КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-застосунок підтримки взаємодії перекладачів і клієнтів»

Виконав _____
(Підпис)
Большаков Денис Олександрович
(прізвище, ім'я, по батькові)

Керівник _Сайко Володимир Григорович
(прізвище, ім'я, по батькові)

Попередній захист:
до захисту

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали)
(Дата)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань
Унікальність тексту - 98 %

Київ – 2022

Київський національний університет імені Тараса Шевченка

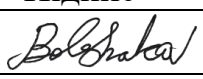
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Веб-застосунок підтримки взаємодії перекладачів і клієнтів»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Большаков Денис Олександрович	

Назва роботи українською та англійською мовами:

Веб-застосунок підтримки взаємодії перекладачів і клієнтів
Web application to support the interaction between translators and clients

Мета бакалаврської роботи: розробка веб-застосунку для підвищення ефективності взаємодії перекладачів та клієнтів.
План роботи: 1. Теоретичні основи побудови веб-застосунку підтримки взаємодії перекладачів і клієнтів 2. Аналіз та вибір підходів та засобів розробки веб-застосунків 3. Проектування та розробка веб-застосунку підтримки взаємодії перекладачів і клієнтів

ПІБ, ступінь, звання наукового керівника роботи:

Сайко Володимир Григорович, професор

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2022	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2022	Виконано
9.	Подання роботи у першому варіанті	23.04.2022	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	23.05.2022	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	30.05.2022	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	11.06.2022	Виконано
14.	Захист кваліфікаційної роботи бакалавра	22-24.06.2022	

Здобувач вищої освіти _____
(підпис)

Керівник _____ _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	2
Вступ	2
Розділ 1	5
Розділ 2	12
Розділ 3	24
Висновки	1
Перелік використаних джерел	3
Додатки	9

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розробн.</td><td>Большаков Д.О.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Сайко В.Г.</td><td></td><td></td><td>1</td><td>64</td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Кваліфікаційна робота: 64 с., 9 рис., 24 джерела, 2 дод.

Ця робота присвячена проектуванню та розробленню веб-застосунку підтримки взаємодії перекладачів і клієнтів.

Метою дипломної роботи є розробка веб-застосунку для підвищення ефективності взаємодії перекладачів та клієнтів.

Для досягнення поставленої мети вирішуються такі *завдання*:

- дослідження теоретичних основ побудови веб-застосунків,
- аналіз існуючих рішень,
- аналіз сучасних підходів та засобів розробки веб-застосунків,
- проектування веб-застосунку підтримки взаємодії перекладачів і клієнтів,
- розробка веб-застосунку;

Об'єктом дослідження є процес підтримки взаємодії перекладачів і клієнтів.

Предметом дослідження є програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку підтримки взаємодії перекладачів і клієнтів.

*Методи дослідження***:** аналітичний, що використовувався при огляді існуючих рішень застосунків підтримки взаємодії перекладачів і клієнтів; порівняльний, що використовувався при виборі підходів та засобів розробки веб-застосунку підтримки взаємодії перекладачів і клієнтів;

Практичне значення полягає у створенні веб-застосунку підтримки взаємодії перекладачів і клієнтів, який дозволяє клієнтам знаходити перекладачів для виконання замовлень та відслідковувати стан замовлення на всіх етапах виконання цього замовлення.

Ключові слова: веб-застосунок, переклад, JavaScript.

ABSTRACT

Qualification work: 64 pp., 9 fig., 24 sources, 2 appendix.

This work is dedicated to the design and development of a web application to support the interaction of translators and clients.

The aim of the thesis is to develop a web application to improve the interaction of translators and clients.

To achieve this goal the following tasks are solved:

- research of theoretical bases of construction of web applications,
- analysis of existing solutions,
- analysis of modern approaches and tools for developing web applications,
- designing a web application to support the interaction of translators and clients,
- web application development;

The object of research is the process of supporting the interaction of translators and clients.

The subject of the research is software and technical, organizational principles, principles, approaches to building a web application to support the interaction of translators and clients.

Research methods: analytical, used in the review of existing solutions of applications to support the interaction of translators and clients; comparative, which was used in choosing the approaches and means of developing a web application to support the interaction of translators and clients;

Of practical importance is the creation of a web application to support the interaction of translators and clients, which allows customers to find translators to fulfill orders and monitor the status of the order at all stages of the order.

ЗМІСТ

ВСТУП

Актуальність теми бакалаврської роботи полягає у постійному збільшенні важливості ролі надання послуг онлайн та зростанні темпів глобалізації, що сприяє стрімкішому розвитку зв’язків між різними народами та їх культурами, в тому числі й різномовними. Для підтримки таких зв’язків і потрібні послуги перекладачів.

Метою бакалаврської роботи є розробка веб-застосунку для підвищення ефективності взаємодії перекладачів та клієнтів.

Завданнями дослідження бакалаврської роботи є:

- дослідження теоретичних основ побудови веб-застосунків,
- аналіз існуючих рішень,
- аналіз сучасних підходів та засобів розробки веб-застосунків,
- проектування веб-застосунку підтримки взаємодії перекладачів і клієнтів,
- розробка веб-застосунку.

Об’єктом дослідження бакалаврської роботи є процес підтримки взаємодії перекладачів і клієнтів.

Предметом дослідження бакалаврської роботи є програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку підтримки взаємодії перекладачів і клієнтів.

Методи дослідження бакалаврської роботи:

- аналітичний – використовується при огляді існуючих рішень застосунків підтримки взаємодії перекладачів і клієнтів;
- порівняльний – використовується при виборі підходів та засобів розробки веб-застосунку підтримки взаємодії перекладачів і клієнтів;

Практичне значення – створення веб-застосунку підтримки взаємодії перекладачів і клієнтів, який дозволяє клієнтам знаходити перекладачів для виконання замовлень та відслідковувати стан замовлення на всіх етапах виконання цього замовлення.

Бакалаврська робота складається з трьох розділів, розділених на підрозділи, висновків, переліку використаних джерел та додатків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ВЕБ-ЗАСТОСУНКУ ПІДТРИМКИ ВЗАЄМОДІЇ ПЕРЕКЛАДАЧІВ І КЛІЄНТІВ

1.1 Поняття веб-застосунку

Веб-застосунок – це програма, яка використовує веб-браузер як посередника для виконання певної задачі. Веб-застосунок має клієнт-серверну архітектуру. Під клієнтом мається на увазі програма, яка використовується для доступу до веб-застосунка, тобто браузер. Під сервером мається на увазі сервер, на якому знаходиться веб-застосунок [1]. Таким чином, клієнт – це програма, яка використовується для візуального представлення даних, а сервер – це програма, яка і виконує функції веб-застосунку, тобто працює з самими даними, відповідаючи на запити, що надходять від клієнта. Принцип роботи веб-застосунку можна описати так[1]:

1. Користувач заходить на сайт.
2. Сервер клієнта надсилає запит на певну URL-адресу сервера веб-застосунку.
3. Сервер веб-застосунку отримує необхідну інформацію з бази даних.
4. Сервер веб-застосунку надсилає клієнту відповідь з необхідною інформацією.
5. Клієнт отримує інформацію у вигляді сторінки веб-застосунку.

1.2 Огляд існуючих рішень

На даний момент для пошуку перекладачів часто використовують сайти, які спеціалізуються на пошуці не конкретно перекладачів, а взагалі виконавців замовлень у багатьох різних сферах послуг, їх називають фрілансерами. Проте, існують і сервіси пошуку саме перекладачів. Розглянемо декілька з них.

1.2.1 TranslatorsCafe

URL-адреса: https://www.translatorscafe.com/cafe/

Цей застосунок має досить велику кількість функціоналу: пошук перекладачів, пошук перекладацьких агенцій, перегляд статей, участь у опитуваннях та спілкування на форумах на різні теми, проте багато з цього не відноситься до основного функціоналу. Він має дуже застарілий не завжди інтуїтивно зрозумілий дизайн з великою кількістю вкладених списків, через що користувачу часто доводиться витрачати багато часу на виконання звичайних дій. Позитивною якістю застосунку є безкоштовна реєстрація.

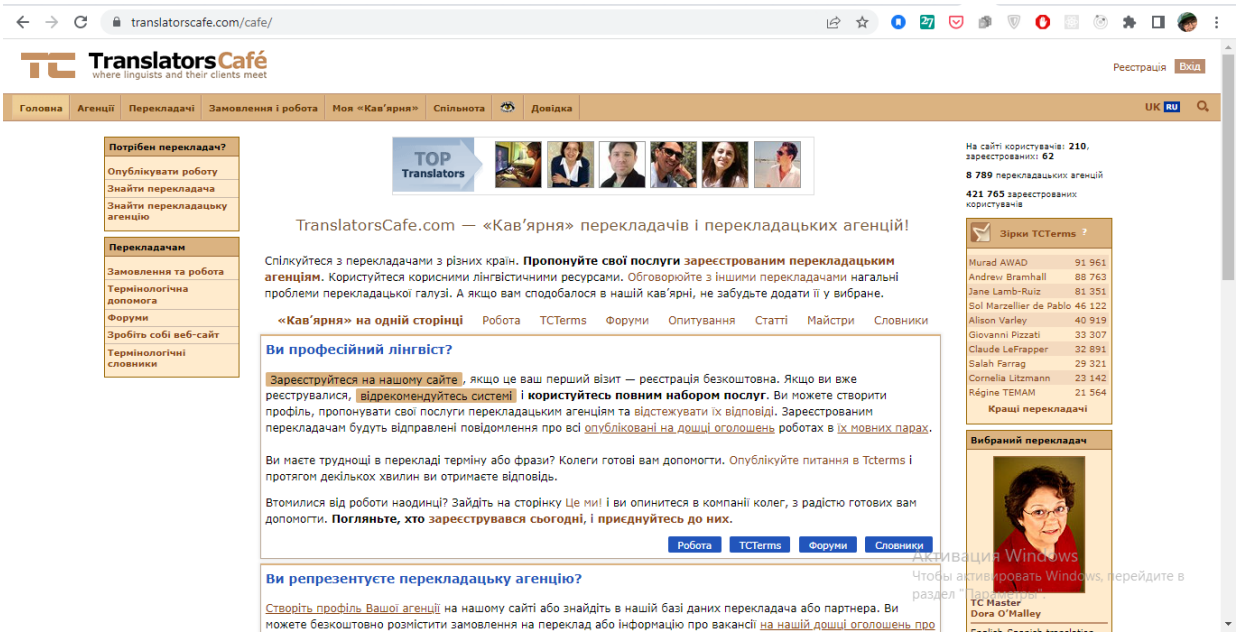


Рисунок 1.2.1.1 – Головна сторінка сайту TranslatorsCafe

1.2.2 Translator-Translation

URL-адреса: https://translator-translation.net/

Цей застосунок надає функціонал пошуку перекладачів та перекладацьких агенцій. Він теж має застарілий дизайн та велику вкладеність, що значно

сповільнює та ускладнює навігацію. Ще одним недоліком дизайну є наявність анімованих колонок зліва та справа від основної частини сторінки (рисунок 1.2.2.1). Вони часто змінюються та відволікають. Також застосунок має складну та довгу систему реєстрації, за якої користувач, який хоче зареєструватись як перекладач має скачати форму, заповнити її та відправити на електронну пошту. Також реєстрація у цьому застосунку не є безкоштовною.

Рисунок 1.2.2.1 – Головна сторінка сайту Translator-Translation

1.2.3 TranslationDirectory

URL-адреса: https://www.translationdirectory.com/

Цей застосунок є досить схожим на TranslatorsCafe, він має дуже схожий функціонал – пошук перекладачів та перекладацьких агенцій, перегляд статей, участь у опитуваннях та спілкування на форумах. Проте застосунок має схожі й недоліки. Головним з них є застарілий дизайн та відсутність структури у головному меню, що міститься у лівій частині сторінки (рисунок 1.2.3.1). Посилання на деякі функції містяться у основній частині сторінки, проте пошук інших розділів ускладнюється тим, що меню не розділене за типом дій, або

роллю користувача, тому щоб знайти необхідний розділ потрібно переглянути увесь список. Також робота з застосунком у ролі перекладача є платною.

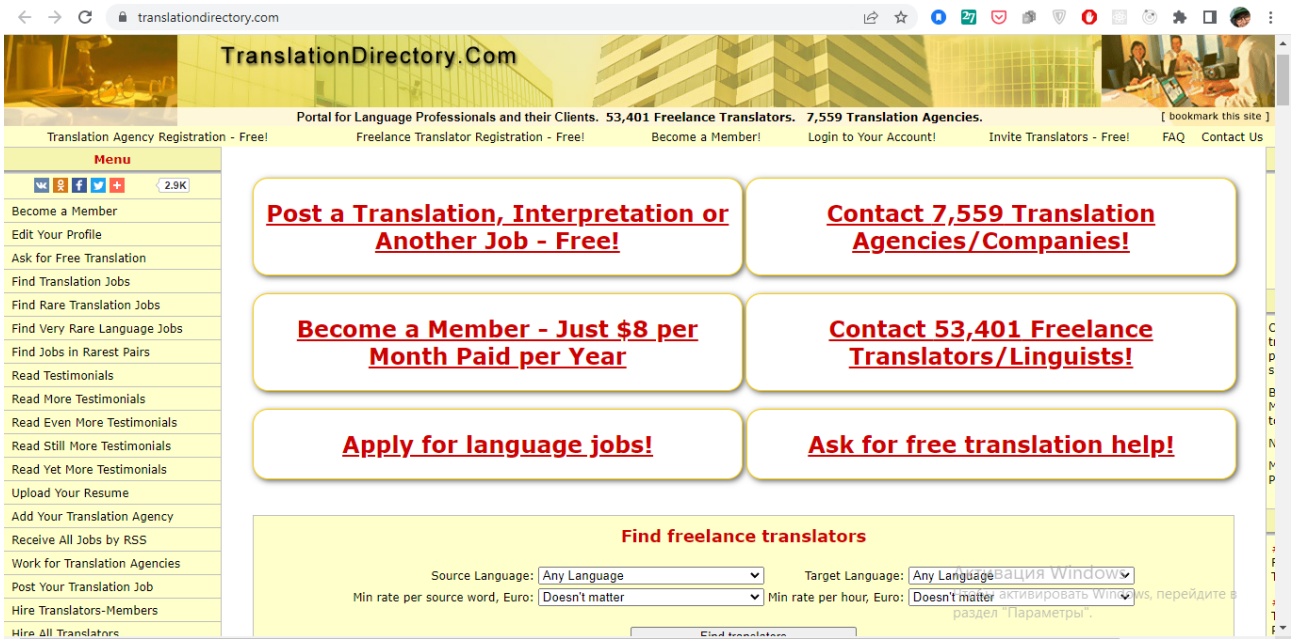


Рисунок 1.2.3.1 – Головна сторінка TranslationDirectory

Отже, розглянуті існуючі рішення веб-застосунків підтримки взаємодії перекладачів і клієнтів є досить схожими. Вони надають однаковий функціонал, проте також мають майже однакові недоліки, серед яких основним є застарілий та інтуїтивно незрозумілий інтерфейс з великим рівнем вкладеності, що погано впливає на комфортність користування застосунками.

1.3 Постановка задачі. Технічне завдання на розробку.

Згідно теми кваліфікаційної роботи бакалавра, потрібно реалізувати веб-застосунок підтримки взаємодії перекладачів і клієнтів. Веб-застосунок має складатись з:

- Сервера веб-застосунку.
- Клієнта веб-застосунку.
- Бази даних.

У розроблюваному веб-застосунку мають бути реалізовані:

- Функції реєстрації та входу у веб-застосунок.
- Функція пропонування користувачем замовлення.
- Функції підтвердження замовлення перекладачем та надсилання виконаного замовлення.

Клієнтська частина веб-застосунку повинна:

- давати змогу користувачу виконувати усі функції, що надає веб-застосунок,
- мати зручний та зрозумілий користувацький інтерфейс.

Отже, після проведення аналізу та виявлення недоліків існуючих рішень веб-застосунків підтримки взаємодії перекладачів і клієнтів було сформульовано основні вимоги до функціоналу розроблюваного веб-застосунку.

РОЗДІЛ 2. АНАЛІЗ ТА ВИБІР ПІДХОДІВ ТА ЗАСОБІВ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

2.1 Серверна частина веб-застосунку

Наявність сервера є необхідною для функціонування веб-застосунка, адже сервер являє собою API, тобто, певний програмний інтерфейс, що забезпечує отримання і обробку запитів на отримання необхідної інформації з бази даних, що надсилаються самим веб-застосунком зі сторони клієнта і, таким чином, реалізує зв’язок між джерелом даних, яким являється база даних, з представленням цих даних користувачу за допомогою клієнтської частини веб-застосунку.

API – це набір функцій, кожна з яких отримує на вхід свій набір даних, що має відповідати певним правилам, і в результаті виконання надає вихідні дані, кожна така функція має свою URL-адресу, на яку надсилається запит, що містить вхідні дані, далі функція обробляє ці дані, наприклад, формує та надсилає запит до бази даних, та за результатом виконання повертає вихідні дані.

На даний момент найпопулярнішими підходами до проектування API є:

- RPC-XML
- SOAP
- REST

XML-RPC – це протокол виклику віддалених процедур, що використовує мову XML для написання повідомлень та HTTP для їх передачі. Цей протокол визначає стандартний набір команд, який можна використовувати для до доступу до функціональності іншого віддаленого сервісу[2]. В порівнянні з SOAP, є більш простим у використанні, але вважається застарілим.

SOAP – це протокол обміну структурованою або типізованою інформацією між сервісами у розподіленому середовищі. SOAP визначає модель пакетування

та кодування даних, запити використовують мову XML[3]. Цей протокол може використовуватись у великій кількості систем, але на даний час вважається застарілим та використовується переважно у банківській та телекомунікаційних сферах через велику кількість стандартів, яким мають відповідати повідомлення.

REST – це стиль архітектури мережевих протоколів, головна ідея якого полягає у тому, що з кожним зверненням до API змінюється стан веб-застосунку на стороні клієнта. Цей стиль архітектури має шість основних принципів:

1. Єдиний інтерфейс, що має свої обмеження: кожен ресурс обробки запитів має мати свій унікальний ідентифікатор, усі ресурси мають мати однаковий формат повідомлень, що надсилаються у відповідь на запити, кожен запит має містити достатньо інформації для його обробки, а також містити дані про те, які додаткові дії клієнт може виконувати на цьому ресурсі, клієнт повинен мати тільки початкову URL-адресу API, а усі подальші дії мають виконуватись за допомогою гіперпосилань.
2. Клієнт-серверна архітектура застосунку, тобто розділення вузлів обробки даних, та представлення їх користувачу, що дозволяє легше додавати нові платформи, на яких може використовуватись клієнтський застосунок.
3. Сервер не має зберігати даних про стан клієнтського додатку, тому, відповідно до першого принципу, кожен запит має містити достатньо даних для його обробки.
4. Відповіді сервера можуть кешуватись клієнтом та використовуватись пізніше, але для цього відповідь повинна містити дані про те, чи можна її кешувати та як довго.
5. Має бути поділ на шари абстракції, де кожен компонент може взаємодіяти тільки з компонентами зі свого ж шару, або сусідніх.

6. Код на вимогу, що означає можливість завантаження клієнтом додаткового виконуваного коду з сервера, що дозволяє спрощувати клієнтську частину застосунку зменшуючи кількість необов'язкового коду.

REST для передачі повідомлень використовує протокол HTTP, тому підтримує усі методи HTTP, тобто GET, PUT, POST, DELETE, але чітких умов коли має використовуватись той чи той протокол немає[4].
API, що використовують архітектуру REST, називаються RESTful API.

Отже, для побудови веб-сервісу підтримки взаємодії прекладачів і клієнтів найкращим є саме підхід REST бо на даний момент він є найпоширенішим, що зумовлює достатню кількість інструментів для розробки застосунку та має меншу кількість умов яким чітко мають відповідати повідомлення для роботи застосунку без помилок.

Наступним кроком потрібно обрати мову, на якій буде розроблятись API. У сучасному світі існує велика кількість мов та написаних на них фреймворків для створення бекенд частини застосунку. Фреймворк – це певний набір вже реалізованих базових модулів, необхідних для виконання базових функцій, для яких призначений сам фреймворк. Використання фреймворків полегшує й пришвидшує процес розробки. Для реалізації REST API найпопулярнішими є фреймворки: Django та Flask для мови програмування Python, Spring Framework + Spring Boot для Java, Express для JavaScript, ASP.Net Core для C#, Ruby on Rails для Ruby, CakePHP та Laravel для PHP. З цього переліку я мав найбільше досвіду роботи саме з мовою JavaScript, тому обрав саме Express. Також використання API, реалізованого за допомогою Express полегшить розробку, адже, таким чином, і серверна, і клієнтська частина будуть реалізовані на одній мові програмування, JavaScript.

Express – це гнучкий фреймворк для розробки API веб-застосунків. Основою фреймворку Express є платформа Node.js[5]. Мова JavaScript

першочергово була створена для роботи у середовищі веб-браузерів, але використання Node.js дозволяє запускати JavaScript-код у середовищі сервера.

2.2 База даних

База даних – це організована колекція структурованих даних. Дані у ній зберігаються у вигляді таблиць з колонками та рядками [6]. Існує певна кількість характеристик, за якими класифікують бази даних, але найсуттєвішою з них є розділення за моделлю зберігання даних, тому виділяють такі типи баз даних: ієрархічні, мережеві, реляційні та NoSQL бази даних [7].

Структура ієрархічної бази даних має вигляд дерева. Запис у ній містить поміж власне даних і інформацію про батьківський запис та записи-нащадки, Проте, записи, що знаходяться на першому і останньому рівнях дерева не мають батьків і нащадків відповідно[8].

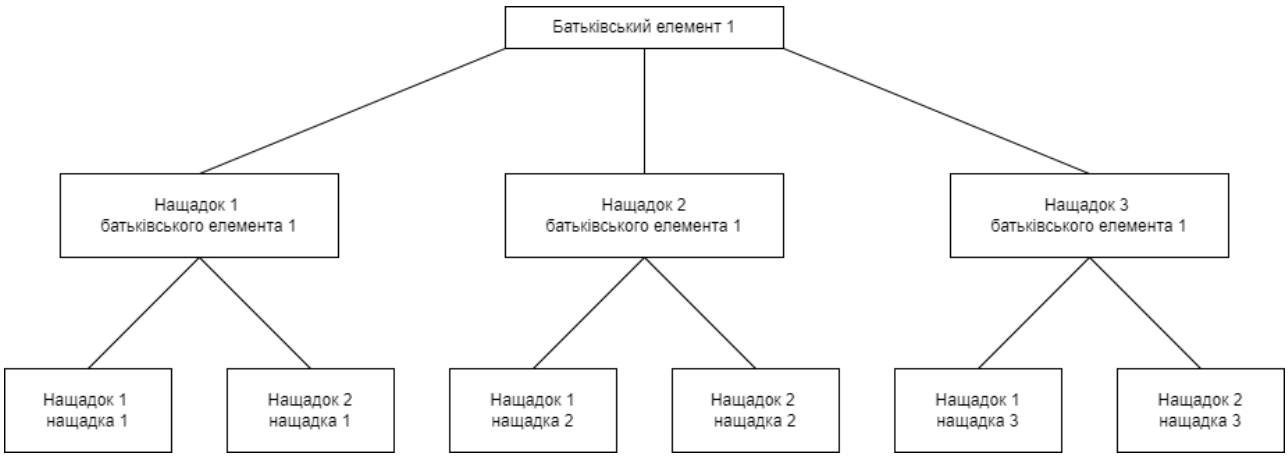


Рисунок 2.2.1 – Структура ієрархічної бази даних

Структура мережевої бази даних схожа на структуру ієрархічної бази даних, але відрізняється від неї тим, що для записів-нащадків у ній немає обмеження на кількість батьківських записів. Тому такі бази даних фактично мають графову модель зберігання даних.

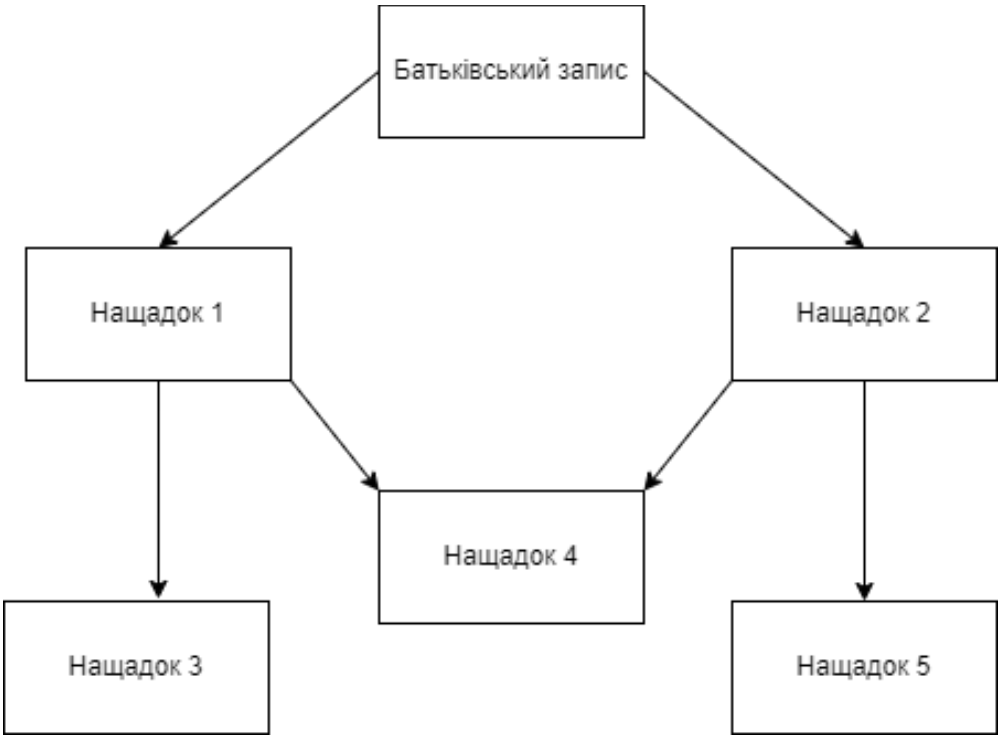


Рисунок 2.2.2 – Структура мережевої бази даних

У реляційних базах даних дані зберігаються у вигляді таблиць. Ці таблиці можуть мати з’язок між собою на основі спільних даних, які містяться у обох з’єднаних таблицях [9]. Наприклад, база даних з замовленнями (рисунок 2.2.3). У ній містяться таблиці з даними про замовників (таблиця Customers), замовлення (таблиця Orders) та відправлення (таблиця Shipments). Таблиця замовників має поля ідентифікатору замовника та його ім’я. Поле ідентифікатору є первинним ключем, бо ідентифікатор є унікальним, тому навіть за умови того, що будуть існувати два різних замовники з однаковими імена, адже імена не є унікальними, їх все одно можна буде розрізнити за унікальним ідентифікатором. Таблиця замовлень складається з полів ідентифікатору замовлення, ідентифікатору замовника та поля дати замовлення. Первинним ключем у ній є поле ідентифікатору замовлення. Але вона має і зовнішній ключ, яким є поле ідентифікатору замовника. Отже, наявність даних ідентифікатора замовника у обох таблицях дає можливість встановити між ними зв’язок. Для цього таблиці у реляційних базах даних і мають мати первинний та зовнішній ключі.

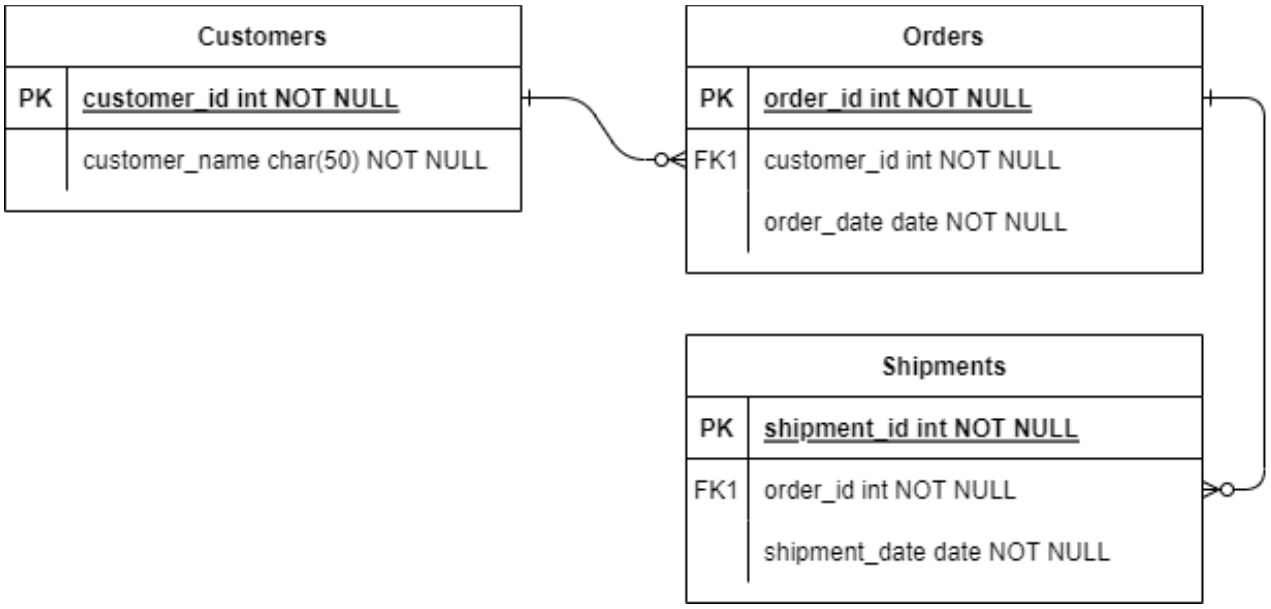


Рисунок 2.2.3 – Структура релятивної бази даних замовлень

Для роботи з реляційними базами даних використовується мова запитів SQL. Вона дозволяє об’єднувати таблиці використовуючи всього кілька рядків коду[9].

У NoSQL базах даних не завжди використовується мова SQL для запитів. Бази даних такого типу є нереляційними і використовують для збереження даних документи а не таблиці. У свою чергу, такі бази даних поділяються на багато видів, найпопулярнішими з них є документові бази даних, бази даних виду “ключ-значення”, ширококолонкові бази даних та графові бази даних. У документних базах даних інформація зберігається у вигляді документів, зазвичай, у форматі JSON. У базах даних виду “ключ-значення” дані зберігаються у колекціях записів, що складаються зі значень, кожному з яких відповідає унікальний ключ. Ширококолонкові бази даних використовують таблиці, але кожен запис може містити свою кількість колонок. Графові бази даних використовують графову модель даних для збереження зв’язків між записами[10].

Структура ієрархічних та мережевих баз даних не підходять для використання у веб-застосунку підтримки взаємодії перекладачів і клієнтів, оскільки дані у застосунку будуть мати структуру не ієрархічних вузлів, а

структуру колекцій даних, які будуть мати свої моделі, тому варто розглянути реляційні та No-SQL бази даних.

Серед переваг реляційних баз даних можна такі[11]:

- Добре підходять для чітко структурованої інформації.
- Використовують SQL.
- Добре підходять для виконання комплексних запитів.
- Мають високу надійність.
- Мають легку навігацію по даних.
- Високий рівень інтеграції даних через наявність зв’язків між таблицями.

Недоліками реляційних баз даних є[11]:

- Сильний зв’язок між даними у таблицях, які пов’язані відношеннями призводить до того, що при зміні або видаленні даних, що є первинним ключем таблиці, мають змінюватись або видалятись пов’язані дані у інших таблицях.
- Продуктивність роботи може бути низькою.

Перевагами No-SQL баз даних є[11]:

- Модель даних є більш гнучкою.
- Зміна даних одного запису не вимагає зміни у інших записах.
- Більша продуктивність виконання запитів.

Недоліками No-SQL баз даних є[11]:

- Нижча надійність.
- Важче перевірити зв’язок між даними.

Так, як веб-застосунок підтримки взаємодії перекладачів і клієнтів може містить досить великий об’єм даних і повинен мати велику продуктивність обробки запитів для швидкої роботи у режимі реального часу, кращим варіантом для використання буде No-SQL база даних. Також важливою перевагою використання бази даних цього виду є те, що, на відміну від більшості

реляційних баз даних, серед No-SQL баз даних є велика кількість хмарних баз даних, що зумовлює відсутність необхідності підтримувати роботи ще одного серверу, на якому і буде розміщена база даних.

Отже, для використання у проекті веб-застосунку підтримки взаємодії перекладачів і клієнтів я обрав базу даних MongoDB. MongoDB – це документо-орієнтована база даних. У ній колекція для зберігання даних необов'язково має мати схему і дозволяє зберігати документи з різною кількістю і типом полей[12]. Для роботи з даними існує модуль mongoose для Node.js. Він дозволяє легко створювати схеми і моделі колекцій, а також виконувати запити до бази даних кількома рядками коду використовуючи функції, які містить модуль.

2.3 Клієнтська частина веб-застосунку

Клієнтською частиною веб-застосунку є веб-сайт. Існує два види веб-сайтів – статичні та динамічні. Статичний веб-сайт являє собою певну кількість звичайних веб-сторінок, написаних на мові гіпертекстової розмітки HTML та використовують мову стилів CSS. Цей вид зазвичай використовується для простого подання інформації будь-якому користувачу. Динамічний веб-сайт є веб-застосунком, який в процесі виконання генерує HTML код, який і використовується браузером для подання інформації користувачу.

Так, як темою бакалаврської роботи є саме веб-застосунок, клієнтською частиною буде саме динамічний веб-сайт, тобто, власне, веб-застосунок. Принцип роботи веб-застосунку полягає у тому, що в результаті дій користувача клієнтська частина веб-застосунку формує запит для сервера на отримання або надсилання необхідної інформації, далі сервер виконує запит, повертає дані користувачу і, отримавши дані, веб-застосунок генерує сторінку, або її частину для представлення отриманої інформації користувачу[13]. Веб-застосунки можуть бути односторінковими (Single Page Application або скорочено SPA), або

багатосторінковими (Multiple Page Application або MPA). Вони відрізняються тільки тим, що SPA має один шаблон веб-сторінки, у який будуть вбудовуватись необхідні в той чи той момент елементи, а MPA – кілька таких шаблонів. Найчастіше веб-застосунки розробляються на мові JavaScript та з використанням фреймворків, то модулів з вже реалізованими деякими функціями, які повинен мати веб-застосунок для його роботи. Найпопулярнішими фреймворками для розробки веб-застосунків на мові JavaScript є AngularJS, React та Vue.js.

AngularJS – це фреймворк для розробки динамічних веб-застосунків, який дозволяє розширити синтаксис HTML щоб використовувати компоненти веб-застосунку у більш чіткому вигляді[14]. Ось деякі з головних особливостей AngularJS є[15]:

- Прив’язка даних – синхронізація даних між компонентами моделі та представлення.
- Об’єкти Scope – вони посилаються на модель і пов’язують представлення та його контролер.
- Деяка кількість вбудованих сервісів які вимагають створення свого об’єкту лише один раз у всьому застосунку.
- Директиви – маркери DOM-елементів, які дозволяють створювати власні HTML теги.
- Шаблони – відрендерене подання, що містить дані від контролера та моделі.
- Роутінг – принцип перемикання подань.
- Model-View-Whatever – дизайн-паттерн для розділення застосунку на компоненти моделі, контролера та подання. Проте, AngularJS не використовує дизайн паттерн Model-View-Controller у звичайному вигляді, а використовує більш подібний до Model-View-ViewModel.
- Глибокі посилання – AngularJS дозволяє записувати стан застосунку у URL-адресі.

- Використання Dependency Injection – це шаблон проектування, який дозволяє створювати об’єкти, які використовують інші об’єкти[16]

React – це бібліотека для створення користувацьких інтерфейсів. Її головна ідея полягає у створенні інтерфейсів, які складаються з перевикористовуваних компонентів, які відображають зміну даних у часі. React не є повноцінним фреймворком, адже він не використовує дизайн-паттерн MVC, а є лише його частиною, а саме – поданням (View)[17]. Проте, він надає можливість роботи з даними іншим шляхом, за допомогою сховища Redux, де зберігаються дані про стан компонентів застосунку і, таким чином, при зміні стану компонента відбувається рендер цього компоненту вже з новими даними[18].

Серед головних особливостей цієї бібліотеки можна виділити такі[17]:

- Використання JSX – розширення мови JavaScript, яке дозволяє створювати HTML шаблони прямо у JavaScript коді компонента[18].
- Однонаправлена прив’язка даних, що дозволяє спростити архітектуру застосунку.
- У React усе є компонентами.

Vue,js – це гнучкий фреймворк для розробки веб-інтерфейсів та односторінкових веб-застосунків. Він використовує дизайн-паттерн MVC (Model-View-Controller) і фокусується саме на представленні[19].

Його головними особливостями можна назвати[19]:

- Використання двосторонньої прив’язки даних.
- У ньому, як і у React, усе є компонентами, які можуть перевикористовуватись багато разів.
- Також дозволяє використання JSX, розширення мови JavaScript.

AngularJS, React та Vue.js є досить схожими, проте мають свої відмінності. У архітектурі застосунку React та Vue.js мають менше вимог, ніж AngularJS, також їх перевагою є можливість використання JSX синтаксису для створення HTML шаблонів, що значно полегшує розробку веб-застосунку. Ще однією відмінністю є швидкість роботи, де React та Vue.js переважають над AngularJS. Серед усіх трьох фреймворків React є найбільш легким для вивчення[20].

Отже, з порівняння видно, що React та Vue.js мають більше переваг для створення невеликих веб-застосунків, проте, зважаючи на легшість вивчення React, я обрав саме його для розробки веб-застосунку підтримки взаємодії перекладачів і клієнтів.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ПІДТРИМКИ ВЗАЄМОДІЇ ПЕРЕКЛАДАЧІВ І КЛІЄНТІВ

3.1 Проектування веб-застосунку

По-перше, у веб-застосунку мають бути реалізовані функції реєстрації та входу у акаунт. Процес реєстрації нового користувача буде виглядати так: користувач переходить на сторінку реєстрації у клієнтській частині веб-застосунку, вводить необхідні дані у формі реєстрації та натискає на кнопку

відправки, після цього клієнт надсилає запит на сервер з цими даними, сервер отримує запит та зберігає дані у базі даних.

Процес входу у акаунт буде виглядати так: користувач переходить на сторінку входу у клієнтській частині веб-застосунку, вводить дані для входу та натискає на кнопку входу, далі клієнт надсилає запит на сервер з цими даними, сервер отримує запит, перевіряє наявність цих даних у базі даних та, у разі, якщо все вірно, генерує JWT-токен, що містить ідентифікатор користувача у базі даних, та надсилає клієнту відповідь з даними цього користувача та його токеном, потім дані користувача зберігаються у сховищі клієнта, а токен у локальному сховищі браузера. Також при успішному вході у акаунт генерує JWT-токен користувача та надсилає його у відповіді клієнту разом з даними користувача. JWT, розшифровується як JSON Web Token – це відкритий стандарт, який визначає компактний та самодостатній шлях захищеного обміну даними у вигляді об’єкту формату JSON. Такий токен складається з трьох частин – заголовку з даними про тип та алгоритм шифрування, основної частини, що містить дані та підпис. Дані, що передаються у токені, шифруються одним з алгоритмів шифрування з використанням довільного секретного ключа[21].

Проте, дані користувача інколи можуть видалятись зі сховища клієнта, тому необхідно реалізувати ще функцію аутентифікації. Принцип її роботи буде таким: користувач переходить на нову сторінку у клієнтській частині веб-застосунку, клієнт надсилає на сервер запит, що містить токен цього користувача, що зберігається у сховищі, сервер перевіряє справжність цього токена та, якщо все вірно, надсилає клієнту відповідь з даними користувача та новим токеном, далі ці дані зберігаються у клієнті.

Принцип роботи з замовленнями виглядатиме так: користувач, який є клієнтом, обирає перекладача та пропонує йому роботу, перекладач погоджується та надсилає підтвердження, після цього він приступає до виконання, після закінчення виконання замовлення перекладач надсилає

замовнику відповідь, замовник перевіряє правильність виконання та підтверджує це. Це можна зобразити у діаграмі прецедентів (рисунок 3.1.1).

Рисунок 3.1.1 – Діаграма прецедентів веб-застосунку

Тому мають бути реалізовані такі функції роботи з замовленнями: пропонування роботи перекладачу, підтвердження замовлення перекладачем, надсилання виконаного замовлення перекладачем, підтвердження правильності виконання замовником.

Структура застосунку сервера веб-застосунку повинна мати такі складники: модулі, що відповідають за працездатність самого застосунку сервера, обробник запитів, пов’язаних з користувачами, що реалізує описані вище функції авторизації, обробник запитів, пов’язаних з замовленнями, що реалізує описані вище функції роботи з замовленнями, обробник запитів, пов’язаних з отриманням даних перекладачів та моделі сутностей, що будуть зберігатись у базі даних.

Для реалізації клієнтської частини буде використовуватись фреймворк React, а дані, отримані від сервера будуть зберігатись у сховищі Redux. Принцип роботи цього сховища такий: виконується якась функція, яка повертає дані, що

треба зберегти, потім ці дані передаються у відповідний генератор екшенів, він генерує екш типу відповідного цим даним, після цього викликається редюсер – функція, яка отримує на вхід екшн та повертає новий стан додатку, потім викликається диспатч, який передає отриманий новий стан у сховище. Тому клієнтська частина веб-застосунку підтримки взаємодії повинна складатись з файлів самого клієнта, файлів з реалізацією редюсерів для даних кожної з сутностей, що будуть зберігатись у сховищі – сутності користувача, перекладачів та замовлень, файлів з реалізацією усіх функцій надсилання запитів та отримання даних від сервера та компонентів усіх сторінок клієнтської частини веб-застосунку – головної сторінки, сторінок входу та реєстрації, сторінки пропонування роботи та сторінки роботи з замовленнями.

3.2 Розробка сервера

Як було описано у пункті 2.1, для розробки серверної частини я обрав фреймворк Express для мови JavaScript. Спочатку потрібно проініціалізувати JavaScript застосунок, для цього у папці, де він буде міститись потрібно виконати команду npm init. Після цього я встановив перші необхідні для розробки модулі – nodemon, express та config. nodemon дозволяє запустити локальний веб-сервер, який буде автоматично перезапускатись при зміні файлів застосунку. express – це модуль самого фреймворку Express. config дозволяє створити JSON файл, який буде містити значення певних налаштувань з ключами, які їм відповідають, та використовувати ці значення у будь-якій точці застосунку. Створив цей файл та додав у нього ключ зі значенням порту, на який будуть прийматись запити з клієнтської частини веб-застосунку. Створив файл ініціалізації застосунку. Імпортував у нього модулі express та config. Створив екземпляр об’єкту додатка фреймворку та змінну, яка отримує з файла налаштувань номер порта. Створив функцію запуску локального сервера, у якій викликається метод

прослуховування заданого порту, тобто дає можливість обробляти запити, що надходять на цей порт. Цей метод надає створений екземпляр додатку Express.

Наступним кроком потрібно створити та підключити до застосунку базу даних MongoDB. Для цього я зареєструвався на сайті, створив кластер та створив у ньому базу даних. При створенні бази даних, MongoDB автоматично надає приклад рядку підключення до неї. Додав цей рядок у файл налаштувань. Встановив модуль mongoose, за використовуючи методи якого, можна взаємодіяти з базою даних. Оновив файл ініціалізації, імпортувавши у нього модуль для роботи з базою даних та додавши у функцію запуску сервера виклик методу підключення до бази даних, що надає модуль mongoose, як аргумент у нього передається отриманий з файлу налаштувань рядок підключення до бази даних, яка буде використовуватись.

Далі потрібно розробити схеми та моделі для кожної з сутностей, що будуть зберігатись у базі даних. Всього таких сутностей буде три: сутність користувача, сутність перекладача та сутність замовлення. Сутність користувача повинна мати наступні поля:

1. адреса електронної пошти, яка має бути унікальною,
2. пароль,
3. об’єкт імені, у якому будуть міститись поля імені та прізвища,
4. роль,
5. масив посилань на записи замовлень, у яких користувач є замовником.

Сутність перекладача повинна мати наступні поля:

1. посилання на запис користувача, якому відповідає цей перекладач,
2. масив мов, з якими цей перекладач може працювати,
3. рейтинг,
4. масив посилань на записи замовлень, у яких, цей перекладач є виконавцем.

Сутність замовлення повинна мати наступні поля:

1. посилання на запис замовника,
2. посилання на запис перекладача, який є виконавцем,
3. статус замовлення,
4. дата створення замовлення,
5. об’єкт з інформацією про замовлення, у ньому містяться:
    5.1. об’єкт з даними про мову, з якої потрібно перекласти замовлення, та мову, на яку потрібно перекласти замовлення,
    5.2. короткий опис замовлення,
    5.3. посилання на файл, текст якого потрібно перекласти,
    5.4. дата, до котрої потрібно виконати замовлення,
    5.5. ціна.
6. об’єкт з відповіддю на замовлення, у якому містяться:
    6.1. посилання на файл з виконаним замовленням,
    6.2. дата надсилання виконаного замовлення.

Наступним кроком я створив схеми та моделі для кожної з цих сутностей. Для цього у окремій директорії створив три JavaScript файли, у кожному з яких імпортував з модуля mongoose конструктори схеми, моделі та тип ObjectId. Схема є певним шаблоном того, як буде виглядати документ у базі даних[22]. Модель відповідає за створення та читання документів цієї схеми у базі даних[23]. ObjectId є типом у схемі, який створює об’єкт з рядком ідентифікатора, за є необхідним для створення у схемі посилання на ідентифікатори записів, які містяться у інших колекціях бази даних. Після реалізації схем сутностей експортував моделі, створені за допомогою схем щоб використовувати їх у сервісах, що будуть обробляти запити, пов’язані з цими моделями.

Далі створив файли з роутерами, тобто обробниками маршрутів – з роутером авторизації, роутером перекладачів та роутером замовлень. Для

роутера авторизації створив три маршрути, на які можуть надходити запити – маршрути реєстрації, входу та аутентифікації.

Запит на реєстрацію має містити такі дані: адресу електронної пошти, пароль, об’єкт з іменем та прізвищем користувача, роллю та масивом мов у разі, якщо роль користувача – перекладач. Першим чином відбувається перевірка правильності адреси електронної пошти, та довжина паролю, який має бути довше ніж три символи, але коротше десяти. Для цього використовується модуль express-validator, що надає проміжні обробники для перевірки даних на відповідність заданим параметрам. Далі, у разі правильного проходження перевірки відбувається пошук запису користувача з даною адресою електронної пошти. Якщо такий запис не знайдено, відбувається шифрування паролю за допомогою модуля bcryptjs та збереження запису нового користувача у колекції користувачів бази даних. У разі, якщо роллю даного користувача є перекладач, відбувається збереження у колекції перекладачів бази даних запису, що містить дані про те, з якими мовами може працювати цей перекладач та посилання на ідентифікатор запису даного користувача, який MongoDB присвоює автоматично при створенні документу.

Наступним маршрутом є маршрут входу. Запит на вхід має містити дані адреси електронної пошти та паролю. Після отримання запиту відбувається пошук запису користувача, зареєстрованого з даною адресою електронної пошти. У разі, якщо такий запис не знайдено, сервер відправляє відповідь з повідомленням про помилку. Якщо запис знайдено, відбувається перевірка правильності паролю засобами модулю bcryptjs для порівняння зашифрованих та не зашифрованих рядків. Якщо пароль не співпадає з паролем у записі, клієнту відправляється повідомлення з помилкою. Якщо паролі співпадають, за допомогою модуля jsonwebtoken генерується JWT-токен даного користувача, у якому кодується ідентифікатор даного користувача, та відправляється клієнту. Також клієнту відправляються і дані про самого користувача.

Останнім маршрутом роутера авторизації є маршрут аутентифікації. Він потрібен для того, щоб обробляти запити, які клієнт буде надсилати автоматично кожен раз при завантаженні нової сторінки додатку і, таким чином, буде перевірятись чи увійшов користувач у систему. Для цього запит аутентифікації має містити лише заголовок авторизації з JWT-токеном, який надсилається клієнту у відповідь на успішний вхід та зберігається у сховищі браузера. Для обробки таких запитів потрібно створити проміжний обробник, який буде перевіряти наявність у запиті заголовка авторизації та обробляти отриманий з нього JWT-токен користувача. Далі цей обробник декодує отриманий JWT-токен, отримуючи ідентифікатор користувача. Після цього цей обробник відправляє цей запит далі, обробнику запитів аутентифікації роутера авторизації. У ньому відбувається пошук у базі даних запису користувача з отриманим ідентифікатором. Далі, у разі, якщо запис знайдено, знову генерується JWT-токен, у якому кодується ідентифікатор користувача та клієнту відправляється відповідь з цим згенерованим токеном та даними користувача.

Для роутера перекладачів створив три маршрути: отримання списку всіх перекладачів, отримання списку перекладачів за конкретною мовою та отримання даних конкретного перекладача за його ідентифікатором.

Запит на отримання списку усіх перекладачів не повинен передавати серверу дані. Після отримання такого запиту сервер звертається до бази даних та отримує усі записи колекції перекладачів та дані про користувачів, на ідентифікатори записів яких посилається кожен запис про перекладача. Таким чином, сервер отримує список усіх перекладачів з повною інформацією про них. Після успішного отримання списку, він відправляється у відповідь клієнту.

Запит на отримання списку перекладачів за конкретною мовою має містити назву мови, список перекладачів якої має відправитись у разі успішного виконання. Після отримання запиту відбувається пошук у колекції перекладачів

за полем масива мов, який має містити дану мови. Після цього сервер надсилає відповідь з отриманим з бази даних списком перекладачів.

Запит на отримання даних конкретного перекладача має містити його ідентифікатор. Після отримання такого запиту сервер виконує пошук одного запису з таким ідентифікатором у колекції перекладачів бази даних, а також отримує дані з запису у колекції користувачів, посилання на який міститься у записі перекладача і, таким чином, отримуються повні дані цього перекладача. Потім вони відправляються сервером клієнту у відповідь на успішне виконання запиту.

Останнім роутером є роутер замовлень, він може опрацьовувати шість типів запитів: запити на пропонування роботи, підтвердження замовлення, відправку виконаного замовлення, підтвердження правильності виконання замовлення та запити на отримання списку усіх замовлень користувача, на отримання списку усіх замовлень, у яких бере участь даний перекладач.

Запит на пропонування роботи відправляється клієнтом коли замовник обирає перекладача, якому хоче запропонувати роботу та заповнює форму з даними про замовлення. Такий запит має містити ідентифікатори акаунтів замовника та перекладача, а також об’єкт з даними про саме замовлення. Після отримання запиту на пропонування роботи створюється новий екземпляр моделі замовлення, у який поміщаються отримані з запиту дані про замовлення. У полі статусу замовлення вказується те, що воно запропоноване перекладачу, а у поле дати створення даного замовлення записуються поточна дата та час. Потім відбувається пошук у базі даних запису користувача з даним ідентифікатором замовника та, при знаходженні запису, у полі замовлень цього користувача створюється посилання на ідентифікатор створеного замовлення. Також відбувається пошук у базі даних запису перекладача з даним ідентифікатором та створення посилання на створене замовлення у полі замовлень, які виконував

даний перекладач. Останнім кроком відбувається збереження екземпляру моделі створеного замовлення у колекції замовлень бази даних.

Запит на підтвердження замовлення відправляється клієнтом коли перекладач підтверджує свою участь у його виконанні. Такий запит має містити ідентифікатор замовлення. Після отримання такого запиту відбувається пошук у базі даних замовлення з таким ідентифікатором та у полі статусу даного замовлення вказується те, що воно підтверджене. Після цього відбувається збереження зміненого запису даного замовлення у базі даних.

Запит на відправку виконаного замовлення відправляється клієнтом коли перекладач заповнює поле посилання на файл з виконаним замовлення. Такий запит має містити ідентифікатор замовлення та посилання на файл з виконаним замовлення. Після отримання такого запиту відбувається пошук у базі даних замовлення з таким ідентифікатором та у полі статусу вказується те, що замовлення виконане, у поле посилання на виконане дане замовлення поміщується посилання, отримане з тіла запиту, також у поле дати відправки виконаного замовлення записуються поточні дата та час. Після цього відбувається збереження зміненого запису даного замовлення у базі даних.

Запит на підтвердження правильності виконання замовлення відправляється клієнтом коли замовник після перевірки файлу з виконаним замовленням підтверджує правильність виконання. Такий запит має містити ідентифікатор замовлення. Після отримання такого запиту відбувається пошук у базі даних замовлення з таким ідентифікатором та у полі статусу вказується те, що правильність виконання замовлення підтверджена замовником. Після цього відбувається збереження зміненого запису даного замовлення у базі даних.

Запит на отримання списку усіх замовлень користувача має містити ідентифікатор користувача. Після отримання такого запиту сервер виконує пошук одного запису з таким ідентифікатором у колекції користувачів бази даних, а також вбудовує у поле посилань на замовлення цього користувача

записи замовлень, де користувач є замовником. Також у поле виконавця у кожному записі замовлення вбудовується запис з усіма даними перекладача, який є виконавцем цього замовлення. Після цього сервер надсилає клієнту об’єкт з цим списком замовлень у відповідь на успішне виконання запиту.

Запит на отримання списку усіх замовлень, у яких бере участь даний перекладач має містити ідентифікатор перекладача. Після отримання такого запиту сервер виконує пошук одного запису з таким ідентифікатором у колекції перекладачів бази даних, а також вбудовує у поле посилань на замовлення, у яких даний перекладач є виконавцем, записи замовлень. Також у поле замовника у кожному записі замовлення вбудовується запис з даними користувача, який є замовником у цьому замовленні. Після цього сервер надсилає клієнту об’єкт з цим списком замовлень у відповідь на успішне виконання запиту.

Останнім кроком потрібно підключити ці роутери до застосунку сервера у файлі ініціалізації.

Отже, сервер може обробляти усі необхідні для роботи веб-застосунка запити, що надходять з клієнтської частини, а саме – запити авторизації користувача надання даних про користувачів і перекладачів, а також запити оперування замовленнями та їх статусами.

3.3 Розробка клієнтської частини

Як було описано у частині 2.3, для розробки клієнтської частини я обрав React. Першим кроком у розробці React-застосунку є його ініціалізація. Для цього використовується модуль create-react-app. Він створює шаблон застосунка.

Як було описано у частині 3.1, для збереження отриманих з сервера даних і подальшого їх використання у застосунку, потрібно створити екземпляр сховища Redux. Для цього спочатку потрібно створити редюсери користувача, перекладача та замовлення.

У файлі редюсера користувача містяться: дві константи з назвами типів екшенів, константа, що містить початковий стан, функція редюсера та дві функції генераторів екшенів. Перша константа містить назву типу екшену збереження у сховище даних поточного користувача, а друга константа – назву типу екшену виходу з акаунту. Константа початкового стану містить порожній об’єкт даних поточного користувача та булеву змінну поточного стану авторизації користувача, у початковому стані вона має значення false. Функція редюсера користувача приймає стан, за замовчуванням початковий, та екшн і залежно від типу екшену повертає новий стан, змінюючи переданий у неї. У разі, якщо типом екшену, переданого у цей редюсер, є збереження даних поточного користувача, у об’єкт даних поточного користувача передаються дані, отримані від екшену, а булева змінна поточного стану авторизації користувача змінює значення на true, потім редюсер повертає цей змінений стан. А у разі, якщо типом переданого у редюсер екшену є вихід з акаунту, то відбувається видалення з сховища браузера токену користувача та повернення редюсером початкового стану, де об’єкт даних поточного користувача є порожнім, а булева змінна поточного стану користувача має значення false. Перша функція генератора екшену відповідає за збереження даних нового користувача. Вона приймає на вхід об’єкт з цими даними та створює екшн, який містить константу назви типу екшену збереження даних поточного користувача та самі отримані на вхід дані. Друга функція генератора екшену відповідає за вихід користувача з акаунту, тому вона не приймає на вхід дані та створює екшн, який містить тільки константу назви типу екшену виходу з акаунту.

Файлі редюсера перекладача має аналогічну структуру, у ньому містяться: дві константи з назвами типів екшенів, константа, що містить початковий стан, функція редюсера та дві функції генераторів екшенів. Перша константа містить назву типу екшену збереження списку перекладачів, а друга константа – назву типу екшену збереження поточного перекладача. Константа початкового стану

містить порожній масив перекладачів та порожній об’єкт поточного перекладача. Функція редюсера перекладача приймає стан, за замовчуванням початковий, та екшн і залежно від типу екшену повертає новий стан, змінюючи переданий у неї. У разі, якщо типом екшену, переданого у цей редюсер, є збереження списку перекладачів, у масив перекладачів передаються дані, отримані від екшена, потім редюсер повертає цей змінений стан. У разі, якщо типом переданого у редюсер екшену є збереження поточного перекладача, то у об’єкт поточного перекладача передаються отримані з екшену дані, потім редюсер повертає змінений стан. Перша функція генератора екшену відповідає за збереження списку перекладачів. Вона приймає на вхід масив перекладачів та створює екшн, який містить константу назви типу екшену збереження списку перекладачів та отриманий на вхід масив. Друга функція генератора екшену відповідає за збереження даних поточного перекладача. Вона приймає на вхід об’єкт з цими даними та створює екшн, який містить константу назви типу екшену збереження поточного перекладача та отриманий на вхід об’єкт.

Структура файлу редюсера замовлень має схожу структуру, він містить одну константу з назвою типу екшену, константу, що містить початковий стан, функцію редюсера та функцію генератора екшену. Перша константа містить назву типу екшену збереження списку замовлень. Константа початкового стану містить порожній масив замовлень. Функція редюсера замовлень приймає стан, за замовчуванням початковий, та екшн і залежно від типу екшену повертає новий стан, змінюючи переданий у неї. У разі, якщо типом екшену, переданого у цей редюсер, є збереження списку замовлень, у масив замовлень передаються дані, отримані від екшену, потім редюсер повертає цей змінений стан. Функція генератора екшену приймає на вхід масив замовлень та створює екшн, який містить константу назви типу екшену збереження списку замовлень та отриманий на вхід масив.

Після реалізації редюсерів створив файл ініціалізації сховища Redux. У ньому об’єднав розроблені редюсери у головний редюсер, передаючи об’єкт зі створеними редюсерами у метод combineReducers. Потім створив сховище, використовуючи метод configureStore, у який на вхід передається об’єкт з створеним головним редюсером.

Наступним кроком потрібно реалізувати необхідні функції роботи з сервером. Ці функції поділяються на три типи: функції, пов’язані з користувачем, функції, пов’язані з перекладачами та функції, пов’язані з замовленнями. Для покращення структури проекту, що дозволить уникнути помилок, я реалізував функції кожного типу у окремих файлах.

У файлі функцій, що пов’язані з користувачем реалізував такі функції: функцію реєстрації, функцію входу в акаунт та функцію аутентифікації. Функція реєстрації приймає на вхід адресу електронної пошти, пароль, об’єкт з іменем та прізвищем, роль та масив мов. Далі за допомогою методу для надсилання запитів, що надає модуль axios, надсилається запит на адресу обробника запитів реєстрації сервера веб-застосунку. У тілі цього запиту надсилаються отримані вхідні дані. Функція входу у акаунт приймає на вхід адресу електронної пошти та пароль. Ця функція повертає нову асинхронну функцію. У ній за допомогою методу для надсилання запитів надсилається запит на адресу обробника запитів входу сервера веб-застосунку. Далі викликається генератор екшену збереження даних поточного користувача, у який на вхід передаються дані користувача з отриманої від сервера відповіді. Після цього відбувається збереження у сховище браузера JWT-токена даного користувача, що також міститься у відповіді від сервера. Функція аутентифікації не приймає на вхід дані. Вона також повертає нову асинхронну функцію. У ній надсилається запит на адресу обробника запитів аутентифікації сервера веб-застосунку. У тілі цього запиту не містяться дані, проте запит має заголовок авторизації, у якому міститься JWT-токен даного користувача, що зберігається у сховищі браузера. Далі викликається генератор

екшену збереження даних поточного користувача, у який на вхід передаються дані користувача з отриманої від сервера відповіді. Після цього відбувається збереження у сховище браузера JWT-токена даного користувача, що також міститься у відповіді від сервера.

У файлі функцій, що пов’язані з перекладачами реалізував такі функції: функцію отримання списку перекладачів та функцію отримання даних конкретного перекладача. Функція отримання списку перекладачів не отримує на вхід дані. Ця функція повертає нову асинхронну функцію. У ній надсилається запит на адресу обробника запитів отримання списку перекладачів сервера веб-застосунку. Після цього викликається генератор екшену збереження списку перекладачів, у який на вхід передаються дані списку перекладачів з отриманої від сервера відповіді. Функція отримання даних конкретного перекладача отримує на вхід ідентифікатор перекладача. Вона повертає нову асинхронну функцію. У ній надсилається запит на адресу обробника запитів отримання даних конкретного перекладача серверу веб-застосунку. Далі викликається генератор екшену збереження даних поточного перекладача, у який на вхід передаються дані перекладача з отриманої від сервера відповіді.

У файлі функцій, що пов’язані з замовленнями реалізував такі функції: функцію пропонування роботи перекладачу, функцію підтвердження замовлення, функцію надсилання виконаного замовлення, функцію підтвердження правильності виконання замовлення та функцію отримання списку замовлень. Функція пропонування роботи перекладачу приймає на вхід ідентифікатор користувача, який є замовником, ідентифікатор перекладача та об’єкт з даними замовлення. У ній надсилається запит на адресу обробника запитів пропонування роботи серверу веб-застосунку. У тілі запиту містяться ідентифікатор замовника, ідентифікатор перекладача та об’єкт з даними замовлення. Функція підтвердження замовлення приймає на вхід ідентифікатор замовлення. У ній надсилається запит на адресу обробника запитів

підтвердження замовлення серверу веб-застосунку. У тілі запиту міститься ідентифікатор замовлення. Функція надсилання виконаного замовлення приймає на вхід ідентифікатор замовлення та посилання на файл з виконаним замовленням. У ній надсилається запит на адресу обробника запитів надсилання виконаного замовлення серверу веб-застосунку. У тілі запиту міститься ідентифікатор замовлення та посилання на файл з виконаним замовленням. Функція підтвердження правильності виконання замовлення приймає на вхід ідентифікатор замовлення. У ній надсилається запит на адресу обробника запитів підтвердження правильності виконання замовлення серверу веб-застосунку. У тілі запиту міститься ідентифікатор замовлення. Функція отримання списку замовлень приймає на вхід роль користувача та його ідентифікатор. Вона повертає нову асинхронну функцію. Виконання цієї функції залежить від ролі користувача. У випадку, якщо користувач є перекладачем, то надсилається запит, у тілі якого міститься ідентифікатор користувача, на адресу обробника запитів отримання списку замовлень, у яких користувач є перекладачем, серверу веб-застосунку далі викликається генератор екшену збереження списку замовлень, у який на вхід передається список замовлень з отриманої від сервера відповіді. У іншому випадку, якщо користувач не є перекладачем, виконуються аналогічні дії, але запит відправляється на адресу обробника запитів отримання списку замовлень сервера веб-застосунку, у яких користувач є замовником.

Далі можна приступити до розробки головної сторінки застосунка. Для цього у папці, де будуть міститись усі сторінки застосунка я створив файл з шаблоном React-компонента. Першим компонентом, який має розміщуватись на головній сторінці є навігаційна панель, де будуть міститись логотип застосунка та кнопка входу якщо користувач ще не ввійшов у свій акаунт, якщо він це зробив, то замість кнопки входу мають бути кнопки переходу на сторінку роботи з замовленнями та кнопка виходу з акаунта. Для реалізації навігаційної панелі у папці, де будуть міститись усі компоненти, створив папку навігаційної панелі. У

ній створив новий файл з шаблоном React-компонента. Для перевірки стану авторизації користувача необхідно отримати змінну поточного стану авторизації користувача, що міститься у стані користувача сховища Redux. Для цього використовується хук useSelector. Хуки у React – це функції, які дають можливість “підчепитись” до стану та методів життєвого циклу з функціонального компонента[24]. У сегменті цього файла, де має міститись код HTML-шаблона компонента створив необхідні теги для логотипа застосунка та умовну конструкцію. У випадку якщо змінна поточного стану авторизації користувача має значення false, ця конструкція вставляє у HTML-шаблон код кнопки входу у акаунт. У іншому випадку, якщо ця змінна має значення true, вставляються посилання на сторінку роботи з замовленнями та кнопка виходу з акаунту, яка при натисканні використовує хук useDispatch для передачі змін стану у сховище, у цей хук на вхід передається раніше реалізована функція виходу з акаунту. Наступним компонентом, з якого складається головна сторінка, є список перекладачів. Для отримання списку перекладачів я використав хук useEffect, який дозволяє виконувати певні діє при завантаженні сторінки. У нього передається диспатч, у який на вхід передається раніше створена функція завантаження списку перекладачів. Після виконання цієї функції список перекладачів зберігається у сховище, тому для його отримання зі сховища потрібно використати хук useSelector. Сам компонент списку перекладачів я реалізував у окремому файлі. У файлі головної сторінки отриманий зі сховища список перекладачів передається у компонент списку перекладачів. У файлі компоненту списку перекладачів для кожного елементу отриманого з головної сторінки списку створюється компонент картки перекладача, у який передається елемент зі списку перекладачів. У HTML-шаблоні картки перекладача я створив необхідні теги для відображення повного імені перекладача, який передається у цей компонент з компоненту списку перекладачів, списку мов, з якими може працювати перекладач, його рейтингу

та кнопки пропонування роботи цьому перекладачу, яка при натисканні буде відкривати сторінку пропонування роботи з формою для вводу даних про замовлення. Але для пропонування роботи користувач має увійти у свій акаунт, або створити його, тому далі необхідно створити сторінки входу та реєстрації.

У HTML-шаблоні файлу з компонентом сторінки входу я додав компоненти навігаційної панелі та форми входу, а також посилання на сторінку реєстрації. У HTML-шаблоні компоненту форми входу створив поля для вводу адреси електронної пошти та паролю користувача, а також кнопку, при натисканні на яку відбувається диспатч раніше реалізованої функції входу, у яку на вхід передаються значення, отримані з полів для вводу електронної пошти та паролю користувача.

Сторінка реєстрації має подібну структуру. У ній містяться компонент навігаційної панелі, форма реєстрації та посилання на сторінку входу. У HTML-шаблоні файлу компоненту форми реєстрації я створив поля вводу імені, прізвища, адреси електронної пошти користувача, випадаючий список з ролями користувача – роллю користувач та роллю перекладача. Далі йде умовна конструкція, завдяки якій у разі, якщо користувач обирає роль перекладача, у HTML-шаблон вбудовується поле для вводу мов, з якими користувач може працювати як перекладач. Після цього йде поле для вводу паролю та кнопка реєстрації, при натисканні якої викликається проміжна функція, у якій відбувається форматування деяких отриманих з полів вводу даних та викликається раніше реалізована функція реєстрації.

Наступним кроком необхідно реалізувати компонент сторінки для пропонування роботи перекладачу. У тілі компонента з параметрів переходу на цей компонент отримується ідентифікатор перекладача, якому користувач хоче запропонувати роботу, та за допомогою хука useEffect виконується диспатч раніше реалізованої функції отримання даних конкретного перекладача за його ідентифікатором, на вхід у цю функцію передається ідентифікатор даного

перекладача. Потім за допомогою хука useSelector відбувається отримання з сховища даних поточного користувача та поточного перекладача. У HTML-шаблоні сторінки я додав компонент навігаційної панелі та форму заповнення даних замовлення. Перший елемент цієї форми – це два випадаючих списки з мовами, з у якими може працювати перекладач. У першому списку користувач обирає мову оригіналу замовлення, у другому – мову, на яку потрібно перекласти замовлення. Далі йде текстове поле для опису замовлення. За ним йде поле для посилання на файл, вміст якого потрібно перекласти. Потім йде елемент вибору дати й часу, до якого замовлення має бути виконане. Далі йде поле для вводу ціни, яку замовник заплатить за виконання замовлення. Останнім елементом форми є кнопка, при натисканні на яку викликається функція обробника натискання, у якій відбувається отримання значень з усіх полів та елементів форми, створюється об’єкт даних замовлення та викликається раніше розроблена функція пропонування замовлення.

Далі потрібно реалізувати компонент останньої сторінки, сторінки роботи з замовленнями. У тілі компонента спершу за допомогою хука useSelector зі сховища отримуються дані поточного користувача. Далі з використанням хука useEffect відбувається диспатч раніше створеної функції отримання списку замовлень, у яку передаються роль поточного користувача та його ідентифікатор. Потім з використанням хука useSelector зі сховища отримується список замовлень даного користувача. У HTML-шаблоні містяться компонент навігаційної панелі та компонент списку замовлень, у який передається сам список замовлень, отриманий у тілі компонента. У HTML-шаблоні компонента списку замовлень для кожного елементу отриманого з компонента сторінки роботи з замовленнями масиву створюється компонент картки замовлення. У нього передається елемент з масиву замовлень. Картка замовлення може надавати різні можливості управління замовленням користувачу залежно від його ролі у виконанні цього замовлення. Для реалізації цього, у тілі компонента

картки замовлення за допомогою хука useSelector зі сховища отримуються дані поточного користувача. У HTML-шаблоні містяться опис даного замовлення, посилання на файл, вміст якого треба перекласти, мова, з якої потрібно перекласти та мова, на яку потрібно перекласти, статус замовлення, дату коли замовник запропонував цю роботу перекладачу, дату, до якої замовлення має бути виконане, ціну, яку заплатить замовник за виконання. Далі йдуть умовні конструкції, які відповідають за відображення даних імені замовника та перекладача. Якщо роллю користувача є користувач, він є замовником, тому відображаються дані з іменем перекладача, який є виконавцем замовлення. Якщо роллю користувача є перекладач, він є виконавцем замовлення, тому відображаються дані імені користувача, який є замовником. Далі йде наступна умовна конструкція – якщо роллю користувача є перекладач, а статусом замовлення є те, що ця робота запропонована перекладачу, відображається кнопка підтвердження замовлення, при натисканні якої викликається функція обробника натискання, у тілі якої викликається раніше реалізована функція підтвердження замовлення, у неї на вхід передається ідентифікатор даного замовлення. Далі йде наступна умовна конструкція – якщо роллю користувача є перекладач, статус замовлення – замовлення підтверджене, відображаються поле вводу посилання на файл з виконаним замовленням та кнопка, при натисканні якої викликається функція обробника, у тілі якої викликається раніше створена функція надсилання виконаного замовлення, у неї на вхід передаються ідентифікатор даного замовлення та значення поля вводу посилання на файл з виконаним замовленням. Після цього йде остання умовна конструкція – якщо роллю користувача є користувач, тобто даний користувач є замовником, а статус замовлення – замовлення виконане, відображається посилання на файл з виконаним замовленням, а також кнопка, при натисканні якої викликається функція обробника, у тілі якої викликається раніше реалізована функція

підтвердження замовлення, у неї на вхід передається ідентифікатор даного замовлення.

Після реалізації усіх компонентів останнім кроком у розробці клієнтської частини веб-застосунку підтримки взаємодії перекладачів і клієнтів є налаштування маршрутів. Для цього потрібно у HTML-шаблоні файлу компоненту застосунку тег списку маршрутів, а всередині цього тегу створити тег маршруту, у який передається компонент сторінки, та URL-адреса.

3.4 Інструкція користувача

Так, як у розробленому веб-застосунку підтримки взаємодії перекладачів і клієнтів користувачі поділяються на дві ролі, інструкцію користувача теж можна розділити на інструкцію користувача, який є клієнтом та інструкцію користувача, який є перекладачем.

3.4.1 Інструкція користувача, який є клієнтом

Процес роботи клієнта з веб-застосунком підтримки взаємодії перекладачів і клієнтів розділений на шість етапів (рисунок 3.4.1.1): реєстрація та вхід у акаунт, вибір перекладача, заповнення даних замовлення, очікування підтвердження замовлення, очікування виконання замовлення та підтвердження правильності виконання замовлення.

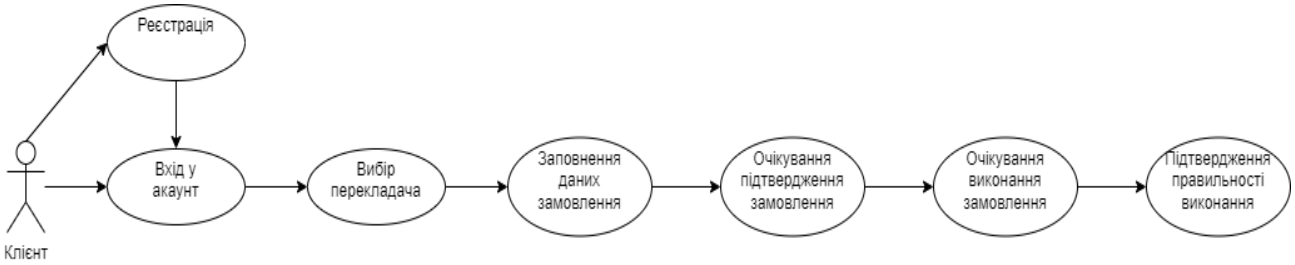


Рисунок 3.4.1.1 – Діаграма процесу роботи клієнта у веб-застосунку

На першому етапі клієнт має зареєструватись та увійти в акаунт. Для цього спочатку потрібно перейти на сторінку реєстрації. Там у відповідних полях потрібно ввести своє ім’я та прізвище, адресу електронної пошти. Після цього у випадаючому списку ролі потрібно обрати роль користувача. Далі ввести пароль та натиснути на кнопку реєстрації. У разі успішної реєстрації користувача з’явиться відповідне повідомлення, у іншому випадку – повідомлення про помилку. Адреса електронної пошти має бути дійсною, а пароль має містити від трьох до десяти символів. Після успішної реєстрації необхідно увійти у акаунт. Для цього на сторінці входу у відповідних полях потрібно ввести адресу електронної пошти, яка використовувалась для реєстрації, та пароль від акаунту та натиснути на кнопку входу.

Після успішного входу в акаунт потрібно обрати перекладача, якому буде запропоноване виконання замовлення. Для цього потрібно перейти на головну сторінку, яка містить список перекладачів. Для виконання замовлення на переклад, перекладач має знати щонайменше дві мови – мову оригіналу тексту, що потрібно перевести, та мову, на яку потрібно перевести замовлення. Після обрання зі списку перекладача, який знає обидві необхідні мови потрібно натиснути на кнопку “Offer”, тобто “запропонувати”. Після цього відкриється сторінка з формою для заповнення даних замовлення.

У формі заповнення даних замовлення спочатку потрібно обрати у випадаючих списках мову, з якої потрібно перевести замовлення та мову, на яку потрібно перевести замовлення. Далі у відповідних полях потрібно ввести опис замовлення, URL-адресу, де міститься файл, вміст якого потрібно перевести та ціну. Також потрібно обрати дату та час, до якого перекладач має виконати замовлення. Після заповнення всіх даних потрібно натиснути на кнопку “Offer”. Далі статус замовлення можна відслідковувати на сторінці роботи з замовленнями, для того, щоб перейти на неї, потрібно натиснути на кнопку “My

works” у правій частині навігаційної панелі. Коли перекладач підтвердить замовлення, поле статусу замовлення зміниться на “confirmed”. Далі, коли перекладач виконає та надішле виконане замовлення, статус замовлення зміниться на “done” та посилання на нього з’явиться під інформацією про замовлення. Після цього потрібно перевірити правильність виконання замовлення та, якщо все вірно, підтвердити правильність виконання, натиснувши на кнопку “Apply”, що з’явиться під посиланням на виконане замовлення.

3.4.2 Інструкція користувача, який є перекладачем

Процес роботи перекладача з веб-застосунком підтримки взаємодії перекладачів і клієнтів розділений на п’ять етапів (рисунок 3.4.2.1): реєстрація та вхід у акаунт, отримання пропозиції роботи, підтвердження участі у замовленні, відправка виконаного замовлення та очікування підтвердження правильності виконання замовлення.

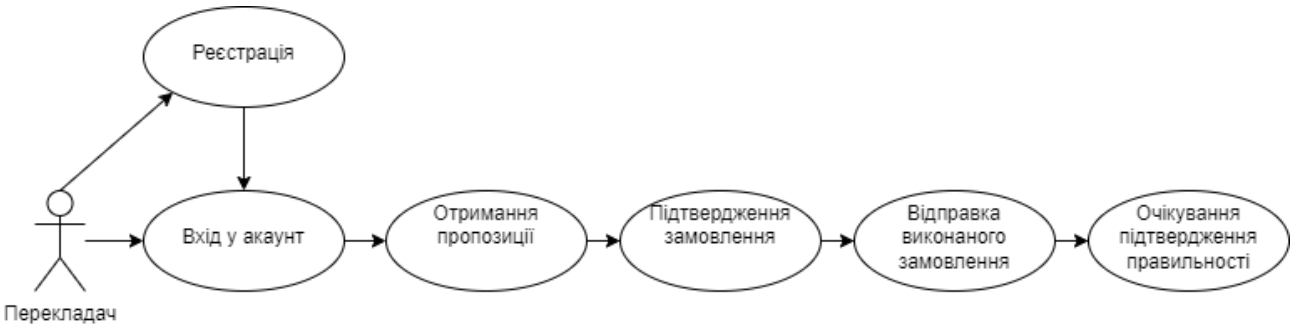


Рисунок 3.4.2.1 – Діаграма процесу роботи клієнта у веб-застосунку

Спочатку перекладач має зареєструватись. Для цього потрібно перейти на сторінку реєстрації. У відповідних полях форми реєстрації необхідно ввести ім’я та прізвище, адресу електронної пошти. Далі у випадаючому списку потрібно обрати роль перекладача. Після цього під випадаючим списком має з’явитись поле для вводу мов, з якими може працювати перекладач, мов має бути не менше двох. Потім потрібно ввести пароль, його довжина повинна бути від трьох до

десяти символів, та натиснути на кнопку “Register”. Якщо реєстрація буде успішною, з’явиться повідомлення про це, а якщо ні – з’явиться повідомлення з помилкою. Після того, як перекладач зареєструвався, його дані з’являються у списку перекладачів та він може отримувати пропозиції роботи.

Далі необхідно увійти у акаунт, для цього потрібно перейти на сторінку входу та ввести адресу електронної пошти, яка використовувалась для реєстрації та пароль. Після успішного входу потрібно перейти на сторінку роботи з замовленнями. Кнопка для переходу міститься у правому кутку навігаційної панелі та має напис “My works”.

На сторінці роботи з замовленнями відображається список карток з замовленнями. У картці замовлення відображаються такі дані: опис замовлення, посилання на файл, вміст якого потрібно перевести, мова оригіналу, з якої потрібно перевести, та мова, на яку потрібно перевести, статус замовлення, дата пропонування перекладачу цього замовлення, дата, до якої замовлення має бути виконане, ціна та ім’я замовника.

У нових запропонованих замовлень статусом є “offered”, тобто “запропоновано”. Також картка запропонованого замовлення має кнопку для його підтвердження.

Після підтвердження замовлення його статус зміниться на “confirmed”, а в нижній частині картки з’явиться поле для вводу посилання на файл з виконаним замовленням. Далі перекладач має приступати до виконання цього замовлення.

Коли виконання замовлення буде закінчене, перекладач має ввести посилання на файл з результатом виконання у поле вводу під даними цього замовлення та натиснути на кнопку “Send”, що знаходить під цим полем. Статус замовлення після цього змінить на “done”.

Останнім етапом процесу роботи перекладача з замовленням у веб-застосунку підтримки взаємодії перекладачів і клієнтів є отримання підтвердження правильності виконання замовлення. Підтвердити правильність

виконання замовлення має клієнт, це відбувається після відправки перекладачем посилання на файл з виконаним замовленням. Клієнт перевіряє його та, якщо все вірно, підтверджує правильність виконання. Після цього статус замовлення замовлення змінюється на “applied”.

ВИСНОВКИ

У ході виконання бакалаврської роботи була досліджена теоретична база розробки веб-застосунків, було оглянуто існуючі рішення веб-застосунків підтримки взаємодії перекладачів і клієнтів та виявлені їх недоліки. З урахуванням виявлених недоліків було сформульовано список вимог до розроблюваного веб-застосунку та розроблене технічне завдання на розробку. Були розглянуті та проаналізовані сучасні підходи та засоби розробки веб-застосунків та були обрані підходи та засоби розробки для реалізації веб-застосунку підтримки взаємодії перекладачів і клієнтів. Було спроектовано та розроблено веб-застосунок, що є темою бакалаврської роботи, а також інструкції для користувача, що є клієнтом та користувача, що є перекладачем.

Розробка веб-застосунку підтримки взаємодії перекладачів і клієнтів проводилась у середі розробки програмного забезпечення Microsoft Visual Studio Code. Розробка велась на мові JavaScript з використанням фреймворку Express для реалізації серверної частини веб-застосунку та бібліотек React та Redux для реалізації клієнтської частини веб-застосунку. Також для реалізації веб-застосунку була використана документо-орієнтована база даних MongoDB.

Реалізований веб-застосунок підтримки взаємодії перекладачів і клієнтів дозволяє клієнтам обирати перекладачів для виконання замовлення, пропонувати їм роботу, відслідковувати стан замовлення на всіх етапах його виконання та підтверджувати правильність виконання замовлення.

Перекладачам реалізований веб-застосунок дозволяє приймати замовлення, виконувати їх та надсилати результат виконання замовлень клієнтам.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is A Web Application? [Електронний ресурс], URL: https://www.ramotion.com/blog/what-is-a-web-application/ (дата звернення: 17.02.2022);
2. XML-RPC for Newbies [Електронний ресурс], URL: http://scripting.com/davenet/1998/07/14/xmlRpcForNewbies.html (дата звернення: 19.02.2022);
3. Simple Object Acess Protocol (SOAP) 1.1 [Електронний ресурс], URL: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (дата звернення: 19.02.2022);
4. REST [Електронний ресурс], URL: https://restfulapi.net/ (дата звернення: 19.02.2022);
5. Express [Електронний ресурс], URL: https://expressjs.com/uk/ (дата звернення: 20.02.2022);
6. What is a Database [Електронний ресурс], URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 24.02.2022);
7. Database [Електронний ресурс], URL: https://en.wikipedia.org/wiki/Database#Classification (дата звернення: 24.02.2022);
8. What is a Hierarchical Database? Definition and FAQs [Електронний

ресурс], URL: https://www.heavy.ai/technical-glossary/hierarchical-database (дата звернення: 24.02.2022);

9. Relational Databases Explained [Електронний ресурс], URL: https://www.ibm.com/cloud/learn/relational-databases#toc-what-is-a--8q6isCrS (дата звернення: 24.02.2022);
10. No-SQL Databases – What They Are and Why you Need One[Електронний ресурс], URL: https://www.couchbase.com/resources/why-nosql (дата звернення: 24.02.2022);
11. Relational vs Non Relational Databases [Електронний ресурс], URL: https://towardsdatascience.com/relational-vs-non-relational-databases-f2ac792482e3 (дата звернення: 24.02.2022);
12. What is MongoDB? A definition from WhatIs.com [Електронний ресурс], URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (дата звернення: 28.02.2022);
13. What Is a Web Application? How It Works, Benefits and Examples [Електронний ресурс], URL: https://www.indeed.com/career-advice/career-development/what-is-web-application (дата звернення: 12.03.2022);
14. AngularJS: Developer Guide: Introduction [Електронний ресурс], URL: https://docs.angularjs.org/guide/introduction (дата звернення: 12.03.2022);
15. AngularJS - Overwiew [Електронний ресурс], URL: https://www.tutorialspoint.com/angularjs/angularjs_overview.htm (дата звернення: 12.03.2022);
16. AngularJS - Dependency Injection [Електронний ресурс], URL: https://www.tutorialspoint.com/angularjs/angularjs_dependency_injection.htm (дата звернення: 12.03.2022);
17. ReactJs - Overview [Електронний ресурс], URL: https://www.tutorialspoint.com/reactjs/reactjs_overview.htm (дата звернення:

12.03.2022);

18. The Missing Introduction to React [Електронний ресурс], URL: https://medium.com/javascript-scene/the-missing-introduction-to-react-62837cb2fd76 (дата звернення: 12.03.2022);
19. The Pros and Cons of Vue.js [Електронний ресурс], URL: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/ (дата звернення: 12.03.2022);
20. Angular vs React vs Vue: Which Framework Is Better? 2022 [Електронний ресурс], URL: https://athemes.com/guides/angular-vs-react-vs-vue/ (дата звернення: 12.03.2022);
21. Introduction to JSON Web Tokens [Електронний ресурс], URL: https://jwt.io/introduction (дата звернення: 25.04.2022);
22. Mongoose v6.3.8: Schemas [Електронний ресурс], URL: https://mongoosejs.com/docs/guide.html#schemas (дата звернення: 26.04.2022);
23. Mongoose v6.3.8: Models [Електронний ресурс], URL: https://mongoosejs.com/docs/models.html (дата звернення: 26.04.2022);
24. Hooks at a Glance [Електронний ресурс], URL: https://reactjs.org/docs/hooks-overview.html (дата звернення: 19.04.2022);

ДОДАТОК А

А.1 Лістинг коду обробника запитів авторизації:

```
const Router = require("express")
const bcrypt = require("bcryptjs")
const jwt = require("jsonwebtoken")
const {check, validationResult} = require("express-validator")
const User = require("../models/User")
const Translator = require("../models/Translator")
const config = require("config")
const authMiddleware = require('../middleware/auth.middleware')
const router = new Router()

router.post('/registration',
    [
        check('email', "Uncorrect email").isEmail(),
        check('password', "Password must be longer than 3").isLength(3,10)
    ],
    async (req, res) => {
    try {
        const errors = validationResult(req)
        if (!errors.isEmpty()) {
            return    res.status(400).json({message:    "Uncorrect    request",
errors})
        }
        const {email, password, name, role, language} = req.body

        const candidate = await User.findOne({email})
        if (candidate) {
            return res.status(400).json({message: `User with email ${email}
already exists`})
        }
```

```
        const hashPassword = await bcrypt.hash(password, 6)
        const user = new User({email, password: hashPassword, name, role})
        const userId = user._id
        await user.save()
        if (role === 'translator') {
            const translator = new Translator({userId, language})
            await translator.save()
        }
        return res.json({message: "User was created"})
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.post('/login',
    async (req, res) => {
    try {
       const {email, password} = req.body
       const user = await User.findOne({email})
       if (!user) {
           return res.status(404).json({message: "User not found"})
       }
       const isPassValid = bcrypt.compareSync(password, user.password)
       if (!isPassValid) {
            return res.status(400).json({message: "Invalid password"})
       }
       const   token   =   jwt.sign({id:   user.id},   config.get("secretKey"),
{expiresIn: "1h"})
       return res.json({
           token,
           user: {
            id: user.id,
            email: user.email,
            name: {
                firstName: user.name.firstName,
                lastName: user.name.lastName
            },
            role: user.role,
            works: user.works
           }
```

```
        })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.get('/auth', authMiddleware,
    async (req, res) => {
    try {
        const user = await User.findOne({_id: req.user.id})
        const  token  =  jwt.sign({id:  user.id},  config.get("secretKey"),
{expiresIn: "1h"})
       return res.json({
           token,
           user: {
            id: user.id,
            email: user.email,
            role: user.role
           }
       })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

module.exports = router
```

А.2 Лістинг коду обробника запитів перекладачів:

```
const Router = require("express")
const Translator = require("../models/Translator")
const User = require("../models/User")
const router = new Router()

router.get('/all',
    async (req, res) => {
    try {
        const translators = await Translator.find({}).populate('userId')
       return res.json({
```

```
            translators
        })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.get('/listbylanguage',
    async (req, res) => {
    try {
        const {language} = req.body
        const translators = await Translator.find({"language": language})
        return res.json({
            translators
        })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.post('/translatorinfobyid',
    async (req, res) => {
    try {
        console.log(req)
        const {id} = req.body
        const translator = await Translator.findOne({"_id":
id}).populate('userId')
        return res.json({
            translator
        })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

module.exports = router
```

А.3 Лістинг коду обробника запитів замовлень:

```
const Router = require("express")
const Translator = require("../models/Translator")
const User = require("../models/User")
const router = new Router()

router.get('/all',
    async (req, res) => {
    try {
        const translators = await Translator.find({}).populate('userId')
       return res.json({
           translators
       })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.get('/listbylanguage',
    async (req, res) => {
    try {
        const {language} = req.body
        const translators = await Translator.find({"language": language})
       return res.json({
           translators
       })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

router.post('/translatorinfobyid',
    async (req, res) => {
    try {
        console.log(req)
        const {id} = req.body
        const translator = await
Translator.findOne({"_id":id}).populate('userId')
```

```
        return res.json({
            translator
        })
    } catch (e) {
        console.log(e)
        res.send({message: "server error"})
    }
})

module.exports = router
```

ДОДАТОК Б

Б.1 Лістинг коду функцій надсилання запитів авторизації:

```
import axios from 'axios'
import { setUser } from '../reducers/userReducer'


export const registration = async (email, password, name, role, language = [])
=> {
    try {
        const response = await
axios.post(`http://localhost:5000/api/auth/registration`, {
            email,
            password,
            name,
            role,
            language
        })
        alert(response.data.message)
    } catch (e) {
        alert(e)
    }

}

export const login = (email, password) => {
    return async dispatch =>  {
        try {
            const response = await
axios.post(`http://localhost:5000/api/auth/login`, {
                email,
```

```
                password
            })
            dispatch(setUser(response.data.user))
            localStorage.setItem('token', response.data.token)
        } catch (e) {
            alert(e)
        }
    }
}

export const auth = () => {
    return async dispatch =>  {
        try {
            const response = await
axios.get(`http://localhost:5000/api/auth/auth`,
                {headers:                                {Authorization:`Bearer
${localStorage.getItem('token')}`}}
            )
            dispatch(setUser(response.data.user))
            localStorage.setItem('token', response.data.token)
        } catch (e) {
            localStorage.removeItem('token')
        }
    }
}
```

Б.2 Лістинг коду функцій надсилання запитів роботи з перекладачами:

```
import axios from 'axios'
import      {      setTranslators,      setCurrentTranslator      }      from
'../reducers/translatorReducer'


export const getTranslatorsList = () => {
    return async dispatch =>  {
        try {
            const response = await
axios.get(`http://localhost:5000/api/translators/all`)
            dispatch(setTranslators(response.data.translators))
        } catch (e) {
            alert(e)
```

```
        }

    }
}

export const getTranslatorInfo = (id) => {
    return async dispatch =>  {
        try {
            const response = await
axios.post(`http://localhost:5000/api/translators/translatorinfobyid`, {
                id
            })
            dispatch(setCurrentTranslator(response.data.translator))
        } catch (e) {
            alert(e)
        }

    }
}
```

Б.3 Лістинг коду функцій надсилання запитів роботи з замовленнями:

```
import axios from 'axios'
import { setWorks } from '../reducers/workReducer'


export const offer = async (customerId, translatorId, data) => {
    try {
        const response = await
axios.post(`http://localhost:5000/api/work/offer`, {
            "customer": customerId,
            "translator": translatorId,
            "data": data
        })
        alert(response.data.message)
    } catch (e) {
        alert(e)
    }

}
```

```
export const confirm = async (workId) => {
    try {
        const response = await
axios.post(`http://localhost:5000/api/work/confirm`, {
            "id": workId
        })
        alert(response.data.message)
    } catch (e) {
        alert(e)
    }

}

export const done = async (workId, doneUrl) => {
    try {
        const                response                =                await
axios.post(`http://localhost:5000/api/work/done`, {
            "id": workId,
            "doneurl": doneUrl
        })
        alert(response.data.message)
    } catch (e) {
        alert(e)
    }

}

export const apply = async (workId) => {
    try {
        const                response                =                await
axios.post(`http://localhost:5000/api/work/apply`, {
            "id": workId
        })
        alert(response.data.message)
    } catch (e) {
        alert(e)
    }

}
```

```
export const getWorkList = (role, id) => {
    return async dispatch =>  {
        try {
            if (role === 'user') {
                const response = await
axios.post(`http://localhost:5000/api/work/ofuser`, {id})
                dispatch(setWorks(response.data.user))
            }
            if (role === 'translator') {
                const response = await
axios.post(`http://localhost:5000/api/work/oftranslator`, {id})
                dispatch(setWorks(response.data.translator))
            }
        } catch (e) {
            console.log(e)
        }
    }
}
```