виконуються у вигляді стека. запит, що полегшує веб-розробникам створення швидких і масштабованих мережевих додатків за допомогою JavaScript. Це покращує сумісність, надійність і робить написання проміжного програмного забезпечення набагато приємнішим.

Зараз багато веб-розробників навіть використовують Node.js для написання інтерфейсу та бекенда веб-програми на JavaScript. Веб-розробники можуть ще більше прискорити розробку користувацьких веб-додатків та інтерфейсів програмного забезпечення (API) за допомогою кількох фреймворків Node.js.

Sails.js

Це фреймворк модель-перегляд-контролер (MVC) для Node.js, який дотримується принципу «конвенції щодо конфігурації». Веб-фреймворк Ruby on Rails надихає його, таким чином емулює знайомий шаблон MVC для створення односторінкових програм, API REST і програм у реальному часі. Широко використовує генератори коду, які дозволяють створювати програми з меншим написанням коду. Фреймворк побудовано на основі Socket.io, бібліотеки JavaScript для додавання до програм у режимі реального часу двонаправленого зв’язку на основі подій і Express.js, однієї з найпопулярніших бібліотек Node.js.

Derby.js

DerbyJS – це повностекова платформа з відкритим вихідним кодом для створення сучасних веб-додатків реального часу. Використовує PubSub і сумісний з будь-якою базою даних. Ми можемо використовувати NPM для додавання функцій і функцій до проекту Derby. Будь-яка бібліотека інших учасників не завантажується автоматично і не включається в Derby глобально, і

потрібно «вимагати», як це було б із будь-яким проектом node.js. Derby зосереджено на тому, щоб дозволити користувачам створювати веб-програми, які швидко завантажуються в реальному часі, і є гнучким і розширюваним. Шаблони можна надати в браузері та на сервері. У браузері DerbyJS відтворює за допомогою швидких, рідних методів DOM.

Особливості:

- Співпраця в реальному часі
- Рендеринг сервера
- Компоненти та зв'язування даних
- Модульний

Total.js

Total.js — це модульна і сучасна платформа Node.js трирічної давності, яка підтримує архітектуру MVC. Клієнтські фреймворки, такі як Angular.js, Polymer, Backbone.js, Bootstrap, повністю сумісні з цим фреймворком. Ця структура є розширюваною та асинхронною та забезпечує чудову продуктивність та стабільність. Будь-які інструменти, такі як Grunt, не потрібні, щоб стиснути його, простий у використанні. Він також має вбудований NoSql і підтримує масив та інші прототипи.

Особливості:

- Швидка підтримка та виправлення помилок
- Підтримує маршрутизацію RESTful
- Підтримує потокове відео
- Підтримує теми
- Підтримує працівників
- Підтримує карту сайту
- Підтримує WebSocket
- Підтримує моделі, модулі, пакети та ізоморфний код
- Підтримує обробку зображень через GM або IM
- Підтримує генератори

- Підтримує локалізацію за допомогою інструмента diff та експорту CSV
- Підтримує обмеження та перенаправлення

Adonis.js

Adonis — це фреймворк node.js, який має жорстку структуру MVC, яка є шаблоном проектування, де він розбиває певні функції на різні розділи додатків. Adonis використовує двигун шаблонів краю, який дуже простий у використанні.

Особливості:

- Він має власний CLI (інтерфейс командного рядка)
- Знайомий з Laravel, так легко навчитися
- Валідатори використовуються для перевірки правильного формату даних, що надходять до контролерів, і надсилання повідомлень у разі виникнення деяких помилок.

Nest.js

NestJS — це прогресивний фреймворк Node.js для створення ефективних, надійних і масштабованих додатків на стороні сервера, що допомагає розробникам створювати модульні, високомасштабовані та підтримувані веб-додатки на стороні сервера.

Він реалізує шаблон MVC (Model-View-Controller) і забезпечує розширюваність. Відмінною особливістю NestJS є вбудована підтримка TypeScript, яка дає вам доступ до додаткової статичної перевірки типів разом із потужними інструментами для великих програм і найновішими функціями ECMAScript.

Особливості:

- Розширюваний: дозволяє використовувати будь-які інші бібліотеки через модульну архітектуру, що робить його дійсно гнучким.
- Універсальний: він пропонує адаптивну екосистему, яка є повноцінною основою для всіх видів серверних програм.

- Прогресивний: вносить шаблони дизайну та складні рішення у світ node.js, використовуючи переваги найновіших функцій JavaScript.

LoopBack.js

LoopBack — це фреймворк Node.js із простим у використанні інтерфейсом командної команди та динамічним дослідником API. Це дозволяє створювати свої моделі на основі вашої схеми або динамічні моделі за відсутності схеми. Він сумісний з великою кількістю служб REST і широким спектром баз даних, включаючи MySQL, Oracle, MongoDB, Postgres тощо.

Це може дозволити користувачеві створювати серверний API, який відповідає іншому серверу, майже як створення API, який є проксі-сервером для іншого API. Це підтримка оригінальних пакетів SDK для мобільних пристроїв і браузерів для таких клієнтів, як Android/Java, iOS, JavaScript для браузера (Angular).

Особливості:

- Неймовірно розширюваний
- Підтримка графіка QL

Для розробки дипломного проекту було вибрано фреймворк Nest.js. Причиною цього стали наступні фактори:

- Швидкий та ефективний процес розробки.
- Високомасштабовані та прості в обслуговуванні програми.
- Найшвидший розвиток фреймворка Node.js за останні 3 роки.
- Велика спільнота та система підтримки.
- Використання TypeScript допомагає гарантувати, що Nest залишатиметься актуальним у JavaScript, що швидко змінюється, і дає розробникам менше перемикань контексту.
- Чудова документація.
- Легке модульне тестування.
- Створено для великомасштабних корпоративних додатків.

- Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати добре тестовані, масштабовані, слабо пов’язані та легко обслуговувані програми.
- За допомогою NestJS ви можете створювати Rest API, програми MVC, мікросервіси, програми GraphQL, веб-сокети або CLI та завдання CRON.
- Він використовує найкращі архітектурні шаблони Node.js, TypeScript.

2.3 Вибір системи управління базами даних

SQL - це мова. Це не частина програмного забезпечення – це програмний синтаксис, який використовується багатьма (але не всіма) програмними платформами баз даних для введення та отримання інформації, що зберігається в базі даних.

Хоча це переважно стандартне рішення, кожне рішення SQL-сервера, як правило, має свою власну реалізацію з невеликими відмінностями та відмінними функціями, тому ви не можете просто взяти SQL, написаний для однієї платформи, і запустити його безпосередньо на іншій без адаптації.

PostgreSQL — це мережеве рішення SQL клієнт/сервер, яке пропонує більш складні структури даних, ніж MySQL.

Це не означає, що він більш складний у використанні – ви можете робити з ним більше речей (якщо потрібно).

Останніми роками популярність PostgreSQL значно зросла.

Для чого це добре

- Все, що робить MySQL
- Складні запити
- Бізнес-аналіз та аналіз даних
- Одночасність – кілька користувачів швидко оновлюють одні й ті ж дані

Хто використовує PostgreSQL

PostgreSQL користується популярністю серед організацій, які займаються запитами великих даних.

- Робін Гуд
- Reddit
- IMDB

Завдяки його популярності серед натовпу Business Intelligence, в Інтернеті є багато матеріалів щодо використання PostgreSQL з мовою програмування Python.

SQLite

На відміну від більшості інших систем баз даних на цій сторінці, SQLite не працює як клієнт/сервер у мережі – бази даних є простим файлом, доступним у файловій системі.

Для забезпечення служби бази даних не потрібно запускати додаткове програмне забезпечення – додайте бібліотеку SQLite до свого проекту, наведіть її на локальний файл SQLite на вашому диску.

Це ідеально підходить для невеликих баз даних, локальної розробки або як сховище даних для настільних додатків.

Для чого це добре

- Невеликі бази даних
- Локальна розробка та тестування
- Ситуації, коли складні запити не важливі
- Настільні програми

Попросити клієнтів встановити сервер SQL перед встановленням вашого продукту може збити їх з пантелику – SQLite усуває потребу в цьому.

Завдяки тому, що дані ваших клієнтів в одному файлі на комп’ютері, їм дуже легко знайти свої дані та створити їх резервну копію.

Його також можна використовувати для веб-програм, коли вам потрібна лише проста база даних для публікацій у блозі або інших видів діяльності, де дані не записуються постійно.

Хто використовує SQLite?

Як я вже згадував, SQLite чудово підходить для настільних програм.

- Adobe
- Microsoft
- Firefox

Усі останні випуски мови програмування Python включають підтримку SQLite.

MongoDB

MongoDB не є програмою баз даних SQL, але це програма бази даних.

MongoDB не використовує мову запитів SQL для читання та запису даних – у нього є власна мова запитів – MQL (Mongo Query Language).

На відміну від перерахованих вище систем баз даних, MongoDB зберігає дані в документах , а не в таблицях .

Структура даних не дотримується жорстко, і ви можете в значній мірі скидати будь-які дані в документ без урахування його формату (поки вам не доведеться запитувати його).

Дані зберігаються в документах у форматі, подібному до JSON, що дозволяє зберігати складні об’єкти без розбиття даних на рядки та стовпці.

Для чого це добре

- Зберігання даних там, де ви можете не знати структуру заздалегідь
- Наприклад, зберігання деталей експозиції сучасного мистецтва: ви можете не знати, які атрибути буде мати кожен елемент заздалегідь, тому фіксовані стовпці таблиці не будуть доречними.
- Менший ризик приєднання через ін'єкції SQL
- Скидайте в нього дані та обробіть це пізніше

- Легко розпочати роботу, оскільки вам не доведеться турбуватися про структуру бази даних заздалегідь
- Це робить його хорошим для швидкої розробки там, де ви не вибрали жорстку структуру даних.

Хто використовує MongoDB

MongoDB користується популярністю серед організацій, які мають справу з поданими користувачами даними, які можуть не відповідати встановленій структурі.

- SAP
- EA
- eBay

Для виконання дипломного проекту було вирішено, аналізуючи вище сказані дані, використовувати СУБД Postgres, через його можливість одночасного підключення великої кількості користувачів, можливість робити складні запити.

2.4 Додаткове програмне забезпечення

Prisma — це об'єктно-реляційна проєкція нового покоління (ORM), яка, як стверджується, допомагає розробникам будувати швидше та допускати менше помилок. Prisma використовує інший підхід до ORM у порівнянні з традиційними ORM. Він використовує спеціальну мову визначення схем (SDL), яка автоматично записує міграції та генерує безпечний для типу код. На відміну від TypeORM і MikroORM, Prisma не використовує класи або декоратори для визначення моделі. Натомість він використовує генерацію коду зі схеми, подібно до популярної утиліти GraphQL Code Generator. Цей згенерований код TypeScript використовується клієнтом Prisma для суворішої безпеки типів і багатих функцій автозаповнення IDE.

Socket.IO — це керована подіями бібліотека JavaScript для веб-додатків у реальному часі . Він забезпечує двонаправлений зв’язок у реальному часі між веб-клієнтами та серверами. Він складається з двох частин: бібліотека на стороні клієнта, яка працює у браузері , і бібліотека на стороні сервера для Node.js. Обидва компоненти мають майже ідентичний API .

Socket.IO в основному використовує протокол WebSocket з опитуванням як запасний варіант , забезпечуючи той самий інтерфейс. Хоча його можна використовувати як просто обгортку для WebSockets, він надає багато інших функцій, включаючи трансляцію на кілька сокетів, зберігання даних, пов’язаних з кожним клієнтом, та асинхронний ввод-вивод .

Його можна встановити за допомогою диспетчера пакетів вузлів (NPM).

Socket.IO надає можливість реалізовувати аналітику в реальному часі, бінарну потокову передачу, обмін миттєвими повідомленнями та спільну роботу з документами. Серед відомих користувачів Microsoft Office , Yammer та Zendesk .

Socket.IO обробляє з'єднання прозоро і автоматично оновиться до WebSocket , якщо це можливо. Це означає, що розробнику не потрібно знати, як використовувати протокол WebSocket , щоб використовувати Socket.IO.

Socket.IO не є бібліотекою WebSocket із резервними параметрами до інших протоколів реального часу. Це індивідуальна реалізація транспортного протоколу в реальному часі поверх інших протоколів реального часу. Сервер, який реалізує Socket.IO, не може підключитися до клієнта WebSocket, який не є Socket.IO. Клієнт, що реалізує Socket.IO, не може спілкуватися з сервером WebSocket або Long Polling Comet , який не є Socket.IO . Socket.IO вимагає використання бібліотек Socket.IO як на стороні клієнта, так і на стороні сервера.

Docker — це відкрита платформа для розробки, доставки та запуску програм. Docker дає змогу відокремити ваші програми від інфраструктури, щоб ви могли швидко постачати програмне забезпечення. За допомогою Docker ви можете керувати своєю інфраструктурою так само, як і своїми програмами.

Використовуючи переваги методологій Docker для швидкої доставки, тестування та розгортання коду, ви можете значно зменшити затримку між написанням коду та його запуском у виробництві.

Docker надає можливість пакувати та запускати програму в слабо ізольованому середовищі, яке називається контейнером. Ізоляція та безпека дозволяють запускати багато контейнерів одночасно на даному хості. Контейнери легкі та містять все необхідне для запуску програми, тому вам не потрібно покладатися на те, що в даний момент встановлено на хості. Ви можете легко ділитися контейнерами під час роботи, і будьте впевнені, що всі, з ким ви ділитеся, отримають той самий контейнер, який працює так само.

Docker спрощує життєвий цикл розробки, дозволяючи розробникам працювати в стандартизованих середовищах, використовуючи локальні контейнери, які надають ваші програми та послуги. Контейнери чудово підходять для безперервної інтеграції та безперервної доставки (CI/CD) робочих процесів.

Docker легкий і швидкий. Він надає життєздатну, економічно ефективну альтернативу віртуальним машинам на основі гіпервізора, тому ви можете використовувати більше своїх обчислювальних потужностей для досягнення ваших бізнес-цілей. Docker ідеально підходить для середовищ з високою щільністю, а також для малих і середніх розгортань, де вам потрібно робити більше з меншими ресурсами.

Висновок до розділу 2

У цьому розділі розглядаються найпопулярніші програми для розробки веб-сайтів і веб-додатків. Також було обрано основне програмне забезпечення для розробки власного програмного продукту. Також було описано додаткові програмні продукти які використовувалися в розробці дипломного проекту.

3 ВЛАСНЕ ОРИГІНАЛЬНЕ АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ РІШЕННЯ ЗАДАЧІ

3.1 Складові програмного рішення

Назва кінцевого продутку – Qchat.

Для розробки даного проекту потрібно було обрати СУБД, вибір базувався на наступних характеристика і вимогах до СУБД:

1) СУБД повинна бути реляційною

2) СУБД повинна бути швидкою

3) Хороша оптимізація для більшості операцій

4) Можливість працювати з обраною серверною технологією, Node.js в даному випадку

5) Можливість її подальшого запуску в Docker

За даним списком вимог було обрано СУБД Postgres, як одну з найбільш популярних та найчастіше використовуваних СУБД на даний час в світі.

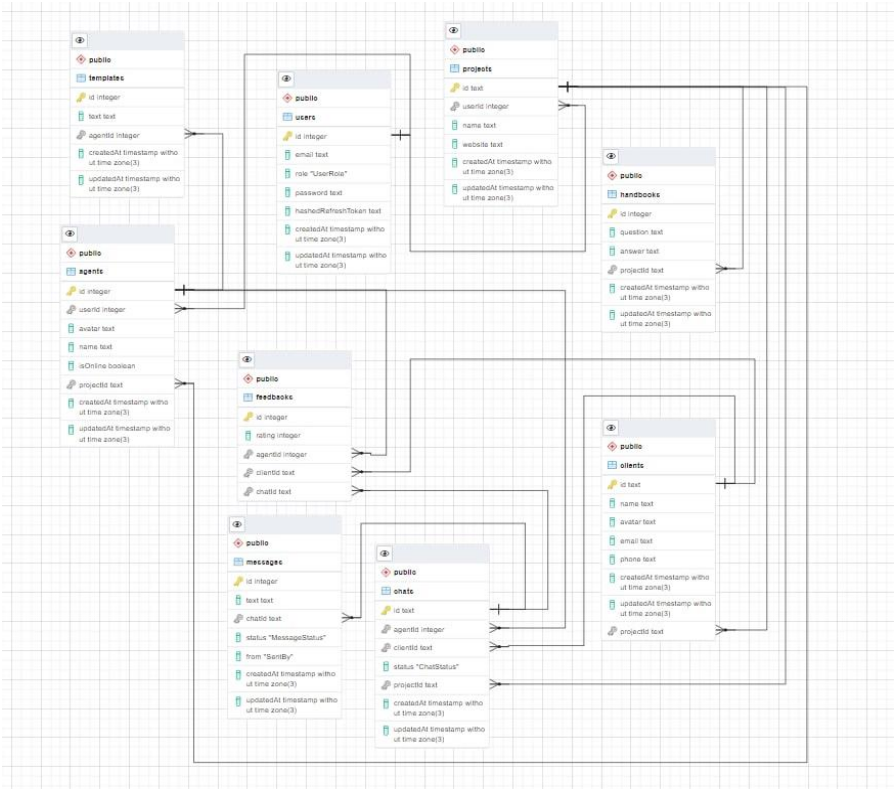


Рисунок 3.1 – Схема БД

Як можна бачити на Рисунку 3.1, база даних складається з кількох таблиць "Users", "Projects", "Agents", "Chats", "Clients", "Messages", "Feedbacks", "Templates", "Handbooks".

Таблиця Users включає в себе всю необхідну інформацію про користувачів. "email" та "password" використовуються для при авторизації, на одну електронну пошту може бути зареєстрований тільки один обліковий запис. І колонка "role", котра може мати одне з значень: Агент – AGENT, Проект– PROJECT.(Рисунок 3.2)

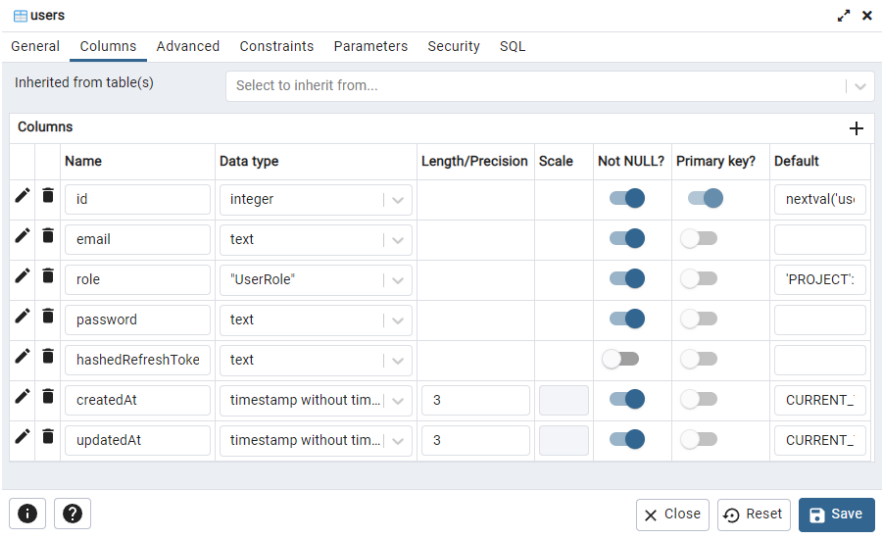


Рисунок 3.2 – Таблиця users

В таблиці "Projects" зберігається інформація про проекти (Рисунок 3.3)

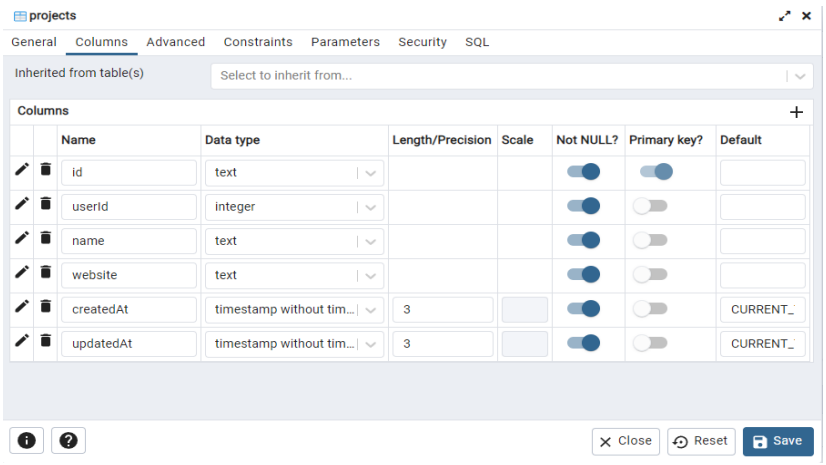


Рисунок 3.3 – Таблиця Projects

В таблиці Agents інформація про агентів (Рисунок 3.4)

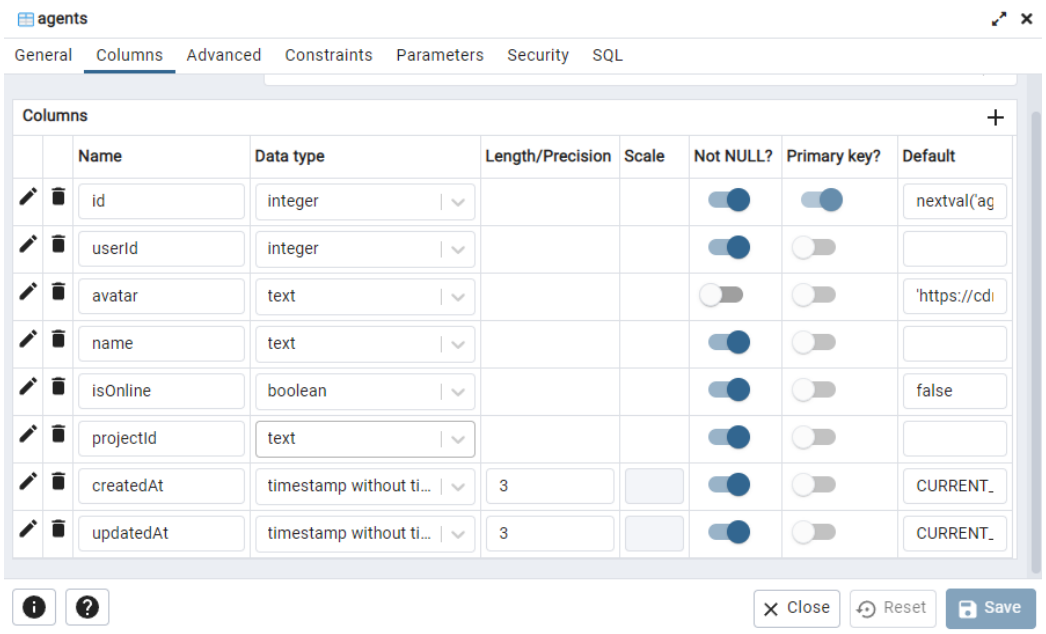


Рисунок 3.4 – Таблиця Agents

В таблиці Chats зберігається інформація про чати, вона також повязана з таблицею Projects, Agents, Clients.(Рисунок 3.5)

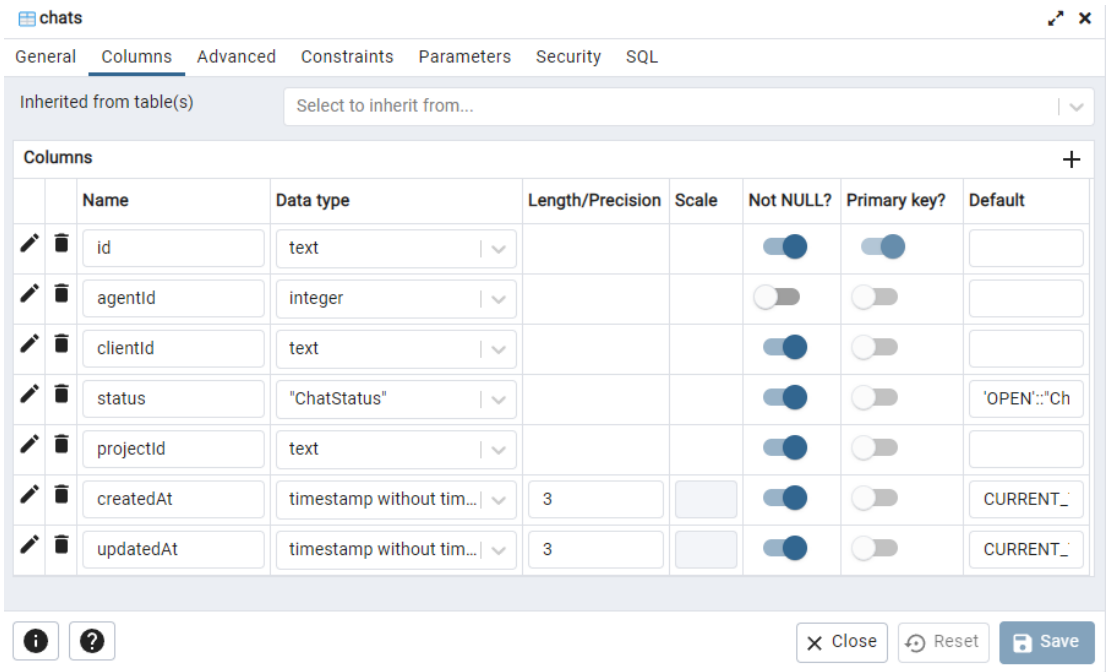


Рисунок 3.5 – Таблиця Chats

Clients таблиця де зберігаються дані про клієнтів.(Рисунок 3.6)

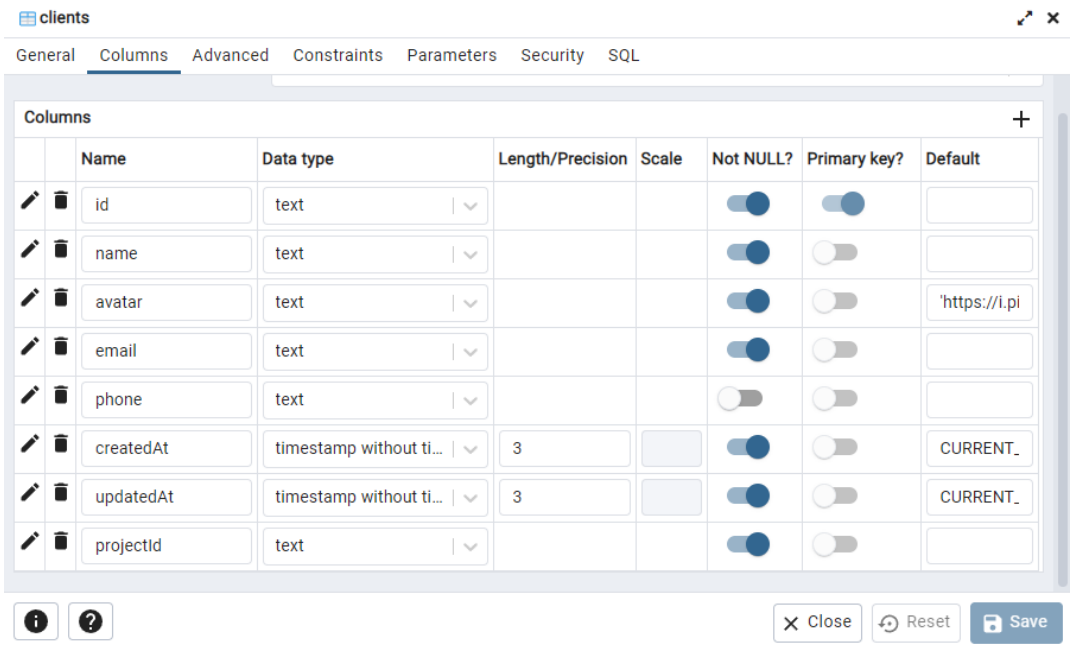


Рисунок 3.6 – Таблиця Clients

Messages таблиця де зберігаються повідомлення, вона також повязана з таблицею чати. (Рисунок 3.7)

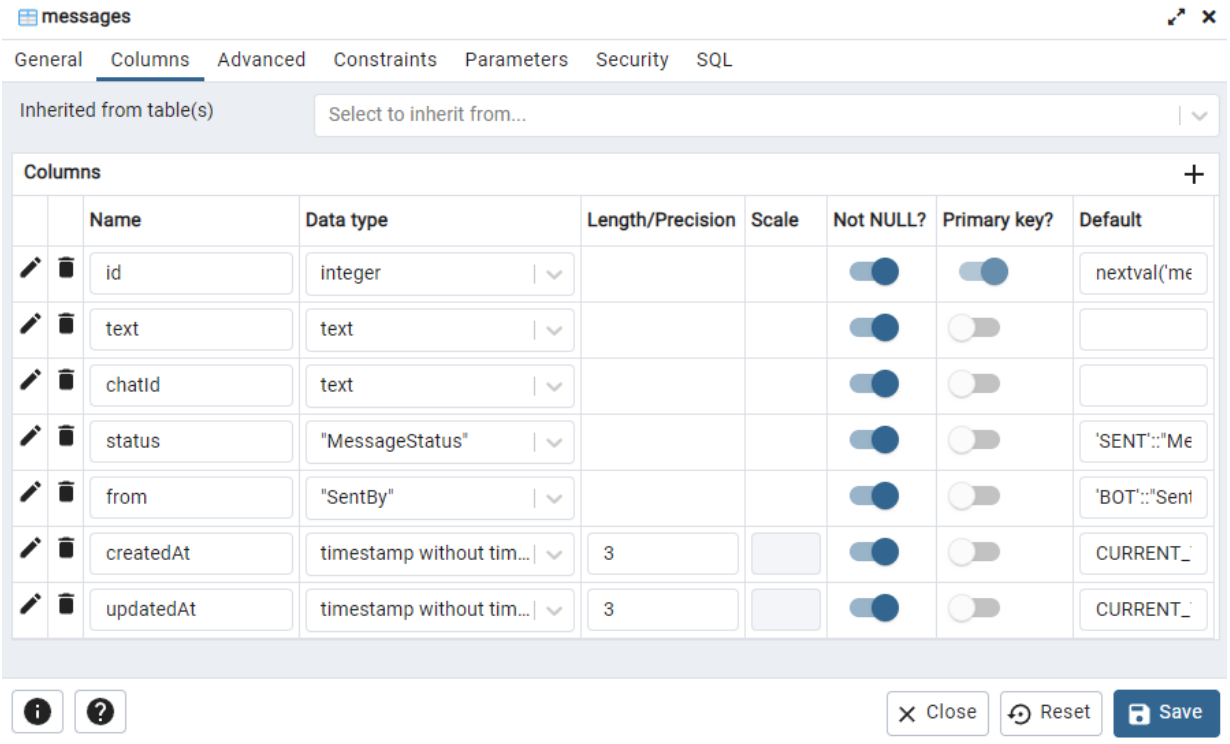


Рисунок 3.7 – Таблиця Messages

Таблиці Feedbacks, Templates, Handbooks, служать для додаткового функціоналу такого як, рейтинги, швидкі відповіді, і часті запитання.

На Рисунку 3.8 можна бачити види використаних зв’язків між таблицями. Дана база даних на момент розробки перебувала в локальному середовищі, а по закінченню розробки була перенесена в засіб контейнеризації і віртуалізації Docker, про який згадувалось раніше.

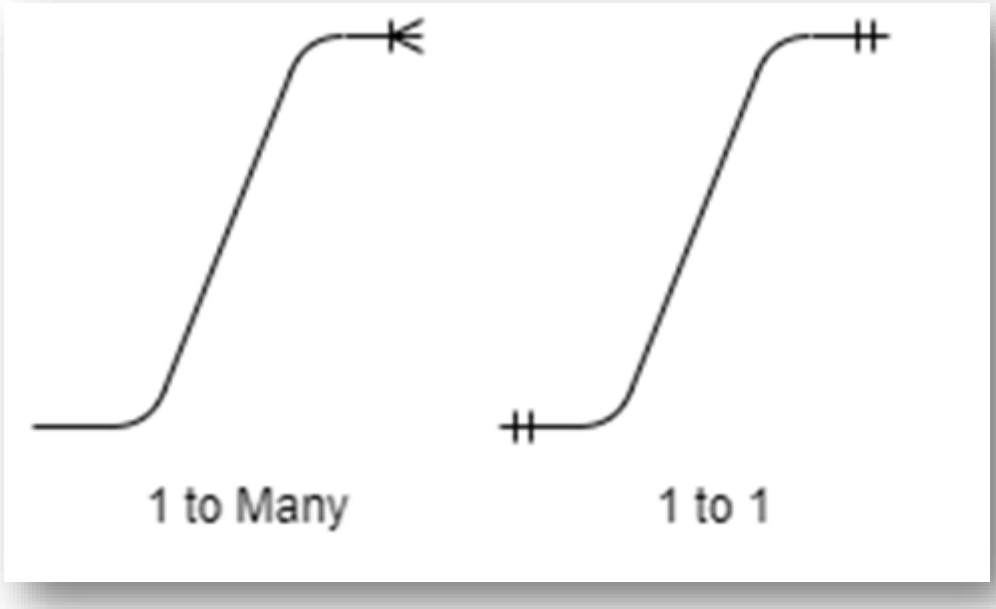


Рисунок 3.8 – Види використаних реляцій

3.1.1 Back-end

Причини вибору Node.js, Typescript і Nest.js Framework:

Крос-платформенність

Node.js не привязаний до конкретної платформи. Ви можете розмістити його на Linux, AWS або Google Cloud. Він сумісний з Windows, macOS та Linux. Його також можна запускати в інших ОС.

Швидкодія

Node.js є асинхронним, однопоточним, але з вбудованою технологію Event Loop, яка забезпечується бібліотекою на C++ libuv та двигуном JavaScript, який

був розроблений компанією Google V8. Це дозволяє йому обробляти велику кількість одночасних запитів, максимально ефективно працювати не тільки з Real Time додатками, а й з звичайними REST додатками.

Просте обслуговування додатків

Оскільки TypeScript є типізованою мовою програмування, більшість помилок можна виявити на ранній стадії за допомогою засобів розробки. Visual Studio Code допомагає підтримувати кодову базу. Це спрощує залучення нових розробників і робить початок проекту досить простим, також дозволяє без зайвих проблем підтримувати і розвивати проект.

Підтримка веб-API

Досить просто створити API для підключення зовнішньої та зовнішньої систем та постійно оновлювати документацію щодо API.

Веб-API є частиною фреймворку і може бути легко інтегрований із Swagger. Під час створення додатків документація є їх важливою складовою. Це спрощує роботу з вашим продуктом для інших розробників. Swagger можна легко інтегрувати, а документація легко описується в коді за допомогою декораторів.

Типізація

На відміну від JavaScript, Typescript є типізованою мовою програмування,

він має сильну систему типів яка дозволяє описувати будь-які необхідні структури даних, та виявляти більшість помилок ще на етапів розробки,а також простіше підтримувати додаток в майбутньому.

Фреймворк – як база готових рішень

Nest.js – з коробки має багато готових рішень, як то валідація, роутинг, Dependency Injection, підтримка більшості популярних технологій, та величезна кількість готових модулів для більшості задач які виникають при розробці Back-End частини.

Масштабування та Dockerization

Node.js дуже просто можна запустити в Докер. Це спрощує контейнеризацію, масштабування та створення інфраструктури. Що дозволяє збільшити можливості бекенду при потребі в його масшатабуванні для обслуговування більшої кількості клієнтів або більшої пропускої здатності.

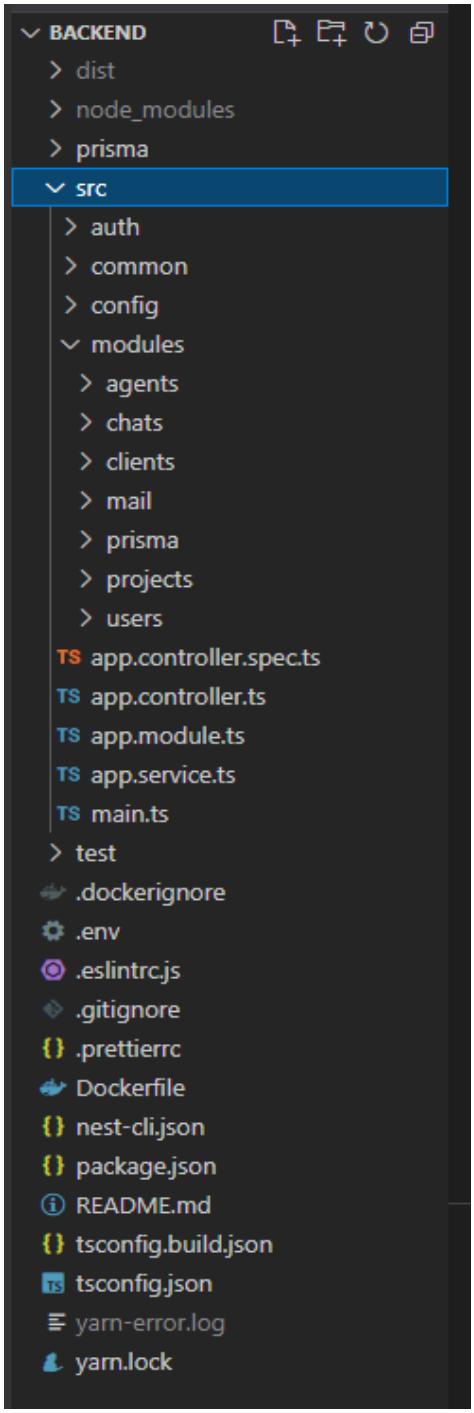


Рисунок 3.9 – Back-end, Структура проекту

Як можна бачити на рисунку 3.9, проект складається з кількох основних частин

- Prisma
- Auth
- Config
- Modules

Prisma - це папка де зберігається основні частина бази даних, такі як опис структури, та певні конфігураційні файли, Prisma це ORM для Node.js, яка дозволяє працювати з базою даних при перетворенні її сутностей на об’єкти Object Relational Mapping. Основна робота якою займається Prisma:

- **Запити**: Перетворює запити у SQL-запити та надсилає їх до бази даних.
- **Відстеження змін**: Відстежує зміни, що відбулися в сутності після запиту з бази даних.
- **Збереження даних**: виконує операції оновлення, вставки та видалення з бази даних на основі станів сутності.
- **Кешування**: За замовчуванням забезпечує кешування першого рівня. Він зберігає сутності, отримані протягом життя контекстного класу.
- **Керувати взаємозв'язками**: керує взаємозв'язками в базі даних.
- **Матеріалізація об’єктів**: перетворює необроблені дані з бази даних в об’єкти сутності.

Приклад синтаксису оголошення бази даних за допомогою Prisma.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
```

```
  id                    Int       @id @default(autoincrement())
  email                 String    @unique
  role                  UserRole  @default(PROJECT)
  user                  Agent?
  project               Project?
  password              String
  hashedRefreshToken    String?
  createdAt             DateTime  @default(now())
  updatedAt             DateTime  @default(now())

  @@map("users")
}

model Agent {
  id          Int       @id @default(autoincrement())
  user        User      @relation(fields: [userId],
references: [id])
  userId      Int
  avatar      String?   @default("https://cdn-icons-
png.flaticon.com/512/236/236832.png")
  name        String
  chats       Chat[]
  project     Project   @relation(fields: [projectId],
references: [id])
  isOnline    Boolean   @default(false)
  projectId   String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @default(now())
  feedback    Feedback[]
  template    Template[]

  @@map("agents")
}
```

Auth – в цій папці зберігаютьсься файли які використовуються для авторизації та автентифікації користувачів в системі. Для авторизації було обрано відносно новий спосіб авторизації за допомого JSON Web Token.

JSON Web Token (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити й довіряти, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секретного (за допомогою алгоритму HMAC) або пари відкритих і закритих ключів за допомогою RSA або ECDSA.

Хоча JWT можна зашифрувати, щоб також забезпечити секретність між сторонами, ми зосередимося на підписаних токенах. Підписані маркери можуть перевірити цілісність заяв, що містяться в них, тоді як зашифровані токени приховують ці претензії від інших сторін.

Коли токени підписуються за допомогою пар відкритих/приватних ключів, підпис також засвідчує, що підписала його лише сторона, яка володіє закритим ключем.

В системі використовується 2 токени, Access Token та Refresh Token.

Access Token – має обмежений час життя та використовується для доступу до системи користувача, та для авторизації і автентифікації його.

Refresh Token – має набагато більший час життя та використовується для отримання нового Access Token коли у попереднього закінчився час життя.

Приклад генерації токенів

```
async getTokens(user: Partial<User>) {
    const data = {
      id: user.id,
      email: user.email,
      role: user.role,
    };

    const [accessToken, refreshToken] = await Promise.all([
      this.jwtService.signAsync(data, {
        secret:
this.configService.get('ACCESS_TOKEN_SECRET'),
        expiresIn: '1h',
      }),

      this.jwtService.signAsync(data, {
        secret:
this.configService.get('REFRESH_TOKEN_SECRET'),
        expiresIn: '14d',
      }),
    ]);

    return {
      accessToken,
      refreshToken,
    };
  }

  async updateUserRefreshToken(userId: number, refreshToken:
string) {
```

```
        const hashedRefreshToken = await
bcrypt.hash(refreshToken, 10);

        await this.prisma.user.update({
          where: {
            id: userId,
          },
          data: {
            hashedRefreshToken,
          },
        });
      }
```

Config – в даній папці знаходиться сервіс для отримання даних середовища, і змінних які можуть бути різними в залежності від режиму запуску додатка, як то наприклад Production, Development, Staging. Цей сервіс дозволяє запускати додаток в різних середовищах без потреби кожного разу змінювати код додатку, а лише міняти самі змінні середовища.

Приклад змінних середовища які використовуються в додатку.

```
NODE_ENV=production
DATABASE_URL="postgres://postgres:12345@postgres:5432/qchat?s
chema=public"
REFRESH_TOKEN_SECRET=refresh-token-secret
ACCESS_TOKEN_SECRET=access-token-secret
POSTGRES_DB=qchat
POSTGRES_USER=postgres
POSTGRES_PASSWORD=12345
```

Сервіс ConfigService

```
@Injectable()
export class ConfigService {

  constructor() {
    const { error } = config();
    if (error) throw error;
  }

   @param  key
   @param  filename
   @return

  public getDest(key: string, filename?: string): string {
    filename = process.env[filename] || filename || '/';
    return join(process.cwd(), this.get(key), filename);
  }
   @param  mode
```

```
         @return

        public getMode(mode: ConfigMode): boolean {
          return process.env['NODE_ENV'] === mode;
        }

         @param  key
         @return

        public get<T = NodeJS.ProcessEnv>(key: string): T {
          const variable = process.env[key];
          if (!variable) throw TypeError(`The ${key} cannot be
undefined`);
          try {
            return JSON.parse(variable);
          } catch {
            return <T>(<unknown>variable);
          }
        }
      }
```

Modules – в цій папці знаходяться основні модулі системи. Всі під-папки тут в основному мають однакову структуру.

Файл *.controller.ts – файл який відповідає за обробку ресурсів та запитів від клієнта. Він обробляє стандартні методи протоколу HTTP, такі як GET, POST, PATCH, UPDATE, DELETE.

Файл *.service.ts – файл який містить в собі всі методи та логіку для даної сутності, взаємодію з базою даних, та специфічну логіку обробку даних.

Файл *.module.ts – файл який відповідає за об’єднання Сервісу і Контролера в один спільний модуль, для подальшого його зручного використання в системі.

Папка dto – папка яка містить в собі DTO, Data Transfer Object (DTO) — один із шаблонів проектування, який використовують для передачі даних між підсистемами програми, а також для їх валідації та обробки помилок даних і сповіщення клієнта про невірні дані, в даному випадку між Front-end та Back-end частинами системи.

Папка enums(Рисунок 3.10) – папка які містить в собі певні константи які використовуються додатком.

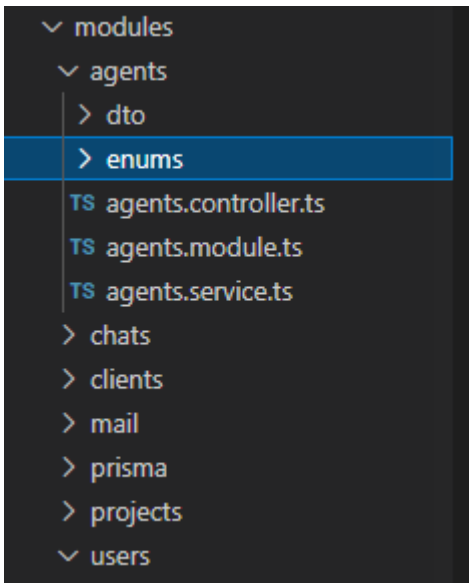


Рис 3.10 – Структура папки modules

Agents – папка яка містить в controller, service і модуль, для читання, видалення, оновлення та інших операцій над сутністю Agents.

Приклад контролеру для сутності Agents.

```
@Controller('agents')
export class AgentsController {
  constructor(private agentService: AgentsService) {}

  @Get('/profile')
  async getProfile(@GetCurrentUser('id') id: number) {
    return this.agentService.getProfile(id);
  }

  @Patch('/profile')
  async updateProfile(
    @GetCurrentUser('id') id: number,
    @Body() dto: UpdateAgentDto,
  ) {
    return this.agentService.updateProfile(id, dto);
  }

  @Get('/templates')
  async getTemplate(@GetCurrentUser('id') id: number) {
    return this.agentService.getTemplatesForAgent(id);
  }

  @Post('/templates')
  async createTemplate(
    @GetCurrentUser('id') id: number,
    @Body() dto: CreateTemplateDto,
  ) {
```

```
    return this.agentService.createTemplateForAgent(id, dto);
  }

  @Patch('/templates')
  async updateTemplate(
    @GetCurrentUser('id') id: number,
    @Body() dto: UpdateTemplateDto,
  ) {
    return this.agentService.updateTemplate(dto.templateId,
dto.text);
  }

  @Delete('/templates/:id')
  async deleteTemplate(@Param('id', ParseIntPipe) id: number)
{
    return this.agentService.deleteTemplate(id);
  }

  @Patch('/online-status')
  async updateOnlineStatus(
    @GetCurrentUser('id') id: number,
    @Body() dto: UpdateOnlineStatusDto,
  ) {
    return this.agentService.updateOnlineStatus(id,
dto.status);
  }
```

Chats – папка яка містить в собі логіку роботи з чатами, для додатку, з незвичного тут міститься файл chats.gateway.ts, який обробляє WebSocket підключення.

WebSocket — це протокол, призначений для обміну інформацією між браузерами та веб-серверами в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket розроблено для реалізації у веб-браузерах і веб-серверах, але також може використовуватися будь-яким клієнт-серверним додатком. API WebSocket стандартизовано W3C, а протокол WebSocket стандартизований IETF як RFC 6455. [1] У веб-додатках цей протокол рекомендується використовувати, якщо інформацію потрібно відображати в режимі реального часу. Альтернативна технологія – події, надіслані сервером.

Щоб встановити WebSocket-з'єднання, клієнт надсилає handshake-запит — так званий запит на встановлення довіри, своєрідне, «цифрове рукостискання».

Клієнт також надсилає свій відкритий ключ `Sec-WebSocket-Key` для шифрування повідомлень для нього. Відкритий ключ в секції параметрів HTTP-запиту кодується в форматі base64.

Для зручної роботи з вебсокетами було обрано бібліотеку socket.io.

Бібліотека дозволяє швидко і зручно, налаштувати роботу з веб-сокетами, та підтримувати сталу архітектуру додатку.

Переваги даної бібліотеки, які розробники самі описують в документації :

- Двонаправлений зв’язок із низькими затримками для кожної платформи.
- У більшості випадків з'єднання буде встановлено за допомогою WebSocket, забезпечуючи канал зв'язку між сервером і клієнтом з низькими витратами.
- Будьте впевнені! Якщо підключення WebSocket неможливе, воно повернеться до тривалого опитування HTTP. І якщо з’єднання буде втрачено, клієнт автоматично спробує знову підключитися.
- З легкістю масштабуйте на кілька серверів і надсилайте події всім підключеним клієнтам.

Приклад коду з файлу chat.gateway.ts

```
@SubscribeMessage(ChatEvents.ClientJoinChat)
  async onClientJoined(socket: Socket, data: CreateChatDto) {
    const chat = await this.chatService.createChat(data);

    socket.data.client = chat.client;

    await socket.join(chat.id);

    await this.emitToAgents(ChatEvents.ChatOpen, chat);

    socket.emit(ChatEvents.ChatOpen, chat);
  }

  @SubscribeMessage(ChatEvents.ClientSendMessage)
  async onClientSendMessage(_: Socket, data) {
    const chat = await
this.chatService.findById(data.chatId);
```

```
          const message = await
this.chatService.sendMessageToChat({
            chatId: chat.id,
            text: data.text,
            from: 'CLIENT',
          });

          this.server.to(chat.id).emit(ChatEvents.ChatMessage,
message);
        }

        @SubscribeMessage(ChatEvents.ClientSeenMessage)
        async onClientSeenMessage(_: Socket, data) {
          const updatedChat = await
this.chatService.markMessagesAsSeen(
            data.chatId,
            'AGENT',
          );

          const chat = await
this.chatService.findById(data.chatId);

          this.server.to(chat.id).emit(ChatEvents.AgentChatUpdated,
updatedChat);
        }

        @SubscribeMessage(ChatEvents.AgentJoinChat)
        async onAgentJoined(socket: Socket, data) {
          const chat = await
this.chatService.findById(data.chatId);

          await socket.join(chat.id);

          const updatedChat = await
this.chatService.joinAgentToChat(
            data.chatId,
            socket.data.agent.id,
          );


this.server.to(chat.id).emit(ChatEvents.ClientChatUpdated,
updatedChat);
          this.server.to(chat.id).emit(ChatEvents.AgentChatUpdated,
updatedChat);

          await this.emitToAgents(ChatEvents.AgentTakeChat,
updatedChat);
        }

        @SubscribeMessage(ChatEvents.AgentSendMessage)
        async onAgentSendMessage(_: Socket, data) {
```

3.1 Розбір функціоналу програмного рішення

Авторизація побудована на основі JWT. JWT це відносно новий спосіб авторизації, який побудований на основі 2 ключів – токенів. Access Token – токен для авторизації в системі, має обмежений час життя, та може містити в собі додаткові дані.

Refresh Token – токен для оновлення Access Token, має набагато більший час життя.

Обидва токени підписуються секретними ключами, і зчитуються також за допомогою них.

Для реалізації авторизації використовувались бібліотеки

- nestjs/jwt
- nestjs/passport
- passport-jwt

Для реалізації стратегій авторизації було створено 2 файли стратегій.

Код з файлу access-token.strategy.ts:

```
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { ConfigService } from 'src/config';

@Injectable()
export class AccessTokenStrategy extends
PassportStrategy(Strategy, 'jwt') {
  constructor(private configService: ConfigService) {
    super({
      jwtFromRequest:
ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: configService.get('ACCESS_TOKEN_SECRET'),
    });
  }

  validate(payload: any) {
    return payload;
  }
}
Код з файлу refresh-token.strategy.ts:
import { Injectable } from '@nestjs/common';
```

```
import { PassportStrategy } from '@nestjs/passport';
import { Request } from 'express';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { ConfigService } from 'src/config';

@Injectable()
export class RefreshTokenStrategy extends PassportStrategy(
  Strategy,
  'jwt-refresh',
) {
  constructor(private configService: ConfigService) {
    super({
      jwtFromRequest:
ExtractJwt.fromBodyField('refreshToken'),
      secretOrKey: configService.get('REFRESH_TOKEN_SECRET'),
      passReqToCallback: true,
    });
  }

  validate(req: Request, payload: any) {
    const refreshToken = req.body.refreshToken;

    return {
      ...payload,
      refreshToken,
    };
  }
}
```

Вони отримують токени з запиту від клієнта, та валідують правильність токенів, а тобто перевіряють що такий користувач справді має доступ до системи.

Також було створено два файли захисту. Вони розширяються за допомогою бібліотеки nestjs/passport, та отримують необхідний функціонал для перевірки, токенів на валідність.

Код файлу acess-token.guard.ts:

```
import { ExecutionContext, Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { AuthGuard } from '@nestjs/passport';
import { Observable } from 'rxjs';
@Injectable()
export class AccessTokenGuard extends AuthGuard('jwt') {
constructor(private reflector: Reflector) {
super();
}
canActivate(context: ExecutionContext,):
boolean | Promise<boolean> | Observable<boolean>
```

```
{
const isPublic = this.reflector.getAllAndOverride(
'isPublic',context.getHandler(),
context.getClass(),
]
);
if (isPublic) {
return true;
}
return super.canActivate(context);}}
```

Код файлу refresh-token.guard.ts:

```
import { AuthGuard } from '@nestjs/passport';

export class RefreshTokenGuard extends AuthGuard('jwt-refresh') {
  constructor() {
    super();
  }
}
```

Також було створено контролер, який відповідає за обробку http-запитів, та їх роутинг, та виклик відповідних функцій з AuthService.

Роути або http-ресурси:

- /auth/local/sign-up – шлях для реєстрації нових користувачів
- /auth/local/sign-in – шлях для входу користувачів та отримання refresh-token
- /auth/refresh-tokens – шлях для оновлення застарілого Access Token,
- /auth/logout – шлях для виходу з системи, і видалення Acess Token,
- /auth/me – шлях для авторизованих користувачів для отримання інформації про поточного користувача

Код файлу auth.controller.ts:

```
import {
  Body,
  Controller,
  Get,
  HttpCode,
  HttpStatus,
  Post,
  UseGuards,
} from '@nestjs/common';
import { Public } from 'src/common/decorators';
import { GetCurrentUser } from 'src/common/decorators/user.decorator';
import { AuthService } from './auth.service';
```

```
import { SignInDto, SignUpDto } from './dto';
import { RefreshTokenGuard } from './guards';

@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Public()
  @Post('/local/sign-up')
  @HttpCode(HttpStatus.CREATED)
  signUp(@Body() dto: SignUpDto) {
    return this.authService.signUp(dto);
  }

  @Public()
  @Post('/local/sign-in')
  @HttpCode(HttpStatus.OK)
  signIn(@Body() dto: SignInDto) {
    return this.authService.signIn(dto);
  }

  @Public()
  @UseGuards(RefreshTokenGuard)
  @Post('/refresh-tokens')
  @HttpCode(HttpStatus.OK)
  refreshTokens(@GetCurrentUser() user) {
            return     this.authService.refreshTokens(user.id,
user.refreshToken);
  }

  @Post('/logout')
  @HttpCode(HttpStatus.OK)
  logout(@GetCurrentUser('id') userId: number) {
    return this.authService.logout(userId);
  }

  @Get('/me')
  @HttpCode(HttpStatus.OK)
  getMe(@GetCurrentUser() user) {
    return this.authService.getMe(user.id);
  }
}
```

В файлі auth.service.ts міститься вся логіка авторизації:

AuthService містить наступні методи:

- signUp – логіка реєстрації нових користувачів
- signIn – логіка для входу користувачів та отримання access і refresh-token
- refreshTokens – логіка для оновлення застарілого Access Token,

- logout – логіка для виходу з системи, і видалення Acess Token,
- me – логіка для авторизованих користувачів для отримання інформації про поточного користувача

Код з файлу auth.service.ts:

```
import {
  ConflictException,
  ForbiddenException,
  Injectable,
} from '@nestjs/common';
import { PrismaService } from
'../modules/prisma/prisma.service';
import { SignInDto, SignUpDto } from './dto';
import * as bcrypt from 'bcryptjs';
import { JwtService } from '@nestjs/jwt';
import { User } from '@prisma/client';
import { ConfigService } from 'src/config';
import { ErrorMessages } from 'src/common/enums/error-
messages';

@Injectable()
export class AuthService {
  constructor(
    private prisma: PrismaService,
    private jwtService: JwtService,
    private configService: ConfigService,
  ) {}

  async signIn(dto: SignInDto) {
    const user = await this.prisma.user.findUnique({
      where: {
        email: dto.email,
      },
    });

    if (user) {
      const isPasswordValid = await
bcrypt.compare(dto.password, user.password);

      if (isPasswordValid) {
        const { accessToken, refreshToken } = await
this.getTokens(user);

        await this.updateUserRefreshToken(user.id,
refreshToken);

        return {
          accessToken,
          refreshToken,
          role: user.role,
```

```
            };
          } else {
            throw new
ForbiddenException(ErrorMessages.InvalidCredentials);
          }
        } else {
          throw new
ForbiddenException(ErrorMessages.InvalidCredentials);
        }
      }
```

Цей метод відповідає за аторизацію користувача. Він перевіряє наявність емейлу в базі даних та звіряє хеш-паролі. Після чого, якщо все вірно, повертає токени для доступу до системи.

```
      async signUp(dto: SignUpDto) {
        const isUserExists = await this.prisma.user.findUnique({
          where: { email: dto.email },
        });

        if (isUserExists) {
          throw new
ConflictException(ErrorMessages.EmailAlreadyExists);
        }

        try {
          const password = await bcrypt.hash(dto.password, 10);

          const user = await this.prisma.user.create({
            data: {
              email: dto.email,
              password: password,
              role: 'PROJECT',
              project: {
                create: {
                  name: dto.name,
                  website: dto.website,
                },
              },
            },
          });


          const { accessToken, refreshToken } = await
this.getTokens(user);

          await this.updateUserRefreshToken(user.id,
refreshToken);

          return { accessToken, refreshToken };
        } catch (e) {
          return e;
```

```
        }
      }

      async refreshTokens(userId: number, refreshToken: string) {
        const user = await this.prisma.user.findUnique({
       where: {
            id: userId,
          },
        });

        if (user && user.hashedRefreshToken) {
          const isValidRefreshToken = await bcrypt.compare(
            refreshToken,
            user.hashedRefreshToken,
          );

          if (isValidRefreshToken) {
            const { accessToken, refreshToken } = await
this.getTokens(user);

            await this.updateUserRefreshToken(user.id,
refreshToken);

            return {
              accessToken,
              refreshToken,
            };
          } else {
            throw new
ForbiddenException(ErrorMessages.InvalidRefreshToken);
          }
        } else {
          throw new
ForbiddenException(ErrorMessages.InvalidRefreshToken);
        }
      }
```

Prisma ORM дозволяє нам легко писати запити до баз даних, використовуючи простий об’єктний синтаксис для доступу до баз даних.

```
      async logout(userId: number) {
        await this.prisma.user.updateMany({
          where: {
            id: userId,
            hashedRefreshToken: {
              not: null,
            },
          },
          data: {
            hashedRefreshToken: null,
```

```
        },
      });
    }

    async getTokens(user: Partial<User>) {
      const data = {
        id: user.id,
        email: user.email,
        role: user.role,
      };

      const [accessToken, refreshToken] = await Promise.all([
        this.jwtService.signAsync(data, {
          secret:
this.configService.get('ACCESS_TOKEN_SECRET'),
          expiresIn: '1h',
        }),

        this.jwtService.signAsync(data, {
          secret:
this.configService.get('REFRESH_TOKEN_SECRET'),
          expiresIn: '14d',
        }),
      ]);

      return {
        accessToken,
        refreshToken,
      };
    }

    async updateUserRefreshToken(userId: number, refreshToken:
string) {
      const hashedRefreshToken = await
bcrypt.hash(refreshToken, 10);

      await this.prisma.user.update({
        where: {
          id: userId,
        },
        data: {
          hashedRefreshToken,
        },
      });
    }

    async getMe(userId) {
      const user = await this.prisma.user.findUnique({
        where: {
          id: userId,
        },
        select: {
```

```
            id: true,
            email: true,
            role: true,
          },
        });

        if (user.role === 'AGENT') {
          const agent = await this.prisma.agent.findUnique({
            where: {
              userId: user.id,
            },
          });

          const project = await this.prisma.project.findUnique({
            where: {
              id: agent.projectId,
            },
          });

          return {
            ...user,
            agent,
            project,
          };
        } else {
          const project = await this.prisma.project.findUnique({
            where: {
              userId: user.id,
            },
          });

          return {
            ...user,
            project,
          };
        }
      }
    }
```

Функціонал чату було реалізовано за допомогою технології WebSockets.

WebSockets – технологія при якій встановлюється постійне дводуплексне з’єднання, через яке як сервер так і клієнт може відправляти повідомлення, це відносно нова технологія для реалізації Real Time Apps.

Для реалізації функціоналу було використано також наступні бібліотеки.

- @nestjs/websockets

- socket.io

Основна частина функціоналу розташована в файлі chat.gateway.ts, а саме в класі ChatGatewey.

```
@WebSocketGateway({ cors: true })
export class ChatGateway implements OnGatewayConnection,
OnGatewayDisconnect {
  @WebSocketServer() server: Server;

  constructor(
    private chatService: ChatService,
    private jwtService: JwtService,
    private prisma: PrismaService,
  ) {}

  async onClientJoin(socket: Socket, clientId: string) {
    const client = await this.prisma.client.findUnique({
      where: { id: clientId },
    });

    if (client) {
      socket.data.client = client;

      const chat = await this.prisma.chat.findFirst({
        where: {
          clientId: client.id,
          OR: [{ status: 'OPEN' }, { status: 'ACTIVE' }],
        },
        include: {
          client: true,
          messages: {
            orderBy: {
              createdAt: 'asc',
            },
          },
          agent: true,
        },
      });

      if (chat) {
        await socket.join(chat.id);

        socket.emit(ChatEvents.ClientSetChat, chat);
      }
    }
  }

  async onAgentJoin(socket: Socket, agentId: number) {
    const chats = await
this.chatService.findActiveChatsForAgent(agentId);
```

```
      const agent = await this.prisma.agent.findUnique({
        where: {
          userId: agentId,
        },
        select: {
          id: true,
          name: true,
          avatar: true,
          userId: true,
        },
      });

      socket.data.agent = agent;

      for await (const chat of chats) {
        socket.join(chat.id);
      }
    }

    async emitToAgents(event: string, data: unknown) {
      const sockets = await this.server.fetchSockets();

      const agentsSockets = sockets
        .filter((socket) => socket.data.agent)
        .map((socket) => socket.id);

      this.server.to(agentsSockets).emit(event, data);
    }

    async emitToClients(event: string, data: unknown) {
      const sockets = await this.server.fetchSockets();

      const clientsSockets = sockets
        .filter((socket) => socket.data.client)
        .map((socket) => socket.id);

      this.server.to(clientsSockets).emit(event, data);
    }
```

Основні методи веб сокетів в яких обробляються підключення це handleConnection та handleDisconnect. В методі handleConnection ми авторизуємо користувача та приєднуємо його до потрібних чатів.

```
    async handleConnection(socket: Socket) {
      const agentToken = socket.handshake.auth.agentToken;
      const clientToken = socket.handshake.auth.clientToken;

      if (agentToken) {
        try {
          const { id } = this.jwtService.decode(agentToken) as
{ id: number };
```

```
            await this.onAgentJoin(socket, id);
          } catch (e) {
            console.log(`Cannot authorize agent, agent not
found.`, e);
          }
        }

        if (clientToken) {
          try {
            await this.onClientJoin(socket, clientToken);
          } catch (e) {
            console.log('client chat not found', e);
          }
        }
      }

      async handleDisconnect() {
        return;
      }

      @SubscribeMessage(ChatEvents.ClientJoinChat)
      async onClientJoined(socket: Socket, data: CreateChatDto) {
        const chat = await this.chatService.createChat(data);

        socket.data.client = chat.client;

        await socket.join(chat.id);

        await this.emitToAgents(ChatEvents.ChatOpen, chat);

        socket.emit(ChatEvents.ChatOpen, chat);
      }

      @SubscribeMessage(ChatEvents.ClientSendMessage)
      async onClientSendMessage(_: Socket, data) {
        const chat = await
this.chatService.findById(data.chatId);

        const message = await
this.chatService.sendMessageToChat({
          chatId: chat.id,
          text: data.text,
          from: 'CLIENT',
        });

        this.server.to(chat.id).emit(ChatEvents.ChatMessage,
message);
      }

      @SubscribeMessage(ChatEvents.ClientSeenMessage)
      async onClientSeenMessage(_: Socket, data) {
```

```
        const updatedChat = await
this.chatService.markMessagesAsSeen(
          data.chatId,
          'AGENT',
        );

        const chat = await
this.chatService.findById(data.chatId);

        this.server.to(chat.id).emit(ChatEvents.AgentChatUpdated,
updatedChat);
      }
```

Івенти це основний спосіб взаємодії в вебсокетах, їх відправляє як клієнт так і сервер. Це можуть бути як передача даних так і їх запит.

```
      @SubscribeMessage(ChatEvents.AgentJoinChat)
      async onAgentJoined(socket: Socket, data) {
        const chat = await
this.chatService.findById(data.chatId);

        await socket.join(chat.id);

        const updatedChat = await
this.chatService.joinAgentToChat(
          data.chatId,
          socket.data.agent.id,
        );


this.server.to(chat.id).emit(ChatEvents.ClientChatUpdated,
updatedChat);
        this.server.to(chat.id).emit(ChatEvents.AgentChatUpdated,
updatedChat);

        await this.emitToAgents(ChatEvents.AgentTakeChat,
updatedChat);
      }

      @SubscribeMessage(ChatEvents.AgentSendMessage)
      async onAgentSendMessage(_: Socket, data) {
        const chat = await
this.chatService.findById(data.chatId);

        const message = await
this.chatService.sendMessageToChat({
          chatId: chat.id,
          text: data.text,
          from: 'AGENT',
        });
```

```
        this.server.to(chat.id).emit(ChatEvents.ChatMessage,
message);
      }

      @SubscribeMessage(ChatEvents.AgentSeenMessage)
      async onAgentSeenMessage(_: Socket, data) {
        const chat = await
this.chatService.findById(data.chatId);

        if (chat.status === 'ACTIVE') {
          await this.chatService.markMessagesAsSeen(chat.id,
'CLIENT');

          const updatedChat = await this.prisma.chat.findUnique({
            where: {
              id: data.chatId,
            },
            include: {
              agent: true,
              client: true,
              messages: {
                orderBy: {
                  createdAt: 'asc',
                },
              },
            },
          });


this.server.to(chat.id).emit(ChatEvents.AgentChatUpdated,
updatedChat);
        }
      }
```

Даний метод веб сокетів, відправляється агентом для отримання нових та активних чатів.

```
      @SubscribeMessage(ChatEvents.AgentGetChats)
      async onAgentGetChats(socket: Socket) {
        const chats = await
this.chatService.findCurrentChats(socket.data.agentId);

        socket.emit(ChatEvents.AgentSetChats, chats);
      }

      @SubscribeMessage(ChatEvents.AgentStartTyping)
      async onAgentStartTyping(_: Socket, data) {

this.server.to(data.chatId).emit(ChatEvents.AgentStartTyping);
      }

      @SubscribeMessage(ChatEvents.AgentStopTyping)
```

```
        async onAgentStopTyping(_: Socket, data) {

this.server.to(data.chatId).emit(ChatEvents.AgentStopTyping);
        }

        @SubscribeMessage(ChatEvents.ClientStartTyping)
        async onClientStartTyping(_: Socket, data) {

this.server.to(data.chatId).emit(ChatEvents.ClientStartTyping);
        }

        @SubscribeMessage(ChatEvents.ClientStopTyping)
        async onClientStopTyping(_: Socket, data) {

this.server.to(data.chatId).emit(ChatEvents.ClientStopTyping);
        }

        @SubscribeMessage(ChatEvents.AgentCloseChat)
        async agentCloseChat(_: Socket, data) {
          const chat = await
this.chatService.closeChat(data.chatId);
          this.server.to(chat.id).emit(ChatEvents.ChatClosed,
chat);
        }

        @SubscribeMessage(ChatEvents.ClientSendFeedback)
        async clientSendFeedback(_: Socket, data:
CreateFeedbackDto) {
          await this.prisma.feedback.create({
            data: {
              rating: data.rating,
              chatId: data.chatId,
              clientId: data.clientId,
              agentId: data.agentId,
            },
          });


this.server.to(data.chatId).emit(ChatEvents.AgentGetFeedback,
data.rating);
        }

        @SubscribeMessage(ChatEvents.ClientGetOnlineAgents)
        async getOnlineAgents(socket: Socket) {
          const agents = await this.prisma.agent.findMany({
            where: {
              isOnline: true,
            },
          });

          socket.emit(ChatEvents.ClientSetOnlineAgents, agents);
        }
```

```
}
```

Основні методи даного класу:

- onClientJoin
- onAgentJoin
- emitToAgents
- emitToClients
- handleConnection

onClientJoin – метод який обробляє підключення від клієнтів.

onAgentJoin – метод який обробляє підключення від агентів.

emitToAgents – метод який відправляє повідомлення для агентів.

emitToClients – метод який відправляє повідомлення для агентів.

handleConnection – метод який обробляє підключення від Front-End частини додатку, та авторизує їх.

Приклад обробки повідомлення на сервері:

```
@SubscribeMessage(ChatEvents.ClientSendMessage)
  async onClientSendMessage(_: Socket, data) {
    const chat = await
this.chatService.findById(data.chatId);

    const message = await
this.chatService.sendMessageToChat({
      chatId: chat.id,
      text: data.text,
      from: 'CLIENT',
    });

    this.server.to(chat.id).emit(ChatEvents.ChatMessage,
message);
  }
```

Сервер очікує на повідомлення ChatEvents.ClientSendMessage, знаходить чат в базі даних, в який відправлено повідомлення, викликає метод sendMessageToChat в класу ChatService, та відправляє повідомлення клієнтам.

Також для чату було створено клас ChatService, який розташований в файлі chat.service.ts. Код класу ChatService:

```
@Injectable()
export class ChatService {
  constructor(private prisma: PrismaService) {}

  async findOne(input: Prisma.ChatFindUniqueArgs):
Promise<Chat> {
    return this.prisma.chat.findUnique(input);
  }

  async findMany(input: Prisma.ChatFindManyArgs) {
    return this.prisma.chat.findMany(input);
  }

  async findChatsByAgentId(agentId: number): Promise<Chat[]>
{
    return;
  }

  async findChatByClientId(clientId: string): Promise<Chat> {
    return;
  }

  async joinClientToChat(chatId: string, clientId: string) {
    return;
  }

  async joinAgentToChat(chatId: string, agentId: number) {
    const agent = await this.prisma.agent.findUnique({
      where: {
        id: agentId,
      },
    });

    return this.prisma.chat.update({
      where: {
        id: chatId,
      },
      data: {
        agent: {
          connect: {
            id: agent.id,
          },
        },
        status: 'ACTIVE',
        messages: {
          create: [
            {
              text: `Оператор ${agent.name} в чаті`,
```

```
              from: 'BOT',
            },
          ],
        },
      },
      include: {
        agent: true,
        client: true,
        messages: true,
      },
    });
  }

  async createChat(dto: CreateChatDto) {
    return this.prisma.chat.create({
      data: {
        project: {
          connect: {
            id: dto.projectId,
          },
        },
        client: {
          create: {
            name: dto.name,
            email: dto.email,
            phone: dto.phone,
            projectId: dto.projectId,
          },
        },
        messages: {
          create: [
            {
              text: 'Вітаємо в чаті!',
              from: 'BOT',
            },
          ],
        },
      },
      include: {
        client: true,
        messages: true,
      },
    });
  }
```

Даний метод відправляється агентом для завершення чатів після успішного обслуговування клієнта. Він змінює статус чата на Closed, що означає, що чат закритий і більше не буде відображатись.

```
  async closeChat(chatId: string) {
    return this.prisma.chat.update({
      where: {
```

```
        id: chatId,
      },
      data: {
        status: 'CLOSED',
        messages: {
          create: {
            text: 'Оператор завершив чат.',
          },
        },
      },
    });
  }

  async createMessage(chatId: string, message) {
    return;
  }

  async createFeedbackForChat(chatId, feedback) {
    return;
  }

  async markChatAsSeen(chatId: string) {
    return;
  }

  async findById(chatId: string) {
    return this.prisma.chat.findUnique({
      where: {
        id: chatId,
      },
    });
  }

  async findActiveChatsForAgent(agentId: number) {
    return this.prisma.chat.findMany({
      where: {
        agentId: agentId,
        status: 'ACTIVE',
      },
    });
  }

  async findCurrentChats(agentId: number) {
    const open = await this.prisma.chat.findMany({
      where: {
        status: 'OPEN',
      },
      include: {
        client: true,
        messages: true,
      },
      orderBy: {
```

```
        createdAt: 'desc',
      },
    });

    const active = await this.prisma.chat.findMany({
      where: {
        agentId: agentId,
        status: 'ACTIVE',
      },
      include: {
        client: true,
        messages: {
          orderBy: {
            createdAt: 'asc',
          },
        },
        agent: true,
      },
    });

    return [...active, ...open];
  }
```

Даний метод, викликається як клієнтом так і агентом, для того щоб змінити статус повідомлення на прочитане, і відобразити зміни в інтерфейсі користувача.

```
  async markMessagesAsSeen(chatId, from: SentBy) {
    return this.prisma.message.updateMany({
      where: {
        chatId: chatId,
        OR: [
          {
            from,
          },
          {
            from: 'BOT',
          },
        ],
      },
      data: {
        status: MessageStatus.SEEN,
      },
    });
  }

  async sendMessageToChat(dto: CreateMessageDto) {
    await this.prisma.chat.update({
      where: {
        id: dto.chatId,
      },
      data: {
        updatedAt: new Date(),
```

```
      },
    });

    return this.prisma.message.create({
      data: {
        chatId: dto.chatId,
        text: dto.text,
        from: dto.from,
      },
    });
  }

  async findLastMessageForChat(chatId: string) {
    return this.prisma.message.findFirst({
      where: {
        chatId: chatId,
      },
      orderBy: {
        createdAt: 'desc',
      },
    });
  }
}
```

В класі було реалізовано функціонал взаємодії з базою даних, для функціоналу чату за допомогою ORM Prisma.

Також було створено файл chat.module.ts, в якому реалізовано клас ChatModule, який обєднує в собі весь функціонал чату, для подальшого підключення його до додатку, так як він має модульну структуру.

Код класу ChatModule:

```
@Module({
  providers: [ChatGateway, ChatService],
  imports: [JwtModule.register({})],
})
export class ChatModule {}
```

3.3 Подальший розвиток продукту

Наразі це є готовим продуктом. Однак потрібно звернути увагу на те, що це лише MVP, і більше функцій буде додано пізніше. Нижче наведено короткий список планів на майбутнє:

- Розширення функціоналу
- Надання можливості повного редагування акаунтів для агентів та адміністраторів
- Додання можливості авторизації через облікові записи Google та Facebook
- Більш гнучкий показ статистки користувачу
- Розширення можливостей операторів та адміністраторів
- Можливість експорту даних клієнтів в окремий файл
- Окрім нового, вже існуючий функціонал теж планується бути покращеним та приведеним в порядок.

Висновок до розділу 3

В даному розділі було розібрано програмний продукт на компоненти та проаналізовано кожен із них. Також було детально розібрано функціонал готового програмного продукту, враховуючи перспективу кожного компонента. Було наведено фрагменти коду із яких складаються компоненти, щоб краще розуміти як працює продукт. Зрештою, розкривається подальша доля програмного продукту.

ВИСНОВКИ

У даному дипломному проекті було висвітлено процес проектування та розробки веб-застосунку для комунікації з клієнтами. Було розглянуто подібні або ж аналогічні за функціоналом веб-сервіси, здійснено аналіз їх позитивних та негативних сторін та зроблено висновки, які допомогли вияснити вимоги для функціоналу власного веб застосунку.

Було виконано покроковий та детальний аналіз програмних засобів, програмного забезпечення та інструментів для розробки, та здійснено обгрунтований вибір на користь тих чи інших інструментів розробки.

Створено бек-енд представлення для всього функціоналу програмного продукту. Здійснено детальний аналіз бази даних та функціоналу програмного продукту.

В результаті отримано продукт, який являється MVP, та ще не є до кінця готовим для виходу на ринок, але основний функціонал реалізовано та в майбутньому даний функціонал буде збільшуватись.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1 Бек-енд розробка [Електронний ресурс] – Режим доступу до ресурсу: https://www.upwork.com/resources/beginners-guide-back-end-development

2 Що таке веб-сервіси та API - [Електронний ресурс] – Режим доступу до ресурсу: https://nordicapis.com/what-is-the-difference-between-web-services-and-apis/

3 Smartsupp - [Електронний ресурс] – Режим доступу до ресурсу: https://www.smartsupp.com/live-chat/

4 Chaport - [Електронний ресурс] – Режим доступу до ресурсу https://www.chaport.com/

5 liveagent - [Електронний ресурс] – Режим доступу до ресурсу: https://www.liveagent.com/#a_aid=wpbeginner

6 [Електронний ресурс] – Режим доступу до ресурсу: https://project-seo.net/uk/blog-uk/krashchyj-onlajn-chat-dlya-sajtu/#idh2

7 Type Script - [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/TypeScript

8 [Електронний ресурс] – Режим доступу до ресурсу: https://www.linuxscrew.com/which-database-to-choose

9 Node.js frameworks - [Електронний ресурс] – Режим доступу до ресурсу: https://hackr.io/blog/nodejs-frameworks

10 Node.js - [Електронний ресурс] – Режим доступу до ресурсу: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

11 [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/w/index.php?curid=307218

12 Socket.io - [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Socket.IO

13 Docker - [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/overview/

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: Розробка backengine бізнес логіки та WEB- сервісу системи комунікації з клієнтами засобами JS "

Об’єм пояснювальної записки: 82 аркуші

Дата закінчення дипломної роботи 3 червня 2021р.

Підпис студента _____