МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ ЗАМОВЛЕННЯМИ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.А. Соболєв
(ініціали та прізвище)

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

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

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Соболєву Івану Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи керування замовленнями

керівник роботи _____
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 27.11.2023 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інформаційної системи керування замовленнями

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.05.2023	
2.	Збір вихідних даних.	05.06.2023	
3.	Обробка методичних та теоретичних	24.07.2023	
	джерел.		
4.	Розробка першого та другого розділу.	25.09.2023	
5.	Розробка третього розділу.	13.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ І.А. Соболєв
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інформаційної системи керування замовленнями»: 49 с., 33 рис., 8 джерел.

БАЗА ДАНИХ, СФЕРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ, УПРАВЛІННЯ ПОСЛУГАМИ, JAVASCRIPT, MYSQL, REACT.

Об’єкт дослідження – процес розробки веборієнтовної інформаційної системи.

Мета роботи: розробка веборієнтовної інформаційної системи для управління послугами в сфері інформаційних технологій для оптимізації надання та отримання послуг організаціями та покращення робочих процесів за рахунок отриманих послуг.

Метод дослідження – порівняльний аналіз, розробка бази даних, клієнтських та серверних компонентів на основі React в середовищі WebStorm, обробка літературних джерел.

Основні сторінки цієї системи будуть розроблені за допомогою HTML, CSS та React разом з JavaScript. Включатимуть головну сторінку вебзастосунку, сторінки реєстрації та авторизації користувачів, відображення оформлених замовлень на сайті та IT-організацій, що відповідають за їх виконання. Усі ці елементи будуть реалізовані за допомогою React компонентів.

Також Apollo Client в React буде використаний для зберігання даних в базі, що дозволяє виконувати різні види запитів для отримання та оновлення інформації, надаючи користувачу актуальну інформацію.

Серверна частина, побудована на Node.js, відповідає за зберігання та обробку даних, які використовуються користувачами у вебдодатку.

SUMMARY

Master’s qualification paper “Development of the Orders Management Information System”: 49 pages, 33 figures, 8 references.

DATABASE, INFORMATION ECHNOLOGY, SERVICE ANAGEMENT, JAVASCRIPT, MYSQL, REACT.

The object of research is the process of developing orders management information system.

The purpose of the work is development of a information system for managing services in the field of information technologies to optimize the provision and receipt of services by organizations and improve work processes due to the services received.

Research method is comparative analysis, development of a database, client and server components based on React in the WebStorm environment, processing of literary sources.

In the master’s qualification work, a web application was developed to optimize the receipt and provision of services in the field of information technologies, taking into account current application development technologies.

The main pages of this system will be developed using HTML, CSS, and React along with JavaScript. This will include the main page of the web application, user registration and authorization pages, display of orders placed on the site and IT organizations responsible for their fulfillment. All these elements will be implemented using React components.

Also, Apollo Client in React will be used to store data in the database, which allows performing various types of queries to retrieve and update information, providing the user with up-to-date information.

The server side, built on Node.js, is responsible for storing and processing data used by users in the web application.

ЗМІСТ

ВСТУП

Сучасні підприємства навряд чи змогли б існувати без інформаційних технологій. Розвиток практично неможливий без підтримки технічних компаній. Компанії, незалежно від розміру, постійно шукають IT-послуги для розширення чи удосконалення робочих процесів.

Технічна підтримка, залучаючи зовнішніх експертів, забезпечує стабільність роботи інформаційних систем.

Вона дає керівникам певні переваги:

- звільняє співробітників від необхідності витрачати час на розробку та підтримку IT-інфраструктури, дозволяючи сконцентруватися на важливіших справах;
- забезпечує якісне обслуговування від досвідчених фахівців;
- підвищує ефективність роботи відділу, оптимізуючи бізнес-процеси за чіткою стратегією;
- дозволяє клієнту повний контроль над послугами, оскільки всі деталі прописуються у договорі.

Залучення зовнішніх експертів значно підвищує стійкість IT-системи підприємства. Це допомагає уникнути простоїв через неполадки, роблячи бізнес більш конкурентоспроможним та гнучким.

Автоматизація процесів надання IT-послуг є актуальним напрямком в сучасній епохі.

1 ОГЛЯД СУЧАСНОГО СТАНУ СФЕРИ ІНФОРМАЦІЙНИХ ПОСЛУГ І ПОСТАНОВА ЗАДАЧІ

1.1 Основні види послуг в інформаційних технологіях

Послуги Інформаційних технологій це можливість отримання підприємствами доступу до технічних засобів та інформації, що застосовуються в операційних процесах та повсякденних завданнях. Часто ці послуги керуються командами досвідчених працівників сфери інформаційних технологій, які сприяють розвитку організацій у різних галузях промисловості.

Залежно від специфіки бізнесу, відділ, що відповідає за ІТ-послуги, може складатися як з внутрішніх, так і зовнішніх ІТ спеціалістів. Наприклад, у медичній галузі команда ІТ надає послуги, що сприяють функціонуванню баз даних лікарень та зручності використання для співробітників. У малих компаніях ІТ-послуги можуть включати облікові записи для даних та мережеву безпеку для онлайн транзакцій.

Існує багато ІТ-послуг, які полегшують роботу бізнесу та дозволяють працівникам ефективно взаємодіяти з технологіями, що сприяють виконанню їх обов’язків та спілкуванню один з одним.

Основні типи ІТ-послуг, що використовуються в багатьох організаціях, включають такі напрями.

Хмарні послуги, що надають різні можливості взаємодії з технологіями через Інтернет-платформи. Це забезпечує зберігання, доступ та використання інформації та програм. Завдяки хмарним сервісам працівники можуть отримувати доступ до інформації вдома або в офісі, а також віддалено використовувати внутрішні бізнес-програми.

Голосовий протокол через Інтернет (VoIP), що є інструментом комунікації для бізнесу. Він дозволяє проводити дзвінки та відправляти повідомлення через Інтернет, полегшуючи внутрішній зв’язок між офісами у

різних місцях.

Рішення для резервного копіювання, які забезпечують захист інформації, зберігаючи її на зовнішніх пристроях або в онлайн-платформах, таких як хмарні сервіси.

Послуги забезпечення мережевої безпеки, які захищають мережу компанії від несанкціонованого доступу та можуть включати побудову брандмауерів, встановлення антивірусного програмного забезпечення та інші заходи для захисту даних.

Послуги електронної пошти – компанії та організації широко використовують електронну пошту у різних комунікативних цілях. Вона використовується для індивідуальних повідомлень, оновлень для всієї команди та взаємодії з клієнтами, що робить її важливою для підтримки ділових зв’язків. IT-команди часто відстежують поштові облікові записи та рекомендують постачальників, які відповідають потребам організації у спілкуванні.

Інформаційна звітність – завдяки збору та моніторингу даних, IT-служби можуть надавати звіти про діяльність вашої організації. Ці звіти стосуються використання технологій та зберігання інформації, що допомагає зрозуміти ефективність послуг вашої компанії. Це важливий елемент для всіх членів команди для оновлення проєкту в цілому, а також для керівників, які контролюють прогрес та виконання завдань.

Дистанційна підтримка – багато IT-сервісів пропонують віддалену технічну підтримку. Наприклад, IT-служби можуть усувати несправності на пристроях через Інтернет. Це дозволяє швидко виявляти та виправляти проблеми на кількох пристроях, включаючи мобільні. Використання віддаленої підтримки може значно збільшити швидкість виправлення помилок та оновлення програм, забезпечуючи мінімальні перерви в роботі.

Програмне забезпечення як послуга (SaaS) – це програми, за які користувачі платять підписку. Це можуть бути текстові редактори чи бази

даних. Доступ до такого програмного забезпечення зазвичай здійснюється через веббраузер, аутентифікація відбувається за допомогою імені користувача та пароля. Такий спосіб дозволяє користувачам отримати доступ до програм через Інтернет, уникнувши необхідності встановлювати їх на своїх комп'ютерах. IT-служби можуть допомогти у підключенні та переконатися, що всім користувачам доступні необхідні функції. Зазвичай SaaS включає хмарні сервіси, що дозволяють команді працювати віддалено за допомогою власних пристроїв. Багато SaaS-підписок також включають технічну підтримку програмного забезпечення.

Розробка програмного забезпечення – одна з основних послуг IT. Це процес створення власних додатків, які відповідають потребам певного бізнесу. Стратегія розробки включає планування кожного кроку, починаючи від перевірки ідеї до запуску продукту.

Усунення несправностей та технічна підтримка – важливі IT-послуги, що включають виправлення проблем з програмами та онлайн-інструментами. Команди, які це роблять, надають допомогу користувачам у вирішенні технічних проблем або рекомендують подальші дії. IT-спеціалісти можуть надати допомогу у виправленні помилок, що з'являються при використанні програм.

Технологічна підготовка – навчання команди користуватися оновленою технологією та програмами, щоб вони могли ефективно працювати з новими інструментами. Багато IT-послуг включають навчання персоналу використанню технічних інструментів у повсякденній роботі.

Установка та обслуговування обладнання – це послуги, які включають встановлення, обслуговування та ремонт технічного обладнання. Це важливі профілактичні та відновлювальні заходи, що полегшують роботу обладнання, включаючи технічне обслуговування за контрактом і ремонт. Технічна підтримка також включає в себе онлайн та телефонну допомогу у вирішенні технічних проблем та установленні програмного забезпечення [1].

1.2 Постановка завдання

На сьогоднішній день більшість бізнес-процесів в організаціях пов’язані з інформаційними технологіями.

Оновлення та підтримка цих пристроїв та процесів потребує спеціалізованого вміння працівників, і в кожній організації може виникнути потреба в залученні висококваліфікованих фахівців.

Основним завданням є розробка інформаційно системи керування послугами в галузі інформаційних технологій. Ця інформаційна система дозволяє IT-компаніям пропонувати свої послуги та забезпечить можливість іншим організаціям спілкуватися з досвідченими спеціалістами в цій галузі.

Основні функції такої системи включають:

- реєстрацію організацій в сфері IT;
- реєстрацію організацій іншого характеру;
- опублікування замовлень на будь які послуги в сфері IT;
- обзнайомлення з існуючими замовленнями, які розміщені на ресурсі і чекають виконання;
- повний список зареєстрованих спеціалістів, з урахуванням досвіду роботи та використаних технологій;
- надсилання прохання компаніям, для виконання поставленої задачі.

Метою даної магістрерської роботи є розробка інформаційної системи для управління послугами в галузі інформаційних технологій, з надавання та отримання послуг організаціями і поліпшення робочих процесів завдяки доступним послугам.

Для досягнення цієї мети потрібно вирішити наступні завдання:

- провести аналітичний огляд предметної області, визначити основні види послуг в сфері IT;
- проаналізувати існуючі рішення для управління послугами та визначити їхні слабкі сторони;
- визначити технології та інструменти для розробки системи для

управління IT-послугами;

- визначити архітектуру системи для управління послугами та необхідний функціонал;
- спроєктувати базу даних системи;
- розробити інтерфейс системи;
- реалізувати функціональну частину системи.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура інформаційної системи

Архітектура вебзастосунків складається з різних компонентів (рис. 2.1), що відповідають за клієнтську та серверну сторони.

Компоненти клієнтської сторони формуються за допомогою JavaScript, HTML, CSS та забезпечують користувацький інтерфейс: панелі інформації, сповіщення, журнали активності та інші елементи. Цей код виконується у веббраузері, не потребуючи налаштувань операційної системи чи інших компонентів пристрою користувача.

Серверні компоненти створюються мовами програмування, такими як Java, .Net, NodeJs, Python тощо. Сервер складається з логіки застосунку та бази даних. Логіка – це центр операцій вебзастосунку, а база даних містить усю необхідну для додатку інформацію, наприклад, дані для входу користувачів.

Рисунок 2.1 – Складові компоненти архітектури інформаційних систем

Під час інтеграції компонентів утворюються чотири рівні:

– рівень представлення даних;

– рівень обслуговування даних;

– рівень бізнес-логіки;

– рівень доступу до даних.

Рівень представлення даних відображає інтерфейс для користувачів та спрощує взаємодію з компонентами. Він містить компоненти інтерфейсу для обробки та відображення даних додатку.

Рівень обслуговування даних отримує та передає дані між рівнями, захищаючи інформацію інформаційної системи.

Рівень бізнес логіки визначає логіку бізнес-операцій та відповідає за обробку запитів клієнтів, а також визначає шляхи доступу до даних.

Розподіл функціоналу між цими рівнями допомагає відокремити відповідальності компонентів та полегшує пошук помилок під час розробки та використання додатку [8].

2.2 Технології завдяки яким розробляється клієнтська складова

Для розробки клієнтської складової вебзастосунку для управління послугами в сфері інформаційних технологій були використані такі технології як HTML, CSS та бібліотека React, побудована на основі мови програмування JavaScript.

2.2.1 HTML

HTML (HyperText Markup Language) – це мова розмітки гіпертексту, яка використовується для створення основної структури вебсторінок. Елементи сторінки описуються за допомогою тегів: назва тегу відповідає за призначення та відображення елементу, а текст всередині тегу відповідає за наповнення контентом цього елементу.

У HTML документі обов’язковими є теги <html></html>, <head></head> і <body></body>. Тег html є обгорткою для структури сторінки, весь контент має бути описаний всередині нього. Тег head містить службову інформацію про сторінку, таку як назва вкладки у браузері, посилання на зовнішні файли, дані про розробника та початкові налаштування для зовнішнього вигляду. В тегу body знаходиться контент сторінки, що відображається користувачеві.

Найпоширенішими HTML тегами є:

– використовується для параграфів тексту;
– це контейнер для групування елементів;
– застосовується для обгортання окремих частин тексту;
– <table></table> використовується для створення таблиць;
– , служать для створення списків та їх елементів;
– <a> використовується для створення посилань на інші сторінки;
– <form></form> використовується для створення форми;
– <input /> використовується для полів введення у формі.

При розробці HTML файлів важливо використовувати семантику – правила, що оптимізують елементи сторінки для браузерів. Семантика HTML допомагає створювати зрозумілі елементи та допомагає у розпізнаванні структури сайту сервісами для людей з обмеженими можливостями.

Основними семантичними HTML тегами є:

– <header></header> – сторінка, де зазвичай розміщуються логотип та меню;
– <nav></nav> – використовується для навігаційного меню сайту;
– <section></section> – використовується для основних секцій на сторінці;
– <article></article> – використовується для окремих статей в секціях;
– <footer></footer> – футер, де містяться посилання на контакти спеціаліста або компанії, інформація о правах.

Для кожного тегу можна використовувати додаткові атрибути, наприклад:

- class – використовується для визначення класу елементу для подальшої стилізації;
- id – це унікальний ідентифікатор елементу;
- href – це атрибут для посилання на ресурс;
- value – використовується для визначення початкового значення полів форми.

Усі ці теги відповідають за структуру та розміщення елементів на сторінці сайту. CSS використовується для оформлення зовнішнього вигляду кожного блоку [3].

2.2.2 CSS

CSS, або каскадні таблиці стилів, є інструментом для оформлення елементів на вебсторінці. Цей інструмент використовує різні типи селекторів, включаючи:

- універсальний селектор застосовує стилі до всіх елементів HTML сторінки;
- селектор тегів встановлює стилі для елементів за певним тегом;
- селектор класів застосовує стилі до елементів з певним значенням атрибуту class;
- селектор ідентифікаторів встановлює стилі для елементів з певним значенням атрибуту id;
- селектор атрибутів застосовує стилі до елементів з визначеними атрибутами;
- селектор дочірніх елементів встановлює стилі для дочірніх елементів певних блоків.

Кожен селектор має свою власну специфіку, що допомагає у вирішенні конфліктів. Проте виникають проблеми, коли до одного елемента застосовується декілька селекторів, і потрібно визначити, які саме стилі слід використовувати.

Також для оформлення елементів використовуються псевдо-класи та псевдо-елементи. Псевдо-класи встановлюють стилі для елементів у певних станах, таких як наведення курсору, перехід за посиланням, клік тощо. Псевдо-елементи відповідають за зовнішній вигляд певних частин інтерфейсу, таких як перша літера або лінія параграфу, елементи перед або після певних блоків.

Користуючись таблицями стилів, можна створити привабливий інтерфейс, що полегшить користування сайтом та покращить розташування елементів на сторінці.

Після створення HTML – структури сторінки та написання необхідних стилів важливо забезпечити можливість взаємодії користувачів з елементами. Наприклад, реакція на клік для відкриття нового вікна можлива за допомогою мови програмування JavaScript [4].

2.2.3 JavaScript

Основна мета JavaScript колись полягала в написанні коротких скриптів для додавання функціональності на вебсторінках у браузері. Проте сьогодні ця високорівнева мова використовується для створення складних веб та мобільних додатків.

Скрипти на JavaScript призначені для безпосередньої взаємодії з елементами сторінок або системи, не потребуючи окремого кроку компіляції. Під час виконання у браузері, JavaScript взаємодіє з Об’єктною Моделлю Документа (DOM), яка представляється як дерево з вузлів, кожен з яких є HTML елементом.

JavaScript є мультипарадигмальною мовою, що підтримує різні стилі

програмування, включаючи керування подіями, функціональне та імперативне програмування. Завдяки можливостям передачі функцій як аргументів та їх збереженню у структурах даних, JavaScript підтримує функції вищого порядку.

JavaScript відповідає стандарту ECMAScript, що створений Ecma. Починаючи з ECMAScript 2015 (також відомого як ES6), відбулися значні зміни у стандарті, що включили синтаксичні доповнення та нові можливості для створення складних програм.

Node.js, заснований на движку V8, надає середовище виконання JavaScript поза браузером. Він широко використовується як середовище виконання на сервері та на клієнтській стороні для створення вебдодатків. Node містить NPM, який дозволяє управляти пакетами та залежностями проєкту.

При розробці вебзастосунків розробники часто використовують бібліотеки та фреймворки JavaScript. Фреймворки мають вже визначену архітектуру, в той час як бібліотеки надають можливість використання необхідної функціональності, дозволяючи розробникам контролювати архітектуру за потреби.

2.2.4 React

React JS – це відкритий JavaScript-фреймворк, а точніше, бібліотекою JavaScript, яка використовується для розробки інтерфейсів користувача. Він був створений компанією Facebook і швидко набув популярності серед розробників з усього світу. Реакт дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у React JS є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за рендеринг певної частини користувацького інтерфейсу. React використовується для розробки користувацького інтерфейсу вебдодатків.

Основною метою React Frontend є створення інтерактивних, динамічних та відзивчивих інтерфейсів для користувачів. React Front end дозволяє створювати багатофункціональні та інтерактивні застосунки зі швидким рендерингом і перехідом між сторінками. React добре підходить для розробки проєктів будь-якого масштабу. Він надає можливості для легкого розширення та перевикористання компонентів, інтеграції з іншими бібліотеками та фреймворками. Також підтримує серверний рендеринг, що дозволяє поліпшити швидкість завантаження сторінок та оптимізувати пошукову оптимізацію.

Загалом використання бібліотеки дає можливість React developers ефективно будувати потужні та швидкі інтерфейси, полегшує роботу з компонентами та станом додатків, і має широку підтримку спільноти розробників. Він відмінно підходить для командної розробки завдяки дотриманню UI та шаблона робочого процесу. React developer відповідає за розробку, впровадження та підтримку користувацького інтерфейсу на основі React. Вони працюють з компонентами, станом, взаємодіють з сервером та API, тестують та оптимізують додаток для забезпечення високої продуктивності та якості.

Для роботи з React розробникам потрібні глибокі знання JavaScript. Також необхідне знаннями для React є HTML і CSS. Вони необхідні для створення компонентів та візуального оформлення, стилізації компонентів, розміщення елементів на сторінці та застосування візуальних ефектів.

Властивості компонента – це незмінні дані, які передаються йому від батьківського елементу при його створенні. Завдяки властивостям, компоненти стають більш гнучкими та можуть відрізнятися у вигляді та поведінці на основі отриманих даних.

У React дані передаються від батьківських до дочірніх компонентів через змінну props. У великих програмах React дерево компонентів може стати глибоким, що призводить до необхідності передачі функцій та даних на кілька рівнів.

У React існують два основних типи компонентів: класові та функціональні. Функціональні компоненти – це чисті функції, які приймають властивості як вхідні дані і повертають JSX-елементи. Приклад функціонального компоненту представлений на рис. 2.2.

```
const Greeting = (props) => {
  return <h1>Hello, {props.name}</h1>
}
```

Рисунок 2.2 – Функціональний компонент

Компонент, який ґрунтується на класі, створюється шляхом розширення React.Component. Стан компонента, заснованого на класі, змінюється через метод setState() та читається через this.state всередині класу. Використання методу setState() спричиняє оновлення відображення компонента, що дозволяє користувачеві бачити актуальну інформацію на сторінці. Приклад класового компоненту представлений на рис. 2.3.

```
class Greeting extends React.Component {
  render() {
      return <h1>Hello, {this.props.name}</h1>
  }
}
```

Рисунок 2.3 – Класовий компонент

Методи життєвого циклу представляють собою внутрішні функції, які автоматично викликаються при оновленні стану або властивостей компонента перед або після його відображення, а також при зникненні компонента зі сторінки. Огляд методів життєвого циклу компонента на основі класів показано на рисунку 2.4.

Метод componentDidMount() виконується після того, як компонент вперше з’явився на сторінці. Зазвичай в ньому міститься логіка створення таймерів, підписок на інші функції, а також завантаження початкових даних із серверу.

ComponentDidUpdate() запускається при кожному оновленні

компонента. Тут описується логіка оновлення даних, які відображаються в компоненті.

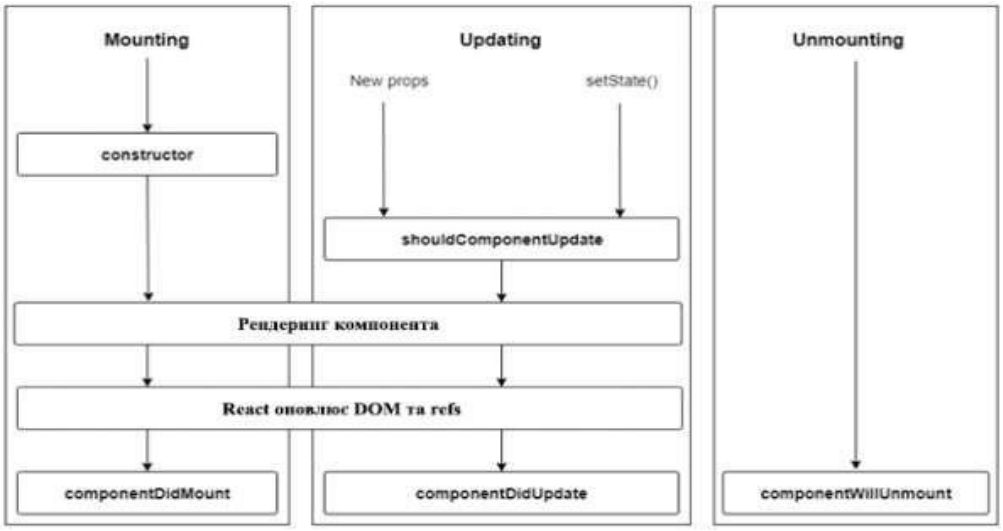

Рисунок 2.4 – Схема компонентів та їх циклу життя

ComponentWillUnmount() виконується після того, як компонент був видалений зі сторінки. В ньому здійснюються дії, протилежні тим, що були виконані в componentDidMount(): скасування підписок, видалення таймерів та інше.

ShouldComponentUpdate() приймає значення наступних властивостей та стану і може повертати true або false в залежності від цих значень. Це впливає на те, чи буде компонент перемальовуватися або залишатися незмінним.

Методи життєвого циклу допомагають управляти відображенням компонента на різних етапах та спрощують обробку даних, необхідних для визначення контенту, що буде показаний на вебсторінці.

React Hooks використовуються для забезпечення функціональним компонентам можливості користуватися станом і методами життєвого циклу, подібно до класових компонентів. Використання функціональних компонентів з React Hooks може спричинити зменшення розміру та складності програми React порівняно з компонентами на основі класів.

React Hooks призначені для більш складних функціональних

компонентів, вони зв’язуються з функціями React. Імена React Hooks повинні починатися зі слова “use”.

Стан не існує в функціональних компонентах за замовчуванням; замість цього можна використовувати хук React під назвою useState. Він зберігає стан протягом життєвого циклу компонентів. Хук useState та всі інші можна використовувати більше одного разу в одному компоненті. Функціональний компонент, що використовує хук useState, представлений на рисунку 2.5.

```
const Counter = () => {
  const [count,setCount] = useState(0);
  return (
    <div>
      <p>{count}</p>
      <button onClick={()=>setCount(count+1)}>
        Increment
      </button>
    </div>
  )
}
```

Рисунок 2.5 – Функціональний компонент з useState

Вбудовані методи життєвого циклу відсутні для функціональних компонентів, але можна створити їх за допомогою хука useEffect. За замовчуванням useEffect відтворює роботу методу під час кожного повторного рендерингу компонента, але його можна налаштувати так, щоб він виконувався лише при певних змінах. Приклад хука useEffect представлений на рисунку 2.6.

```
const Counter = () => {
  const [count,setCount] = useState(0);
  useEffect(() => {
    docoument.title = 'Count: ${count}'
  })
  ...
}
```

Рисунок 2.6 – Хук useEffect

У React для обміну даними між компонентами використовується вбудований інструментарій Context API. Це дозволяє спільно використовувати контекст між кількома компонентами через метод createContext у бібліотеці

React. Для додавання даних до контексту та їх доступу у дочірніх компонентах використовується провайдер, що доступний у випадку екземпляра Context .

Провайдер це самостійний компонент, який приймає одне значення властивості, де передаються дані для контексту. Ці дані можуть бути змінними, функціями або об’єктами. Усередині компонента Provider можна отримати доступ до даних контексту з будь-якого дочірнього компонента, використовуючи Consumer, доступний у випадку екземпляра Context. Додаток може мати кілька екземплярів контексту для різних цілей.

Передача функцій зворотного виклику дочірнім компонентам через Context API дозволяє змінювати стан батьківського компонента. Згідно з офіційною документацією, Context використовується для обміну даними, що вважаються глобальними для дерева компонентів React. Однак використання контексту може зменшити можливість повторного використання компонентів, оскільки вони звертаються до даних, отриманих через контекст від іншого компонента.

Для функціональних компонентів, які обгорнуті постачальником контексту, метод useContext виступає як провайдер, що дозволяє доступ до даних. Приклад використання хука useContext представлений на рис. 2.7.

Поміж компонентами можна створювати свої власні хуки, що дає можливість повторно використовувати функціональні можливості, окрім вбудованих хуків. Ще однією перевагою React є використання Virtual DOM. Це дозволяє оптимізувати відображення змін у DOM, що пришвидшує роботу додатку.

```
const PersonsContext = React.createContext();
<PersonsContext.Provider value={{persons: persons}}>
    <Greetings />
</PersonsContext.Provider>

const Greetings = () => {
  const { persons } = useContext(PersonsContext);
  return persons.map(person =>
    <h1>Hello, {person.name}</h1>
  )
}
```

Рисунок 2.7 – Хук useContext

Віртуальний DOM – це спрощена модель реального DOM-дерева вебдодатку. При внесенні змін до певного компонента у додатку, React порівнює реальний DOM з віртуальним, і перерисовує на сторінці лише ті компоненти, в яких сталися зміни. React обчислює мінімальний набір змін DOM, необхідних для того, щоб реальний DOM відповідав віртуальному.

JavaScript використовується для маніпуляцій з DOM, що обробляються пакетно для оптимізації часу. Розробники використовують JSX для визначення відображення компонентів у різних станах, а React відповідає за маніпулювання DOM для зміни станів.

Бібліотека React включає компілятор, який перетворює декларативний синтаксис JSX у JavaScript для інтерпретації браузером. На рисунку 2.8 показаний код JSX перед компіляцією, а на рисунку 2.9 – код, який виконується браузером. Віртуальний DOM може використовуватися для відтворення на сервері, оскільки він не залежить від браузера.

```
<div>
  <Greeting name="John" />
</div>
```

Рисунок 2.8 – Код JSX перед компіляцією

```
React.createElement("div",null,
  React.createElement(Greeting, {name: "John"})
)
```

Рисунок 2.9 – Код виконаний у веббраузері

React має інструмент під назвою create-react-app, що налаштовує робоче середовище для React використовуючи Node. Він забезпечує корисні інструменти, такі як перезавантаження, налагодження, менеджер пакетів, компілятор та інші.

Для використання create-react-app потрібно встановити та налаштувати Node і менеджер пакетів. Налаштування здійснюється за допомогою команди npm init react-app <application-name> з використанням версії NPM 6 або вище.

JavaScript V8 не може обробляти синтаксис, такий як JSX, тому потрібен компілятор, який перетворює JSX на звичайний JavaScript. У create-react-app використовується компілятор Babel. Babel-core містить необхідність для його роботи і функціональність presets, які використовуються для компіляції різних видів JavaScript. Наприклад, preset-env використовується для компіляції сучасного ECMAscript, а preset-react – для компіляції коду JSX.

create-react-app включає пакет Webpack, який об'єднує та мінімізує всі активи програми, такі як модулі вузлів і необхідні JavaScript файли. Налаштування Webpack здійснюється через файл webpack.config.js, де вказується точка входу файлів та місце виведення з'єднаних файлів.

Для додаткової функціональності можна використовувати плагіни Webpack. Один з таких плагінів, Hot Module Replacement, використовується для перебудови модулів під час змін в вихідному коді.

У файлі конфігурації package.json присутні сценарії, які виконують різні завдання, такі як розгортання локального сервера для розробки, збірка програми для виробництва та запуск тестів.

Структура створеного додатку на React включає директорії: node_modules, public (зі статичними файлами) та src (з вихідним кодом, включаючи файли CSS і JavaScript). Клієнтський інтерфейс проєкту базується на React і зазвичай запускається в середовищі Node.js [5].

2.3 Засоби для створення серверної частини системи

Розробка серверної частини вебзастосунку використовує середовище виконання Node.js, побудоване на основі движка JavaScript V8. Це середовище адаптує можливості та синтаксис JavaScript для створення backend-частини застосунку. Поєднання React.js із Node.js дозволяє створювати ізоморфні програми, забезпечуючи швидкий та ефективний зв'язок між клієнтською та серверною частинами вебзастосунку.

Node.js вважається ідеальним партнером React.js для backend-розробки через чудову сумісність функцій. Використання однієї мови JavaScript як для інтерфейсних, так і для серверних бібліотек дозволяє створювати сучасні додатки. Основні особливості використання Node.js включають масштабованість, швидкість, гнучкість та крос-платформність.

Комбінація React та Node.js надає кілька переваг у розробці вебзастосунків:

- можливість створення універсальних програм з кодом JavaScript як на клієнті, так і на сервері;
- оптимізація для SEO за допомогою відтворення переглядів додатків на сервері;
- використання єдиного механізму JavaScript V8 для візуалізації на обох сторонах – клієнті та сервері;
- використання спеціально розроблених для роботи з Node.js компонентів React DOM;
- використання npm пакетів для прискорення циклу розробки вебдодатків;
- об’єднання програми React у один файл за допомогою модулів Node.js.

Щодо зберігання та обміну даними між клієнтом і сервером, для отримання та оновлення інформації з бази даних на сервері використовується API через TP-запити. Існують різні типи API, серед яких найпоширеніші – RESTful та GraphQL API.

2.3.1 GraphQL API

Відсутність конкретних специфікацій для REST API спричинила різноманітні підходи до його створення. Основна структура API REST

залишається однаковою: при використанні цього API метод у HTTP-запиті описує дію з даними, а URL-адреса визначає, які саме дані підлягають операції. URL вказує на кінцеву точку API, де відбуваються обчислення, отримання даних з бази або виклики інших API. Результатом обробки цих маршрутів зазвичай є набір даних у форматі JSON.

GraphQL, представлений Facebook у 2015 році, став альтернативою REST API. Він пропонує мову запитів для створення та використання API GraphQL, де клієнт може запитувати конкретні дані, які йому потрібні, у відміну від REST API, де кожна кінцева точка повертає фіксований набір даних. API GraphQL пропонує лише одну точку входу для отримання всієї потрібної інформації. Це дозволяє створювати нові запити без необхідності змінювати або створювати нові кінцеві точки на сервері. Зв’язок між клієнтом і API здійснюється за допомогою HTTP-запитів, як і у REST API.

При використанні стороннього API розробник не має контролю над структурою кінцевих точок на сервері. У REST API додатку доводиться здійснювати багато запитів туди й назад для отримання всієї необхідної інформації, що призводить до неефективності між API та програмою. Крім того, програма повинна знати відповіді API для різних кінцевих точок, щоб правильно їх обробляти. GraphQL вирішує ці проблеми, надаючи гнучке API, де клієнт має більше контролю над даними.

Опис операцій, доступних у API, називається схемою – це план типів та їх зв’язків у API. Схема використовує мову визначення схем GraphQL (SDL) і є строго типізованою. Перед тим як виконується запит до API GraphQL, він перевіряється відповідно до схеми. Основним елементом API GraphQL є тип, що описує об’єкт, який може бути отриманий через API. Також існують вбудовані скалярні типи: ID, String, Int, Float і Boolean.

Існує окремий вид – тип запиту, що називається “Query”, який уточнює доступні запити в API та їхні результати. Приклад запиту представлений на рисунку 2.10.

```
type person {
  name: String
  age: Int
}
type Query {
  persons: [person]
}
```

Рисунок 2.10 – Опис запита Query

Квадратні дужки показують повернення масиву певного типу. Існує також тип Mutation, який визначає доступні мутації (оновлення даних) у API. Кожен запит і мутація мають свій власний resolver, де зазначена логіка обчислень, взаємодія з базою даних чи виклик інших API. Resolver повертає дані згідно зі схемою.

Простий приклад запиту – це запит API для отримання поля всередині об’єкта, показаного на рисунку 2.11. Запит визначається ключовим словом “query”, а перша пара фігурних дужок є кореневим об’єктом. У вказаному прикладі для кореневого об’єкта обираються поля “person”, а для кожного об’єкта person вибираються поля «ім’я» та «вік».

```
query GetPersonsQuery{
  persons {
    name,
    age
  }
}
```

Рисунок 2.11 – Запит Query

Виконання запиту на рисунку 2.12.

```
{
  "data": {
    "persons": [{
      "name": "John",
      "age": "32"
    }]
  }
}
```

Рисунок 2.12 – Результат запиту

Для актуалізації даних застосовується мутація. Змінні в мутації використовуються для передачі API інформації про об’єкти та поля, які

необхідно змінити.

На рисунку 2.13 показано приклад простої мутації.

```
mutation UpdatePersonMutation($name:String, $age:Int){
  updatePerson(name: $name, age: $age) {
    name,
    age
  }
}
```

Рисунок 2.13 – Приклад простої мутації

Мутація повертає корисні дані для оновлення компоненту клієнтського інтерфейсу при зміні стану після оновлення на сторінці. Результат виконання цієї мутації представлений на рисунку 2.14.

```
{
  "data": {
    "updatePerson": {
      "name": "John",
      "age": "33"
    }
  }
}
```

Рисунок 2.14 –Виконання простої мутації

Поміж запитами та мутаціями є третій вид операції, відомий як підписка. Ця операція аналогічна запитам, але, замість того, щоб активно запитувати дані, вона слідкує за їх змінами через API та отримує сповіщення про зміни. Для ефективного та швидкого управління інформацією у розробленому додатку застосовується Apollo Client [6].

2.3.2 Apollo client

Apollo Client – це рішення для управління даними, яке спрощує оновлення та збереження інформації в вебзастосунках, а також отримання даних із API GraphQL.

Використання Apollo Client сприяє створенню архітектури з одним

напрямком потоку даних, схожої на Flux. В цьому випадку основним джерелом даних програми є кеш клієнта Apollo, який зберігає стан програми і всі результати запитів.

Кеш Apollo Client зберігає результати запитів, уникаючи надсилання декількох мережевих запитів до API. Якщо потрібно отримати дані з сервера знову, політику отримання можна змінити на «тільки для мережі», щоб обійти кеш.

Кеш Apollo Client нормалізує отримані дані, тобто якщо запит витягує список користувачів, кожен користувач кешується окремо, і наступні запити для одного користувача отримуються з нормалізованого кешу.

Крім виконання запитів до віддалених даних з API, можна також виконувати та зберігати локальні дані. Ці локальні дані зберігаються в кеші поряд з віддаленими даними і отримуються за допомогою додаткового декоратора @client, розміщеного після запиту.

Доступ до локальних типів даних та запитів вказується за допомогою схеми, схожої на API GraphQL на сервері, описану раніше в цьому розділі. Оновлення локальних даних в кеші відбувається за допомогою клієнтських розпізнавачів, так само, як і для віддалених даних [6]. Односпрямований потік з використанням клієнта Apollo представлений на рисунку 2.15.

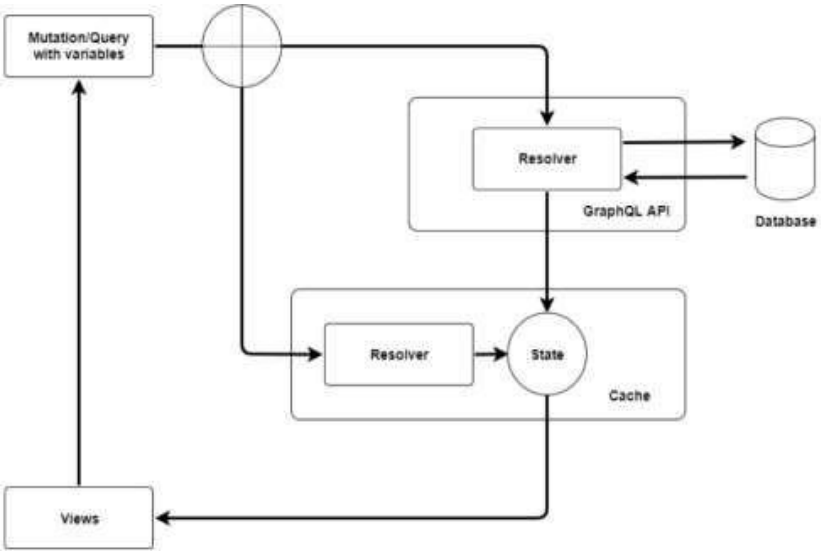

Рисунок 2.15 – Схематичне використання утіліти Apollo client

2.3.3 Клієнт Apollo в React

Бібліотека apollo-react використовується для надання функціональних можливостей клієнта Apollo для React Views. Щоб ініціалізувати клієнта Apollo, використовується кореневий компонент програми. Є чотири варіанти отримання доступу до клієнта з будь-якого дочірнього компонента. Можна скористатися наступними опціями.

Використання ApolloProvider для обгортання кореневого компонента. Компонент ApolloProvider надає можливість встановити сутність клієнта Apollo як властивість і доступатися до неї з будь-якого дочірнього компонента за допомогою методу withApollo, який знаходиться у пакеті react-apollo. Такий підхід представлений на рисунку 2.16.

```
<ApolloProvider client={client}>
  <Greetings />
</ApolloProvider>

const Greetings = ({ client }) => {
  const { loading, error, data } = client.query(GET_PERSONS)
  if (loading) return "Loading..."
  if (error) return error.message
  return data.persons.map(person =>
    <h1>Hello, {person.name}</h1>
  )
}
export default withApollo(Greetings);
```

Рисунок 2.16 – Компоненти обгортання ApolloProvider

Використання хуків React для звертання до Apollo Client у функціональних компонентах відбувається за схожим принципом. Для підключення ApolloProvider використовується додаткова бібліотека з ім’ям @apollo/react-hooks.

Клієнт Apollo стає доступним у дочірніх компонентах за допомогою хука useApolloClient. Для виконання запитів та мутацій можна користуватися хуками useQuery та useMutation, що проілюстровано на рисунку 2.17.

Використання ApolloConsumer компонента з будь-якого компонента, доступного в бібліотеці apollo-react, можна реалізувати шляхом створення функції render. Ця функція приймає єдиний аргумент – клієнт Apollo. Приклад компонента з використанням ApolloConsumer представлений на рисунку 2.18.

```
<ApolloProvider client={client}>
  <Greetings />
</ApolloProvider>

const Greetings = () => {
  const { loading, error, data } = useQuery(GET_PERSONS)
  if (loading) return "Loading..."
  if (error) return error.message
  return data.persons.map(person =>
    <h1>Hello, {person.name}</h1>
  )
}
export default Greetings;
```

Рисунок 2.17 – useQuery у ApolloProvider

```
const Greetings = () => (
  <ApolloConsumer>
    {client => {
    const { loading, error, data } = client.query(GET_PERSONS)
     if (loading) return "Loading..."
     if (error) return error.message
     return data.persons.map(person =>
        <h1>Hello, {person.name}</h1>
     )
    }}
  </ApolloConsumer>
);
```

Рисунок 2.18 – Компонент з використанням ApolloConsumer

Використання технологій Apollo Client дозволяє ефективно та швидко обробляти дані у React-додатку, а використання Node.js для створення серверної частини відкриває широкі можливості для створення оптимізованого та універсального вебзастосунку. Окрім цього, важливою складовою проєкту є база даних, яка зберігає значну кількість інформації про клієнтів, IT-спеціалістів, замовлення та інше. У цьому проєкті використовується реляційна база даних та мова запитів SQL для ефективного управління необхідною інформацією.

2.4 Методи технологічного впровадження при створенні бази даних

Електронні таблиці призначені для опрацювання числових даних, тоді як бази даних використовуються для обробки інформації, зокрема, структурованої інформації. Розроблення баз даних може відповідати різним потребам, таким як:

– відстеження, організація та редагування даних;

– збір даних і створення необхідних звітів;

– створення сховища для складних та динамічних вебзастосунків.

На сьогоднішній день найпоширенішою технологією баз даних є реляційні бази даних, які зберігають дані у нормалізованому вигляді, розбиваючи їх на різні таблиці. Хоча реляційні бази даних існують давно, вони залишаються універсальним інструментом для зберігання та управління даними.

Початок сучасних технологій баз даних відзначається на початку 1970-х років з появою першої «реляційної моделі даних». Сьогодні реляційні бази даних є ключовим елементом у створенні вебсайтів, а будь-який додаток, що використовує дані з бази, повинен включати сценарії отримання даних на сервері, зовнішній вигляд для відображення даних за допомогою HTML і CSS, підтримку мови SQL для виконання запитів та систему управління базами даних (СУБД).

Реляційні бази даних складаються з таблиць, що містять пов’язану інформацію. Системи управління реляційними базами даних (RDBMS) дозволяють розробникам створювати та підтримувати програми баз даних, включаючи інструменти для створення та редагування даних, проєктування структури бази даних, збору інформації і генерації звітів. RDBMS також містять вбудовану мову програмування, таку як SQL, для автоматизації функцій, таких як отримання даних, оновлення рядків та інші операції.

Структура реляційної бази даних включає таблиці, стовпці, індекси та обмеження. Вони дозволяють зберігати дані, оптимізувати запити, а також застосовувати обмеження для контролю коректності даних. Розробка бази даних є ключовим етапом у створенні вебзастосунків, де важливо зберігати та обробляти інформацію про клієнтів, організації та замовлення [7].

3 РЕАЛІЗАЙЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ ЗАМОВЛЕННЯМИ

3.1 Проєктування та подальша розробка бази даних

Основна інформація, необхідна для ефективної роботи вебзастосунку, знаходиться в базі даних. Для створення цієї бази даних потрібно визначити ключові сутності та їх атрибути, які визначають дані, що зберігаються. У розробленому вебзастосунку основні сутності включають:

- послуга (service) – має в собі інформацію про послугу в інформаційних технологіях, яку бажає отримати замовник;
- спеціаліст (expert) – містить дані про спеціаліста або ІТорганізацію, яка може виконати поставлене завдання;
- замовник (client) – представляє працівника чи організацію, які розміщують запит на виконання поставленої задачі;
- замовлення (order) – розміщений запит від замовника на певну послугу в сфері ІТ.

Кожна сутність репрезентована окремою таблицею в базі даних, де стовпці таблиці відповідають атрибутам кожної сутності. Дані для таблиці вводяться у новий рядок, де кожна клітинка представляє значення конкретного атрибуту для кожного окремого об’єкта.

Схема описаних сутностей та їх атрибутів представлена на рисунку 3.1.

Характеристики ІТ-послуг включають в себе унікальний ідентифікатор для кожної послуги, її назву, короткий опис робочого процесу та технології, які можуть використовуватися для виконання цієї послуги. Щодо ІТ-спеціаліста, його атрибути включають унікальний ідентифікатор, назву (яка може включати назву організації або ПІБ окремого фахівця), досвід (вказує на тривалість робочого стажу в місяцях або роках), рейтинг (розраховується на основі відгуків на розробленому вебзастосунку) та опис основної діяльності.

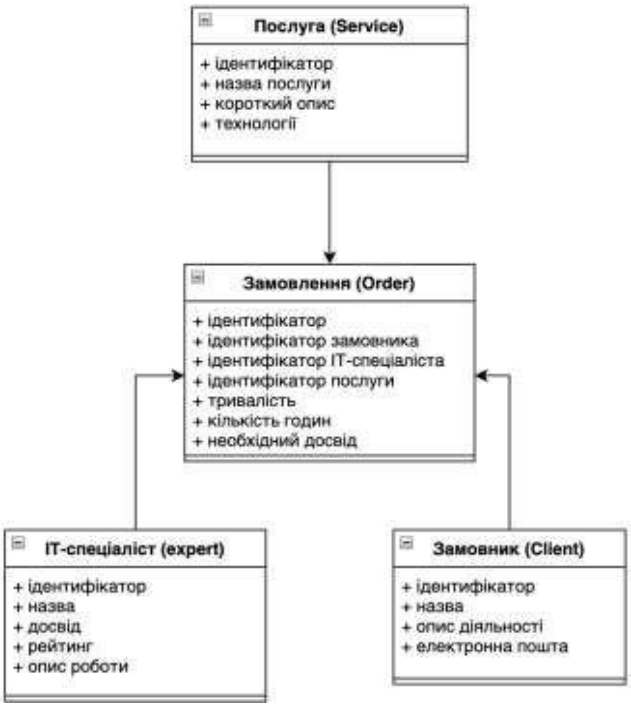

Рисунок 3.1 – Схематичне відображення сутностей бази даних вебзастосунку

Атрибути замовника включають унікальний ідентифікатор, назву (аналогічну назві сутності IT-спеціаліста), опис основної діяльності замовника та електронну пошту для можливості додаткових контактів.

Сутність замовлення пов’язана з усіма зазначеними об’єктами через унікальні ідентифікатори. Вона містить тривалість часу для виконання замовлення, кількість годин, які розробник може витратити щотижня, та необхідний досвід спеціаліста, відповідального за виконання конкретного замовлення.

Кожен атрибут має свій тип даних та додаткові властивості, які вказують, чи може даний атрибут мати нульове (null) значення або чи може він бути пов’язаний з іншою таблицею.

Створення нових таблиць у базі даних виконується за допомогою мутацій, зокрема, на рисунку 3.2 представлена мутація для створення таблиці замовників.

При створенні кожної таблиці фіксується її назва та потенційні поля з урахуванням їх особливостей.

В описі мутації вказані два основних методи: метод “up” вносить описані зміни в базу даних, тоді як метод “down” відміняє внесені зміни. Такий підхід виявляється важливим при розробці, оскільки дозволяє швидко відмінити останні кроки у випадку виявлення помилок при створенні таблиці.

Рисунок 3.2 – Мутація у таблиці «client»

Файли для роботи з базою даних розташовані в окремих модулях та складаються з файлів “resolvers”, які включають файли “useCases”, виконуючі відповідні дії.

У файлах “useCases” описана логіка обробки даних для виконання запиту та виклик основних методів роботи з даними з файлу “repository”.

Приклад файлу “repository” представлений у додатку А.

При виклику команди “npm run graphql:generate” створюються згенеровані файли, які дозволяють використовувати описані запити для роботи з базою даних у компонентах користувацького інтерфейсу.

Це розділення дозволяє відокремити логіку роботи з базою та інтерфейсом у відділені директорії, спрощуючи подальший процес розробки та підтримки проєкту [2].

3.2 Розробка інтерфейсу користувача

У розробленій вебсистемі інтерфейс користувача включає наступні основні сторінки:

– головна сторінка вебсайту;

– сторінка з формою реєстрації;

– сторінка з формою авторизації;

– сторінка з переліком всіх існуючих замовлень;

– сторінка з інформацією про зареєстрованих IT-спеціалістів.

Елементи цих сторінок розроблені за допомогою React компонентів, які повертають JSX-синтаксис. Цей синтаксис складається з HTML-тегів та JavaScript-логіки. Головна сторінка вебсистеми зображена на рисунку 3.3.

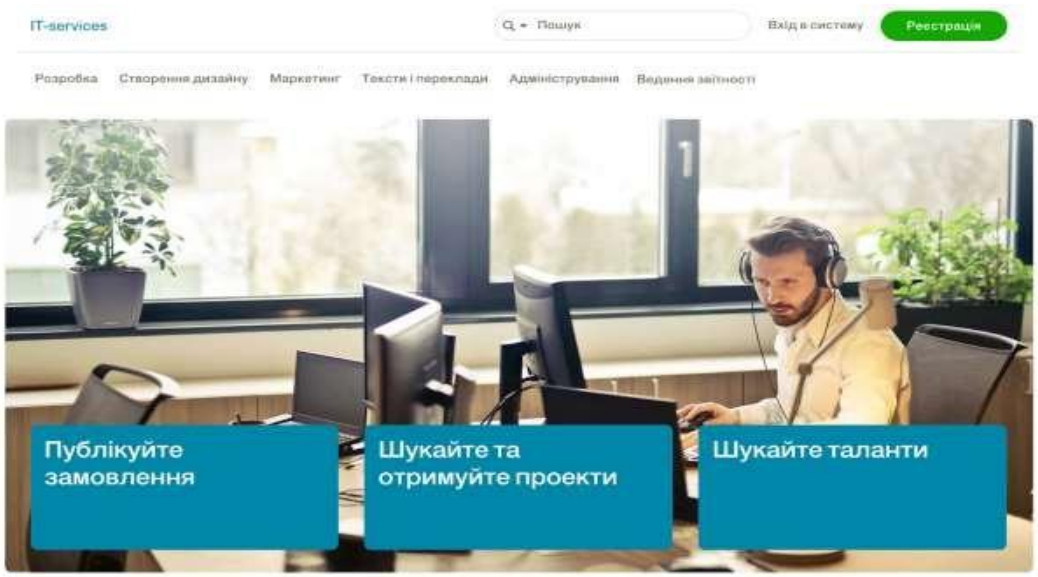

Рисунок 3.3 – Відображення головної сторінки вебсистеми

На головній сторінці розташовано меню, яке включає основні категорії послуг у сфері інформаційних технологій, а також кнопки для реєстрації та авторизації на вебсайті. Процес реєстрації у розробленому вебзастосунку складається з двох етапів: перевірка електронної пошти та налаштування інших обов’язкових даних. Перший етап реєстрації відображений на зображенні 3.4.

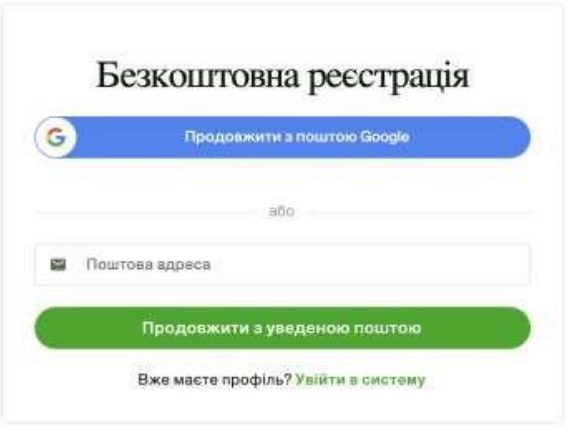

Рисунок 3.4 – Відображення першого етапу реєстрації користувача

На початковому етапі користувач може обрати реєстрацію через Google пошту або ввести альтернативну електронну адресу. У випадку, якщо користувач вже має обліковий запис на сайті, він може перейти до сторінки авторизації. Другий етап реєстрації ілюстрований на рисунку 3.5.

На наступному етапі користувач повинен внести такі дані: ім’я, прізвище, назву організації (необов’язково), короткий опис діяльності та обрати свою роль на сайті – замовника або IT-спеціаліста. Крім того, користувач має створити пароль для можливості подальшого входу на сайт.

Для авторизації на сайті необхідно ввести ім’я та прізвище (username), які були вказані під час реєстрації, або адресу електронної пошти. Якщо реєстрацію виконано за допомогою пошти Google, можлива також авторизація через цей обліковий запис. Форма для авторизації представлена на рис. 3.6.

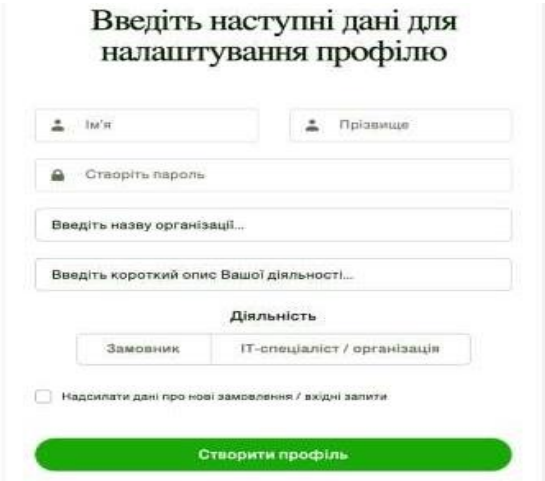

Рисунок 3.5 – Відображення другого етапу реєстрації користувача

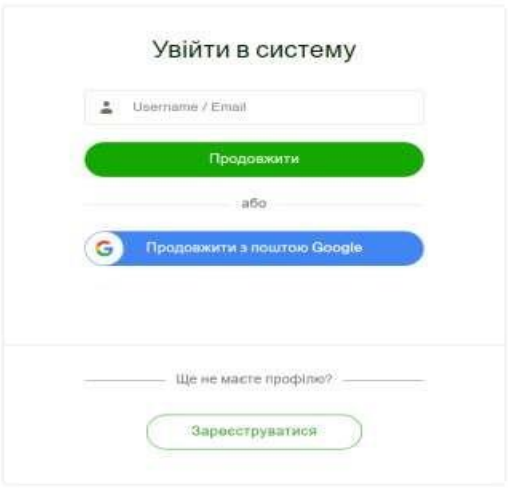

Рисунок 3.6 – Форма для авторизації користувачів

Після входу в систему користувач отримує можливість переглядати всі оформлені замовлення у вебзастосунку. Замовлення можна переглядати відповідно до визначеної категорії. На рисунку 3.7. показано приклад перегляду наявних замовлень у сфері розробки.

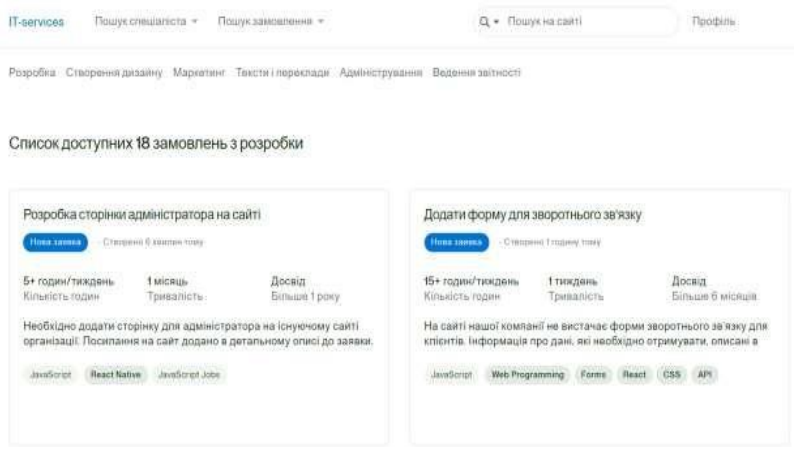

Рисунок 3.7 – Сторінка із списком замовлень які доступні

У блоку замовлення міститься ключова інформація, яка сприятиме визначенню розробником можливості виконання завдання. Замовлення включають назву, статус та час, що вказують на актуальність виконання, кількість годин та загальний час виконання роботи (приблизний, оскільки визначається замовником, який може помилитися з оцінкою), короткий опис із зазначенням необхідного функціоналу та технологій.

При натисканні на конкретне замовлення IT-спеціаліст або організація може вирішити, надсилати запит на виконання чи ні. При висиланні запиту IT-спеціаліст повинен надати дозвіл на обробку особистих даних, введених при реєстрації, і вказати коментар до виконання, де може додатково розповісти про себе або задати питання.

Після висилання запиту замовлення приховується та переходить в статус очікування відповіді. Замовник розглядає отриманий запит і вирішує, чи відкривати особисті контакти для подальшої комунікації. У випадку відмови може надати причину, і замовлення знову відображатиметься в списку актуальних завдань.

При підтвердженні запиту замовник розкриває свої контакти та налаштовує комунікацію із IT-спеціалістом. Після виконання завдання замовник може залишити відгук та встановити оцінку, яка впливатиме на рейтинг виконавця. Рейтинг та інша інформація про співробітників або зареєстровані IT-організації представлений на рисунку 3.8.

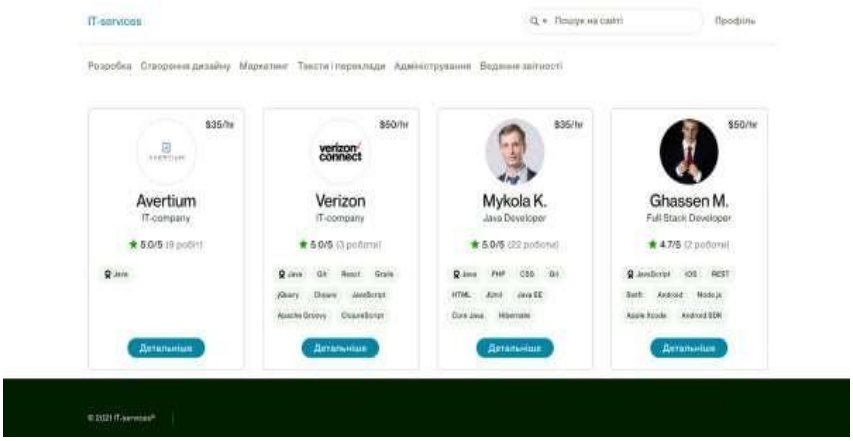

Рисунок 3.8 – Сторінка із списком організацій та спеціалістів

Для кожного спеціаліста вказана його посада, робочий досвід, кількість виконаних замовлень через розроблений вебзастосунок, а також основні технології, з якими він працює. Розроблений інтерфейс, що складається з описаних сторінок, надає користувачам можливість публікувати та виконувати різноманітні завдання в галузі інформаційних технологій. Для

належного функціонування та виконання всіх вказаних завдань в даній роботі використано JavaScript та бібліотеку React, які забезпечують логіку виконання основних дій користувачів на сайті.

3.3 Розробка складових інформаційної системи

Оскільки інтерфейс користувача розроблений за допомогою React компонентів, найкращим підходом буде написання функціональної логіки на мові програмування JavaScript із використанням вбудованих можливостей бібліотеки React.

Логіка роботи вебзастосунку описується таким чином: дані отримуються та обробляються безпосередньо в компонентах, а потім передаються дочірнім елементам за допомогою об’єкта властивостей. У випадку класових компонентів початкові дані зберігаються в об’єкті стану.

Основні функції розробленого вебзастосунку включають:

- реєстрацію нових користувачів;
- авторизацію зареєстрованих користувачів;
- публікацію замовлень (необхідних робіт);
- надсилання запиту на виконання замовлення;
- отримання списку всіх замовлень відповідно до обраної послуги.

За реєстрацію нових користувачів відповідає компонент UserSignUp.

Основний код компоненту представлений на рисунку 3.9.

В цьому компоненті використовується користувацький хук useSignUpModule, який відповідає за обробку функції реєстрації та взаємодію з сервером. Компонент повертає форму для реєстрації. При її відправці викликається метод submitCallback, який обробляє дані, що ввів користувач на сайті, та здійснює реєстрацію.

У додатку Б наведено код мутації, яка виконується під час реєстрації нового користувача. Для кожного користувача генерується унікальний ключ

(токен), який потім зберігається в локальному сховищі (local storage) веббраузера користувача. Це дозволяє користувачу уникати повторної авторизації при оновленні сторінки або завершенні роботи системи. Код головного компоненту для авторизації користувача представлений на рисунку 3.10.

Рисунок 3.9 – Компонент UserSignUp

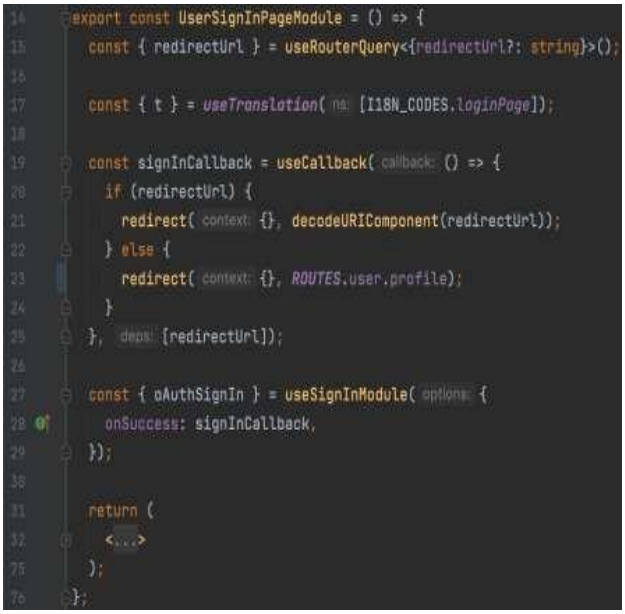

Рисунок 3.10 – Компонент UserSignUpPageModule

Для отримання функції авторизації використовується хук, аналогічний хуку реєстрації, який був розроблений – `useSignInModule`. Після успішної

авторизації користувач автоматично переходить на свою сторінку профілю для перегляду та перевірки введеної інформації. У випадку, якщо незареєстрований користувач намагається переглядати замовлення, розміщені на сайті, без авторизації, йому відкриється форма для введення облікових даних. При успішному введенні даних користувач автоматично перенаправляється на ту саму сторінку, яку він спробував переглянути. Код компонента, що відповідає за форму авторизації представлений на рисунку 3.11.

```
export const UserSignIn: FC<Props> = (props ) => {
  const { onSuccess, email } = props;

  const { baseSignIn } = useSignInModule( { onSuccess });

  const submitCallback = useCallback(
    async (formData: SignInFormData) => {
      await baseSignIn(formData);
    }, [baseSignIn],
  );

  return (
    <Form<SignInFormData>
      subscription={{ submitting: true, pristine: true }}
      onSubmit={submitCallback}
      initialValues={{
        email,
      }}
      render={(formRenderProps ) => (
        <BaseSignInForm {...formRenderProps} />
      )}
    />
  );
};
```

Рисунок 3.11 – Компонент UserSignIn

Для обробки надісланої форми використовуються передвизначені типи даних полів для авторизації (SignInFormData). Функція submitCallback описується і викликається при відправці цієї форми. При авторизації користувача проводяться перевірки його ролі (замовник або виконавець), валідація введених даних через запит до бази даних, а також створення та збереження унікального користувацького коду в веббраузері для спрощення подальшої взаємодії з вебзастосунком.

Після успішної авторизації замовнику стає доступний функціонал публікації робіт, що означає можливість розміщення запитів на отримання

конкретної послуги в галузі інформаційних технологій. За публікацію нових замовлень відповідає компонент CreateJob, представлений на рисунку 3.12.

Рисунок 3.12 – Компонент CreateJob

Після натискання на кнопку «додати замовлення» відбувається відкриття модального вікна для введення необхідної інформації. Інформація про відкриття чи закриття цього вікна зберігається в об’єкті стану функціонального компонента за допомогою хука useState().

При створенні форми для додавання замовлення використовуються два основні методи: getInitialValues() та handleSubmit(), які представлені на рисунку 3.13.

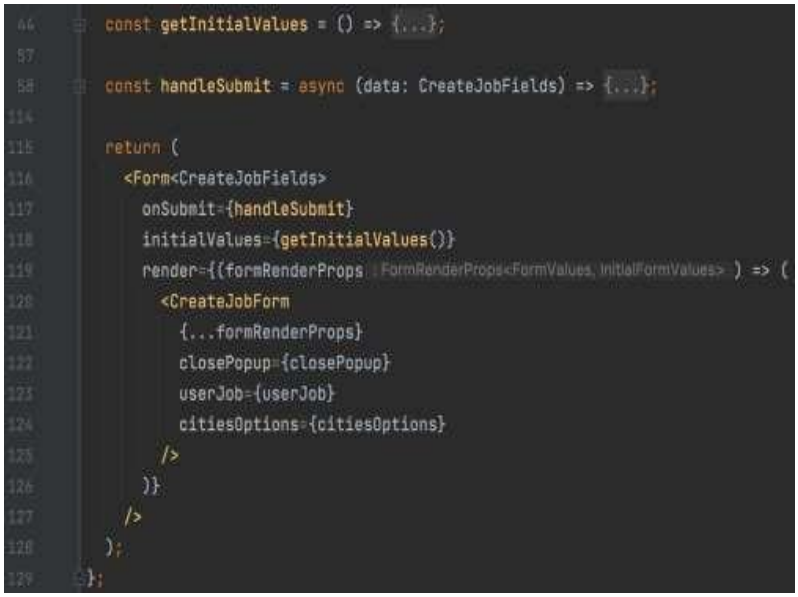

Рисунок 3.13 – Обробка форми створення замовлення

Функція getInitialValues() призначена для визначення початкових значень форми. При створенні нового замовлення ці значення встановлюються як кульові, а при редагуванні конкретного замовлення вони отримуються з бази даних та приймаються як початкові.

Функція handleSubmit() виконується при надсиланні форми з даними про замовлення. Усі введені дані обробляються відповідною мутацією, і в базі даних створюється новий запис в таблиці “orders”. При створенні нового замовлення в базі даних його статус автоматично встановлюється в значення 'new'.

Авторизовані користувачі можуть переглядати список всіх доступних замовлень на вебсайті. Цей функціонал реалізовано за допомогою компонента OrderList, представлений на рисунку 3.14.

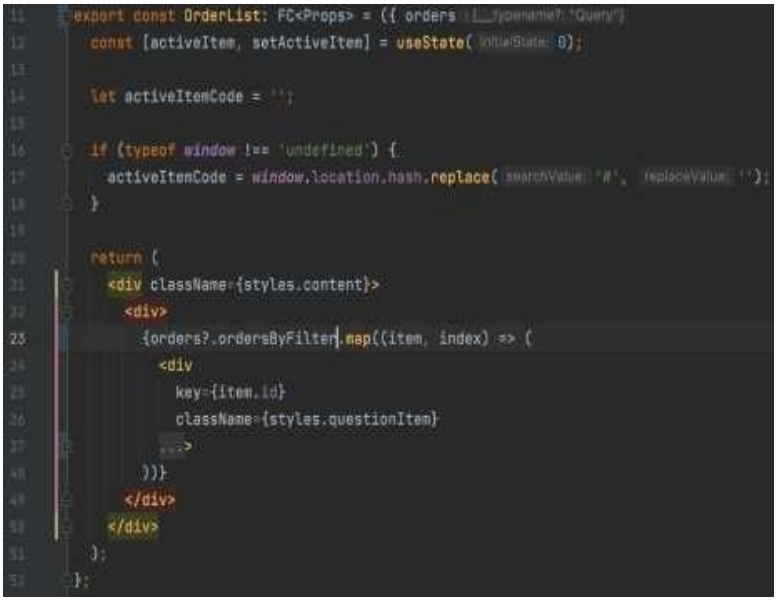

Рисунок 3.14 – Компонент OrderList

У даному компоненті властивості об’єкта містять масив усіх замовлень, який отримано через виконання запиту до бази даних. При використанні конкретного фільтра (пошуку замовлень лише за певними послугами в сфері інформаційних технологій) до запиту в базі даних додаються додаткові умови, що забезпечують відбір лише необхідних робіт.

При натисканні на певне замовлення воно стає активним і записується в

змінну “activeItemCode” для подальшого полегшення обробки обраної роботи. Для кожного замовлення створюється окремий компонент, який виводиться на вебсторінку користувачу з необхідною інформацією. З метою виконання конкретного замовлення IT-спеціалістам потрібно надіслати відповідний запит.

Функція по надсиланню запитів представлена на рисунку 3.15.

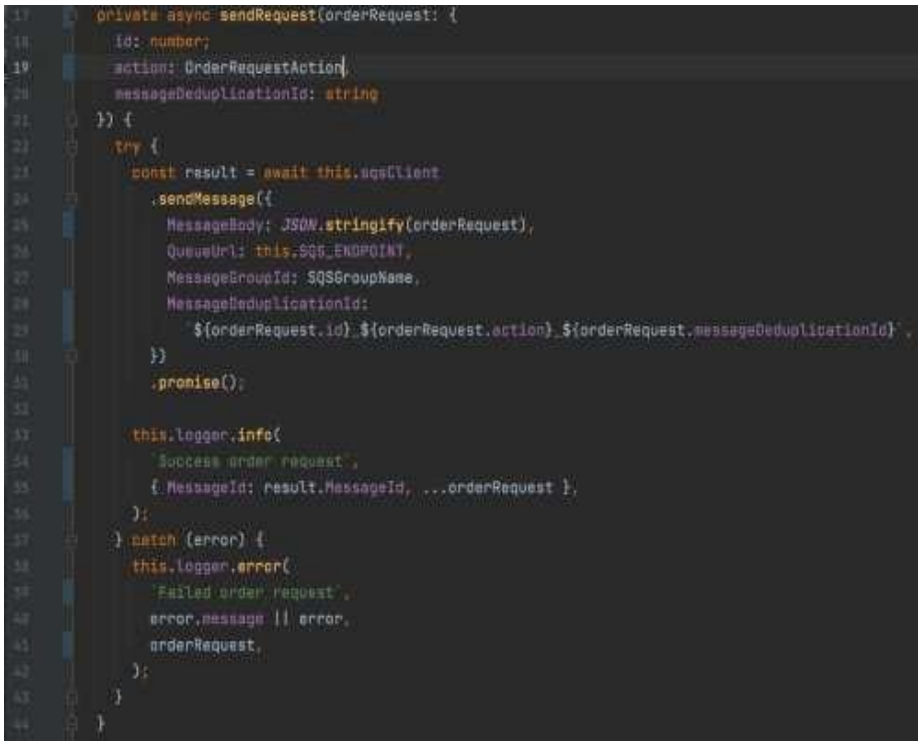

Рисунок 3.15 – Метод sendRequest

Під час виконання запиту на виконання конкретного замовлення відбувається відправлення повідомлення замовнику, яке містить інформацію про здійснену подію. У випадку будь-яких помилок під час надсилання повідомлення або у випадку успішного виконання, IT-спеціаліст отримає відповідне повідомлення на відповідній сторінці. Помилки під час надсилання запиту можуть виникнути, наприклад, якщо IT-спеціаліст має недостатній досвід для обраного замовлення або якщо була вказана неточна інформація під час реєстрації на сайті, така як створення копії певної IT-організації.

Коли замовник отримує таке повідомлення, він може переглянути

інформацію про ІТ-спеціаліста або організацію, яка відправила запит, та прийняти чи відхилити його. Замовник також може, за бажанням, вказати певний коментар і надати свої контактні дані для подальшої комунікації з виконавцем.

ВИСНОВКИ

В роботі реалізовано інформаційну системи, що використовує HTML, CSS та React разом з JavaScript. Ця система відповідає за взаємодію користувача з додатком у веббраузері та надає необхідну інформацію.

Основні сторінки цієї системи, розроблені за допомогою вказаних технологій, включають головну сторінку вебзастосунку, сторінки реєстрації та авторизації користувачів, відображення оформлених замовлень на сайті та IT-організацій, що відповідають за їх виконання. Усі ці елементи реалізовані за допомогою React компонентів.

Серверна частина, побудована на Node.js, відповідає за зберігання та обробку даних, які використовуються користувачами у вебдодатку.

У цьому вебзастосунку реалізовані мутації для реєстрації користувачів, запити на отримання та додавання нових замовлень у базу даних та інші функції. Основна логіка обробки даних описана у React компонентах за допомогою JavaScript. Ці функції включають обробку інформації з бази, фільтрацію замовлень відповідно до обраних послуг на сайті та повідомлення про виконання запиту на певне замовлення для користувачів.

Для зберігання даних в базі використано Apollo client в React, що дозволяє виконувати різні види запитів для отримання та оновлення інформації, надаючи користувачу актуальну інформацію.

У майбутньому можна розширити можливості вебдодатка шляхом:

- додавання можливості редагування інформації про замовлення;
- покращення комунікації між замовником та виконавцем всередині вебдодатка, без необхідності розкриття особистих контактних даних;
- створення адміністративної панелі для додавання нових видів послуг.

ПЕРЕЛІК ПОСИЛАНЬ

1. Marketing. Types of services. URL: https://pidru4niki.com/73961/marketing/vidi_informatsiynih_produktiv_poslug (дата звернення 02.08.2023).

2. Informatics. Data base. URL: https://stud.com.ua/informatika/ (дата звернення 07.08.2023).

3. HTML. URL: https://en.wikipedia.org/wiki/HTML (дата звернення 07.08.2023).

4. CSS. URL: https://ru.wikipedia.org/wiki/CSS (дата звернення 07.08.2023).

5. React. Components. URL: https://ru.legacy.reactjs.org/ (дата звернення 10.09.2023).

6. Apollo Client. GraphQL. URL: https://habr.com/ru/articles/745848/ (дата звернення 13.09.2023).

7. Ярцев В. П. Організація баз даних та знань: сукупність відомостей щодо понять баз даних. Київ, 2018. С. 10–89.

8. React. Props and components. URL: https://ru.legacy.reactjs.org/docs/components-and-props.html (дата звернення 02.10.2023).