МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

**на тему:** «Web-сервіс оцінки якості обслуговування наданих автомобілістам послуг»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

**Виконавець роботи:** студент групи ІТ-61 Гріцун Олександр Олександрович

**Кваліфікаційна робота бакалавра**
**захищена на засіданні ЕК**
**з оцінкою** _____ «__» _____2020 р.

Науковий керівник _____ к.т.н., доц., Ващенко С. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів відповідних посилань.

Студент _____
(підпис)

Суми-2020

**Сумський державний університет**
**Факультет** електроніки та інформаційних технологій
**Кафедра** комп'ютерних наук
**Секція** інформаційних технологій проектування
**Спеціальність** 122 «Комп'ютерні науки»
**Освітньо-професійна програма** «Інформаційні технології проектування»

**ЗАТВЕРДЖУЮ**
Зав. секцією ІТП
В. В. Шендрик
«___»_____2020 р.

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ**

*Гріцун Олександр Олекксандрович*

**1 Тема роботи** *Web-сервіс оцінки якості обслуговування наданих автомобілістам послуг*

**керівник роботи** Ващенко Світлана Михайлівна*, к.т.н., доцент*,

затверджені наказом по університету від « 17 » травня 2020 р. № 0834-III

**2 Строк подання студентом роботи** «3» червня 2020 р.

**3 Вхідні дані до роботи** _____

**4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібномрозробити)**
1) Аналіз предметної області
2) Моделювання та проектування додатку
3) Програмна реалізація додатку

**5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)**
Актуальність, постановка задачі, аналіз аналогів, функціональні вимоги, контекстна діаграма візуалізації процесу, діаграма використання, засоби реалізації, демонстрація додатку, висновки.

**6. Консультанти розділів роботи:**

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

**7.Дата видачі завдання**_____

**КАЛЕНДАРНИЙ ПЛАН**

| № п/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Створення моделі «Comment» | 13.01.20 | |
| 2. | Створення моделі «Service» | 17.01.20 | |
| 3. | Створення моделі «Post» | 23.01.20 | |
| 4. | Створення моделі «User» | 16.01.20 | |
| 5. | Дизайн головної сторінки | 27.01.20 | |
| 6. | Дизайн профілю користувача | 03.02.20 | |
| 7. | Дизайн сторінок створення | 10.02.20 | |
| 8. | Верстання сторінок згідно дизайну | 09.03.20 | |
| 9. | Написання тестів | 23.03.20 | |
| 10. | Налаштування оточення | 25.03.20 | |
| 11. | Виконання тестування | 08.04.20 | |
| 12. | Фікс дефектів | 17.04.20 | |

**Студент** _____ Гріцун О.О.

(підпис)

**Керівник роботи** _____ к.т.н., доц. Ващенко С.М.

(підпис)

# РЕФЕРАТ

Тема бакалаврської роботи: «Web-сервіс оцінки якості обслуговування наданих автомобілістам послуг».

Кваліфікаційна робота бакалавра присвячена розробці додатку для автомобілістів та полегшення пошуку автосервісів.

Перед початком розробки інформаційної системи було проведено огляд останніх досліджень і публікацій, аналіз програмних продуктів-аналогів та постановку задачі. Також було розроблено технічне завдання на виконання роботи та проведено планування робіт.

Результатом моделювання та проектування роботи додатку є контекстні діаграми процесів інформаційної системи у нотації IDEF0, діаграми варіантів використання системи. Результатом виконаної роботи є веб-додаток, що дозволяє користувачам шукати та оцінювати рівень наданих послуг від різних типів автосервісів.

Додаток було розроблено на мові Java Script, отже клієнтом для користування системою є браузер.

Практичне значення роботи полягає в тому, що додаток дозволяє моніторити найкращі автосервіси та розповідати власні історії обслуговування іншим користувачам.

Пояснювальна записка містить вступ, 3 розділи, висновки, додатки та список літератури з 25 найменувань. Загальний обсяг роботи – 66 сторінок: 37 сторінок основного тексту, 2 сторінки списку використаних джерел, 26 сторінок додатків.

Ключові слова: ВЕБ-ДОДАТОК, АВТОСЕРВІС, NODEJS, MONGODB, JAVA SCRIPT, REACTJS.

# ЗМІСТ

# ВСТУП

В даний час технології стрімко розвиваються кожну секунду нашого життя. В Україні кожного року показник кількості машин на 1000 чоловік росте. З 2016-го співвідношення машин до людей стало більше 20%, тобто на кожні 1000 чоловік 200 автівок (69-те місце по світу за цим показником).

Не менш важливо і те, що більшість машин в нашу країну завезено з європейський країн не новими, іноді битими, а тому питання ремонту та обслуговування автомобілей є актуальніше, ніж в Америці (де люди купують нові машини кожні 2-5 років та не переймаються за проблеми машин, які виникають з часом).

Так як запит на СТО та мийки збільшується, то й кількість цих сервісів збільшується, але не всі вони виконують свою роботу правильно та чесно. Щоб не натрапити на нечесних людей або тих, хто виконує свою роботу не недостаньому рівні, є потреба у веб-сервісі для відслідковування відгуків про СТО, мийки, дилерські центри та послуги «пригону» авто з інших країн.

Тому метою роботи є розробка web-сервісу оцінки якості обслуговування наданих автомобілістам послуг.

Для досягнення мети необхідно виконати такі задачі:

- проаналізувати предметну область та ресурси-аналоги;
- обрати засоби та методи реалізації поставленої задачі;
- провести моделювання додатку;
- створити базу даних, провести налаштування бази даних та додатку;
- виконати дизайн;
- створити додаток та провести інтеграцію бази даних;
- провести налаштування додатку та його тестування.

Даний проект орієнтований в першу чергу на всіх українських автомобілістів.

Практичне значення додатку – розроблено зручний ресурс для обміну досвідом користування автомобільним сервісами, знаходження найкращих сервісів та засобів комунікації з ними.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Огляд останніх досліджень і публікацій

Загальновідомий факт, що клієнти є запорукою процвітання. Немає клієнтів — немає грошей. 77,5 % компаній погоджуються, що саме рівень обслуговування клієнтів безпосередньо впливає на прибуток. Також, 82% компаній вірять, що високий рівень сервісу допомагає випереджати конкурентів. Тому, варто ставитись до всіх клієнтів, як до VIP. Такий клієнтоорієнтований підхід допоможе побудувати довгострокові та взаємовигідні відносити, які здатні зробити репутацію надійного бренду, якому довіряють. У публікації [18] найкращим способом оцінити сервіс – є відгуки клієнтів. Стаття [19] розповідає нам про найкращі способи мотивації клієнтів для створення відгуків. Найкращим є прямий запит про відгук.

Проте, у світі автосервісів існують дві проблеми: власники станцій не мають бажання діджиталізувати бізнес, друга – шахраї [20]. В обох випадках клієнти сервісу не можуть отримати необхідну інформацію про якість обслуговування та чесність майстрів.

Отже, гарний сервіс оцінки може допомогти власникам бізнесів збільшити кількість клієнтів, а власники автівок захищені від шахраїв.

Наразі існує декілька способів отримання інформації з відгуками про сервіси. Перший – книга відгуків, застаріла, іноді відсутня, крім того зазвичай її не використовують клієнти. Другий – різноманітні форуми. Головна незручність – складність пошуку та різноманітність тематик на таких сторінках. Третій – соц. мережі. Доволі гарний спосіб, адже можна спілкуватися с реальними клієнтами сервісів. Проте виникає проблема фіктивних відгуків, не всі сервіси мають свої сторінки в соц. мережах.

Як було зазначено вище, головним джерелом відгуків є форуми на різноманітні тематики з незручним, застарілим інтерфейсом. Проаналізувавши програмні продукти, можна сказати, що альтернатив веб-сервісу, що розробляється, не існує.

## 1.2 Огляд аналогів

Проведено аналіз предметної області створюваного проекту, для цього його було порівняно з іншими програмними продуктами, що мають схожу тематику, вихідний результат і мету.

Першим є Trip Advisor (рис.1.1), котрий дозволяє знайти найкращі заклади та місця для відпочинку туристам з усих країн світу. Користувачі мають змогу залишити відгук з фото та поставити оцінку закладу [21].

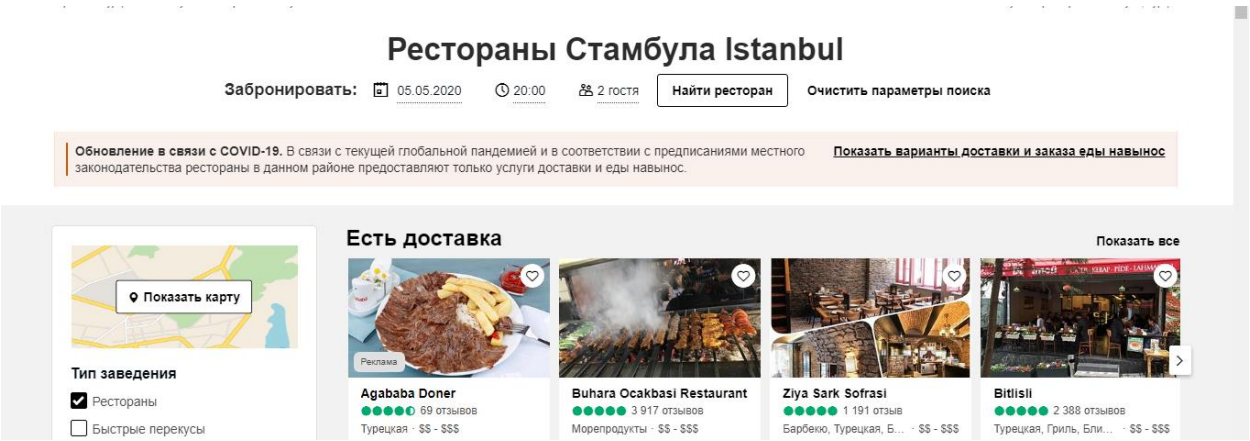


Рисунок 1.1 – Рсйрс «Trip Advisor»

Другий інструмент - це відгуки від Google [22]. Тут можна віднайти відгук про будь яке місце або послугу (рис.1.2). Проте, проблемою є нечесні відгуки фейкових акаунтів, в цілому відгуки не мають певною тематики,

тому інтерфейс користувача не є ідеальним, не вистачає спеціалізованої політики написання відгуків.

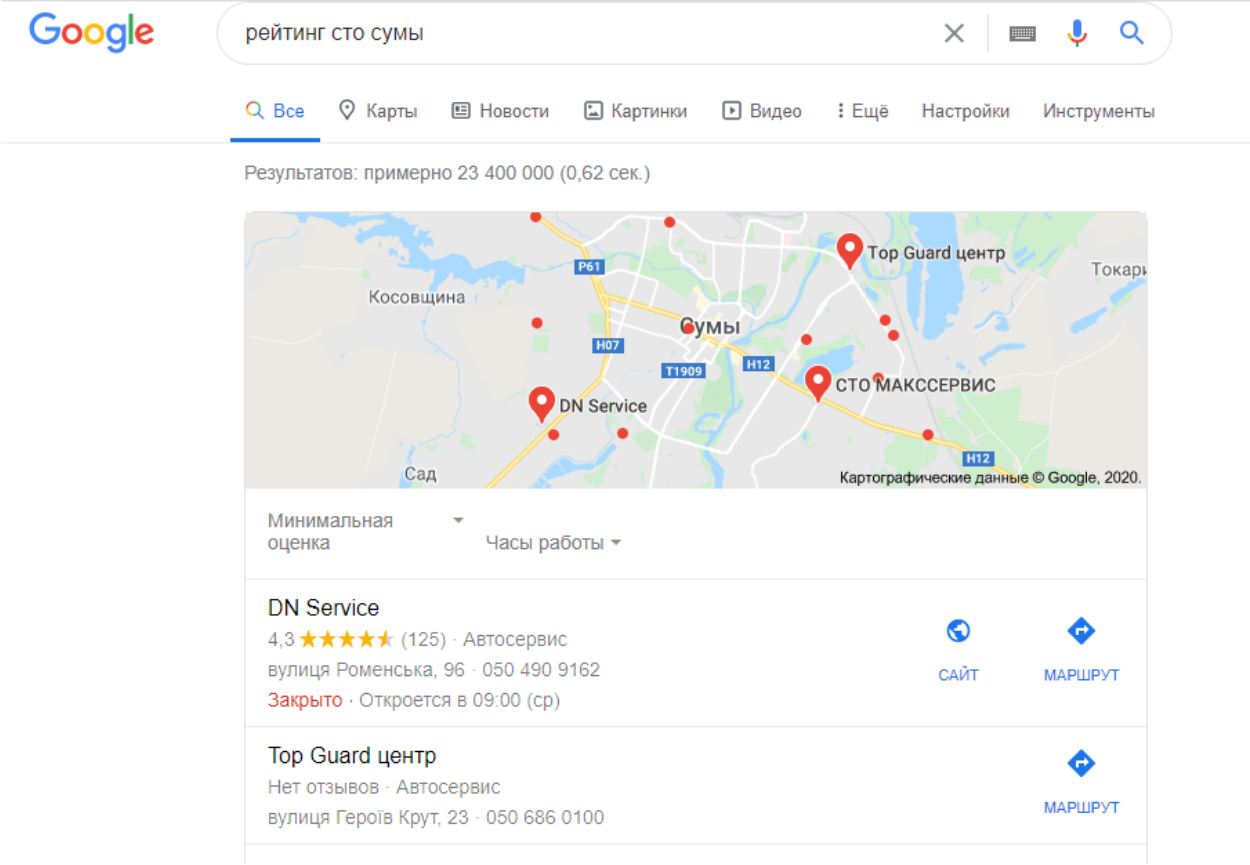


Рисунок 1.2 – Ресурс Google

Останнім є сервіс Vse-sto (рис.1.3) [23]. Він є прямим конкурентом моєї розробки, проте має критичні недоліки, а саме застарілий перенавантажений інтерфейс та багато зайвої реклами.

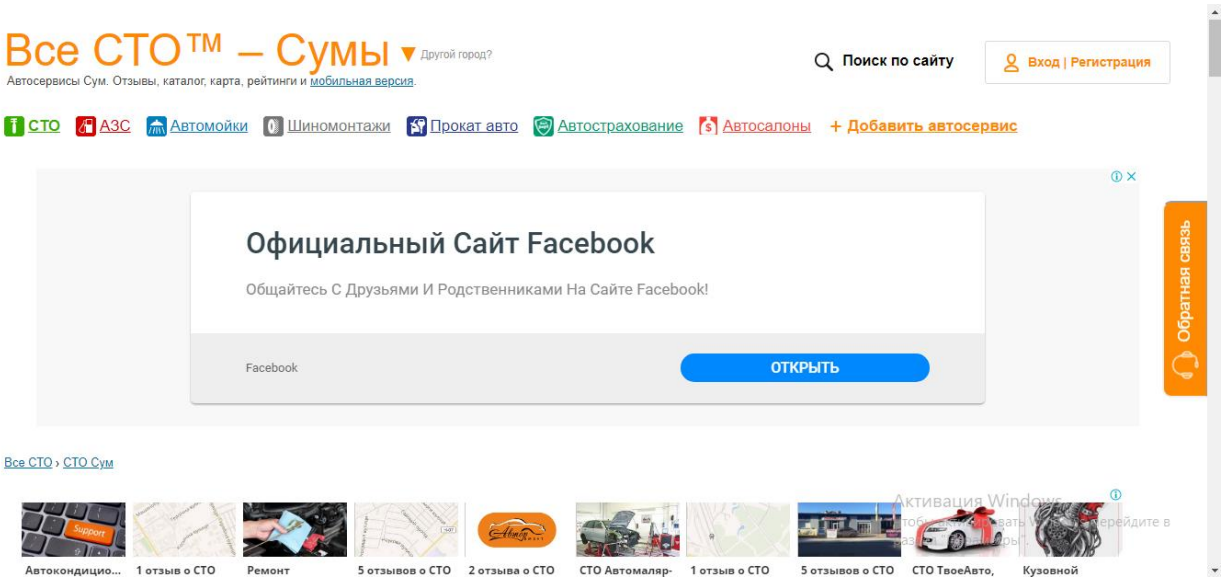


Рисунок 1.3 – Ресурс «vse-sto»

Далі наведено порівняльну таблицю для порівняння з конкурентами.

Таблиця 1.1 – Порівняння аналогів

| **Характеристика** | **Vse-sto** | **Google** |
|---|---|---|
| Зручний інтерфейс | - | - |
| Можливість залишити розгорнутий відгук с файлами-додатками | - | + |
| Нав’язлива реклама | + | + |
| Зручна навігація | - | - |

## 1.3 Постановка задачі

Узагальнена постановка задачі полягає у розробці веб-сервісу для оцінки наданих послуг на СТО, мийках або в дилерських центрах.

Додаток, що розробляється,повинен виконуватиувати ряд задач:

- збереження та фільтрація відгуків про сервіси;
- відображення шуканої інформації;
- постійний та швидкий доступ до додатку, без необхідності встановлення додаткового ПЗ;
- постійний та швидкий доступ до додатку, незалежно від типу пристрою та операційної системи користувача.

Для реалізації поставленого завдання були виділені наступні задачі та дослідження:

- розробка інтерфейсу;
- розробка модулю пошуку;
- розробка модулю авторизації та реєстрації;
- відладка та тестування;
- оформлення супроводжуючої документації.

Було розроблено карту сайту (додаток А) та обрано такі технології розробки: Node.js, React.js, MongoDB та Express.js. MERN стек є дуже популярним та актуальним. Ці технології було обрано саме через велику кількість доступних матеріалів та community. Крім того на сьогодні JS є однією з найрейтинговіших для створення Web-додатків (рис.1.3) [25].

Node.js – це середовище запуску js коду, щоб він міг працювати не лише в браузері. Express js – фреймворк для back-end частини, чудово взаємодіє з хмарною ДБ – MongoDB. React js – фреймоврк для розробки front-end частини додатку.

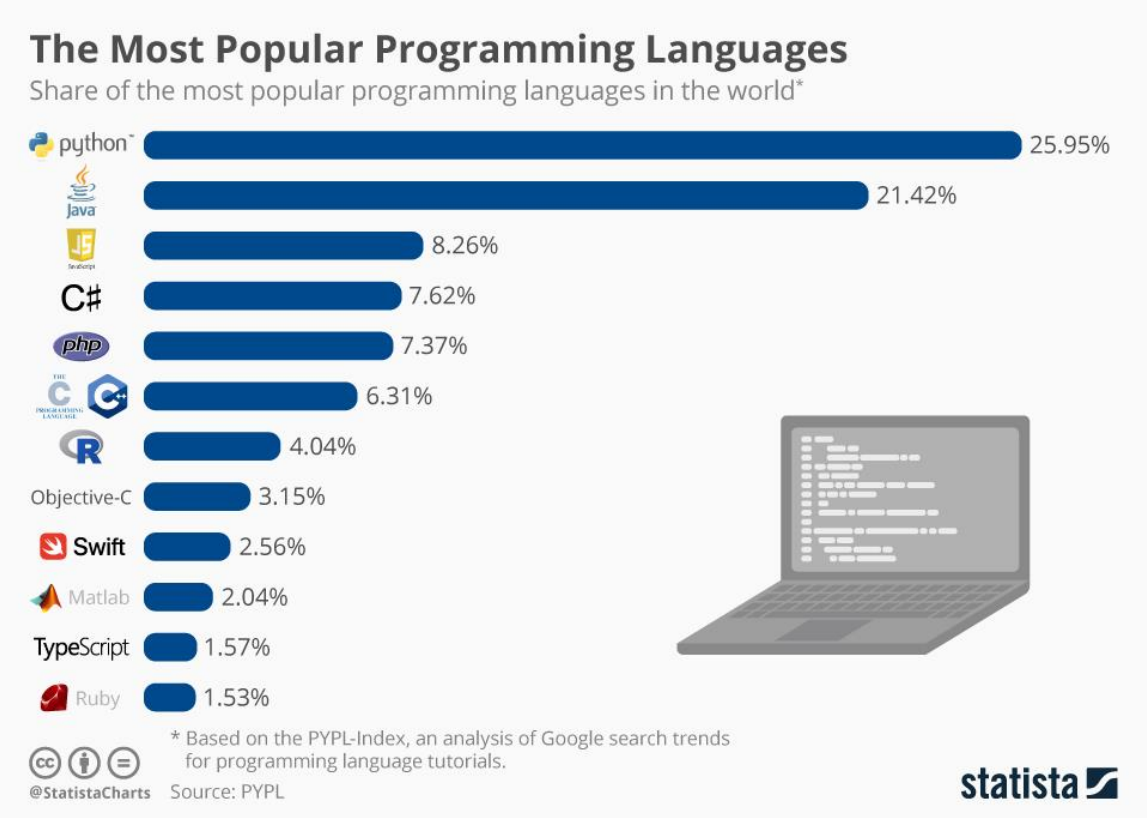


Рисунок. 1.3 – Рейтинг мов програмування

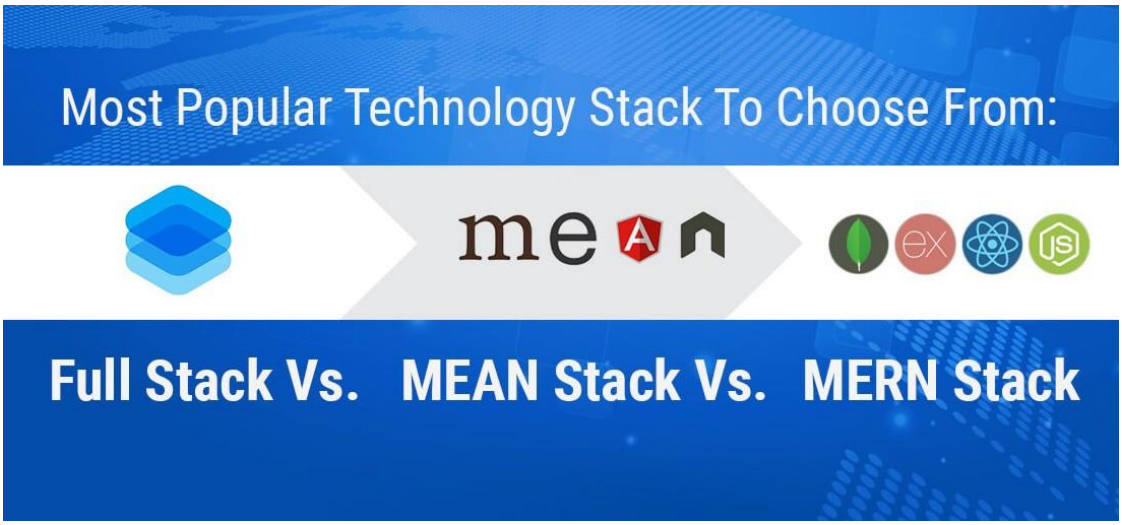


Рисунок. 1.4 – Приклад full-stack фреймворків для web-додатків [24]

Технічне завдання в повному обсязі на розробку наведено в додатку А.

Для вчасного та ефективного виконання проекту сплановано всі роботи (додаток Б).

# 2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ДОДАТКУ

## 2.1 Структурно-функціональне моделювання

Для чіткого представлення роботи сервісу проведено структурно-функціональне моделювання цього процесу згідно стандарту IDEF0. Створена модель у форматі to-be.

Для візуалізації розповсюджених процессів була створена структурно-функцональна модель у вигляді IDEF0 діаграми

Головні елементи для побудови діаграми першого рівня:

Вхідні дані: Інформація від користувача

Вихідні дані: Відображення інформації, нова інформація в БД.

Управління: Інформація у базі даних, правила роботи з сервісом.

Механізми: Технічне забезпечення, користувач, Web-додаток.

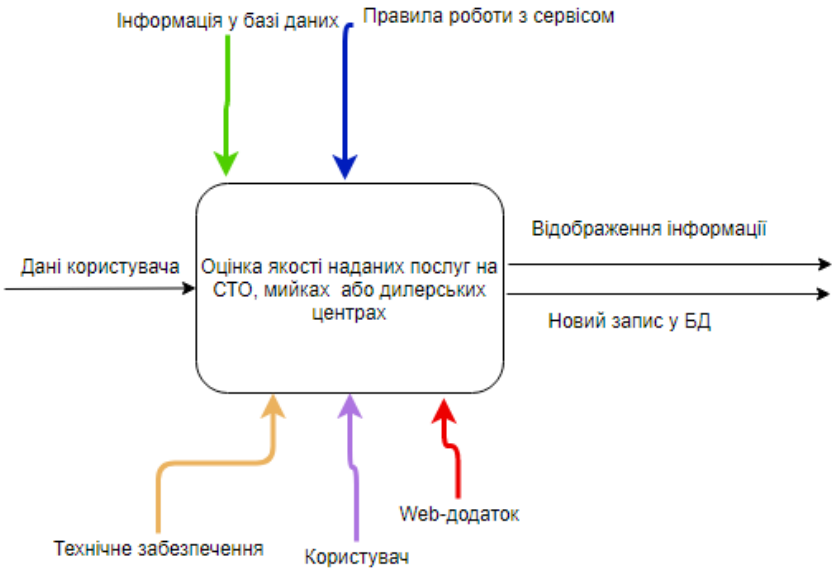


Рисунок 2.1 – Контекстна діаграма

Наступний крок – проведення декомпозиції основного процесу. Діаграма декомпозиції першого рівня представлена на рис.2.2.

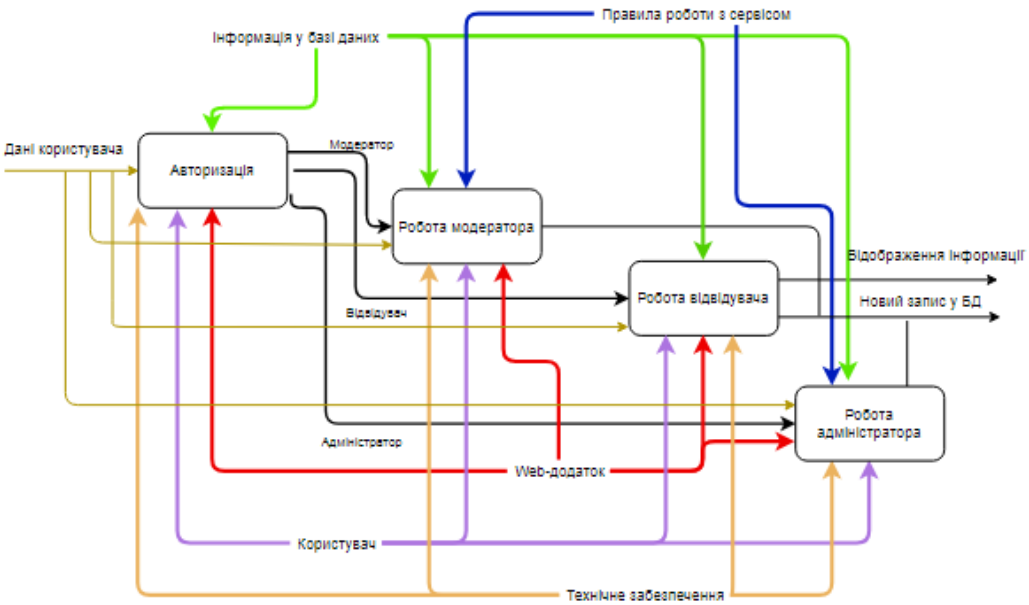


Рисунок 2.2 – Діаграма декомпозиції  першого рівня

## 2.1 UML-моделювання

Діаграма варіантів використання дає концептуальне уявлення про роботу майбутньої системи. Вона складається з варіантів використання, акторів, та відносин між ними.

Варіанти використання це описання послідовності дій які робить система у відповідь на діяльність користувачів, або інших програмних систем. Вони відображають функціональність системи.

User – користувач сервісу:

- Залишає відгуки

- Шукає інформацію
- Фільтрує дані

Moderator – має усі права користувача, а також:

- Видаляти коментарі
- Видаляти відгуки

Admin – має усі права модератора, а також:

- Може редагувати користувачів
- Може видаляти користувачів

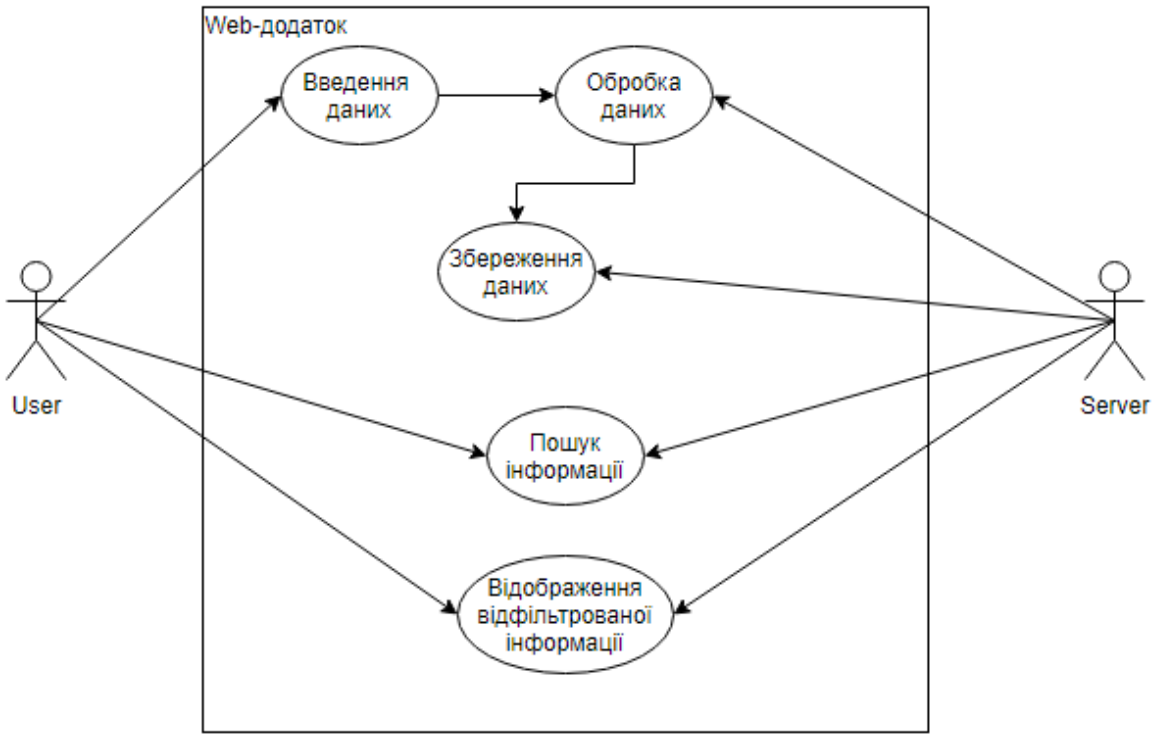


Рисунок 2.3 – Діаграма варіантів використання для ролі User

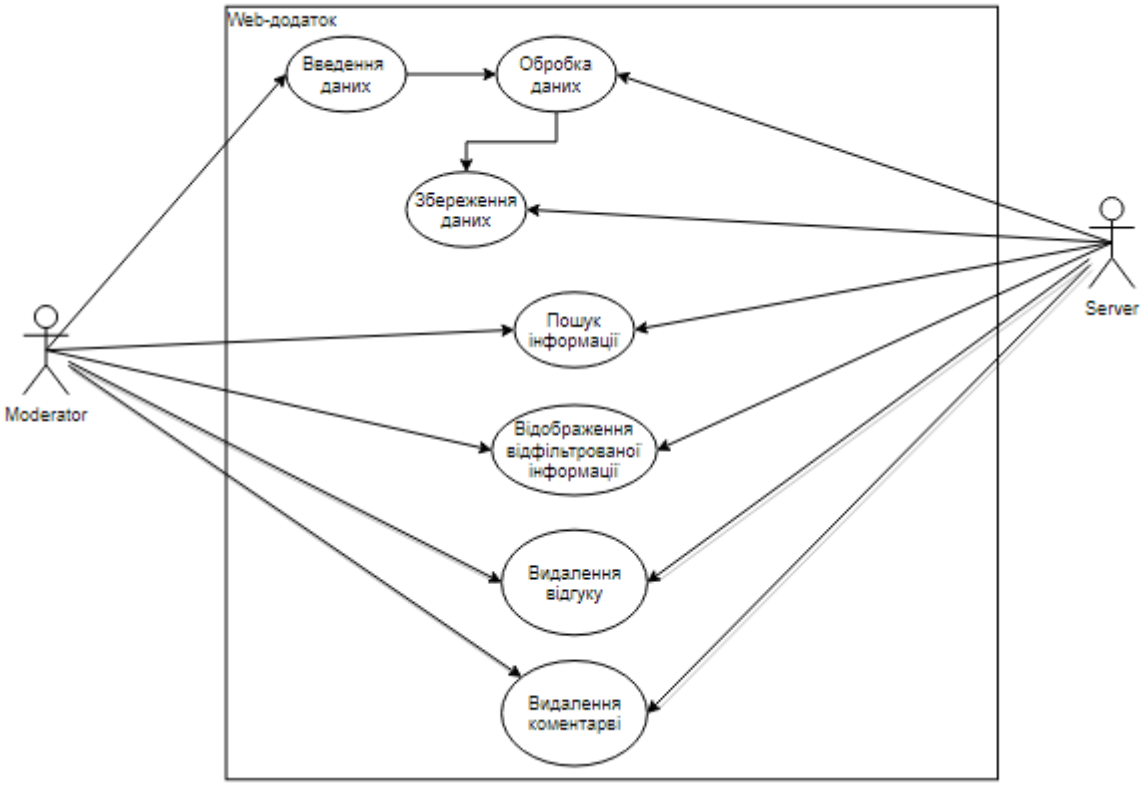


Рисунок 2.4 – Діаграма варіантів використання для ролі Moderator

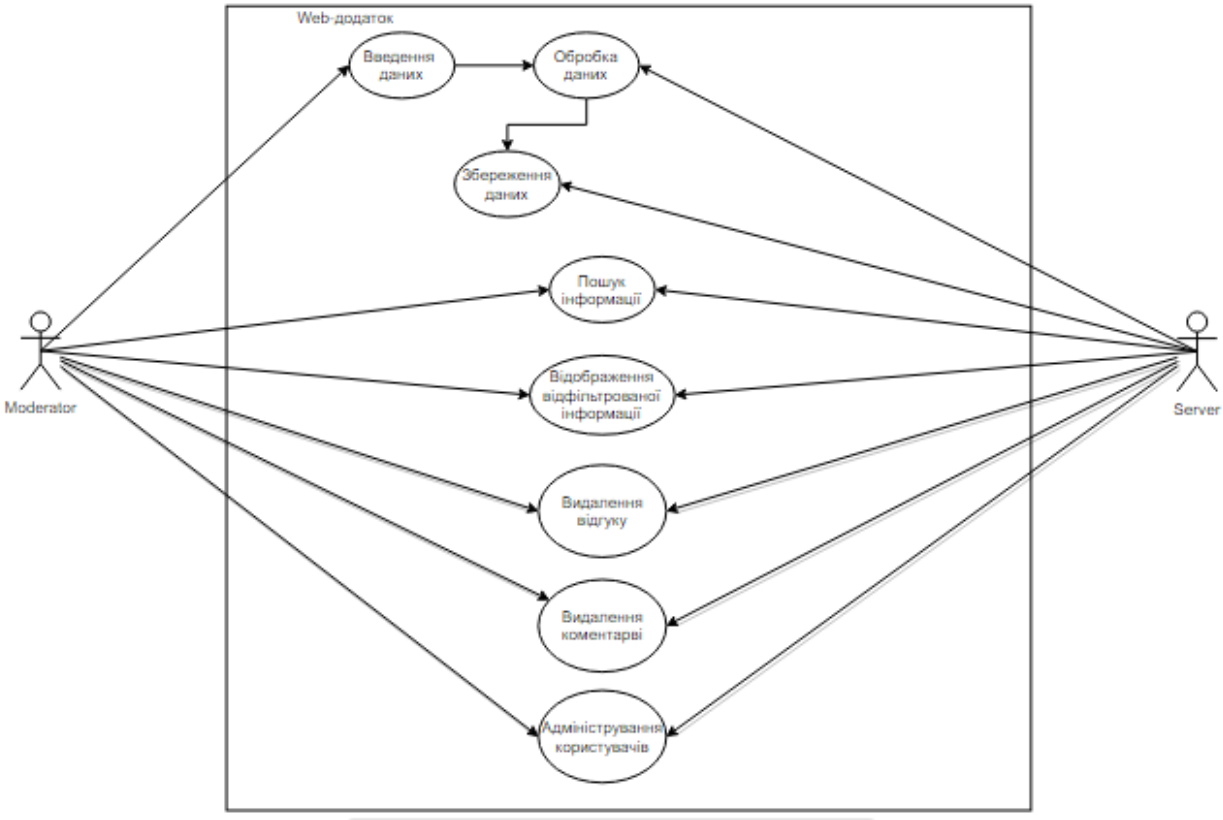


Рисунок 2.5 – Діаграма варіантів використання для ролі Admin

Діаграма класів (class diagram) (рис.2.3) використовується для представлення статичної структури моделі системи в термінології класів об'єктно - орієнтованого програмування.

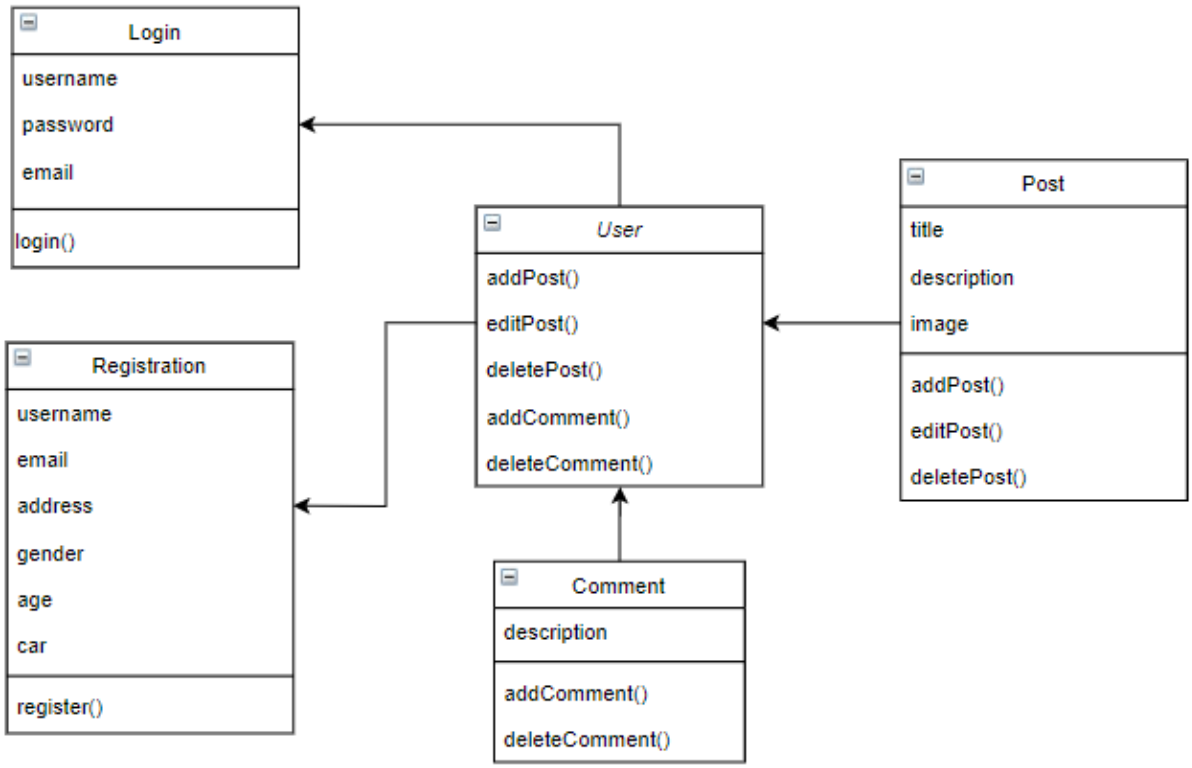


Рисунок 2.4 – Діаграма класів додатку

Для спрощення сприйняття розробки додатково були створені діаграми комунікацій та автоматів (рис. 2.5 та 2.6).

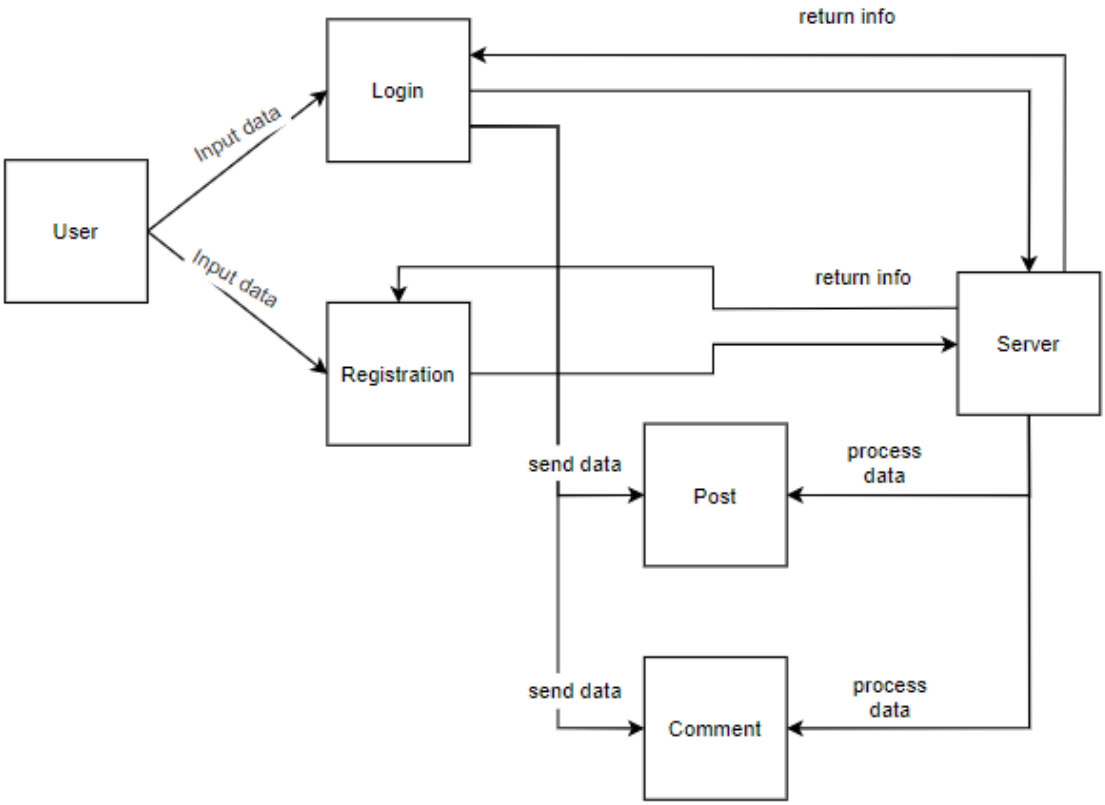


Рисунок 2.5 – Діаграма комунікації

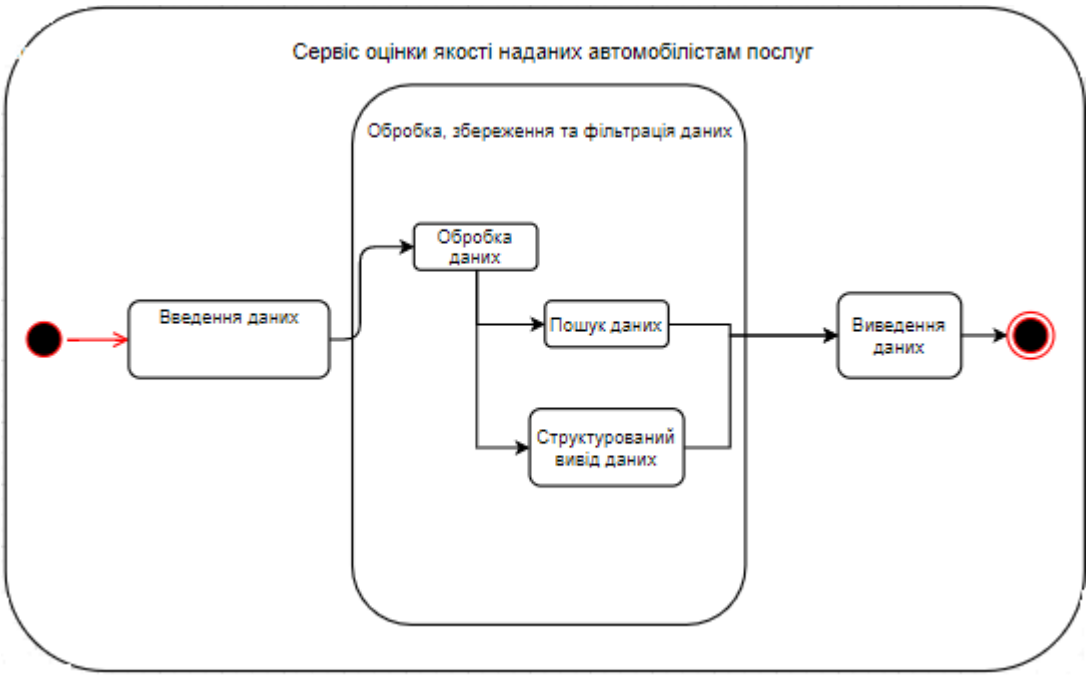


Рисунок 2.6 – Діаграми автоматів

Програмні модулі майбутнього проекту та структурні зв’язки між ними представлено на діаграмі компонентів (рис. 2.7).

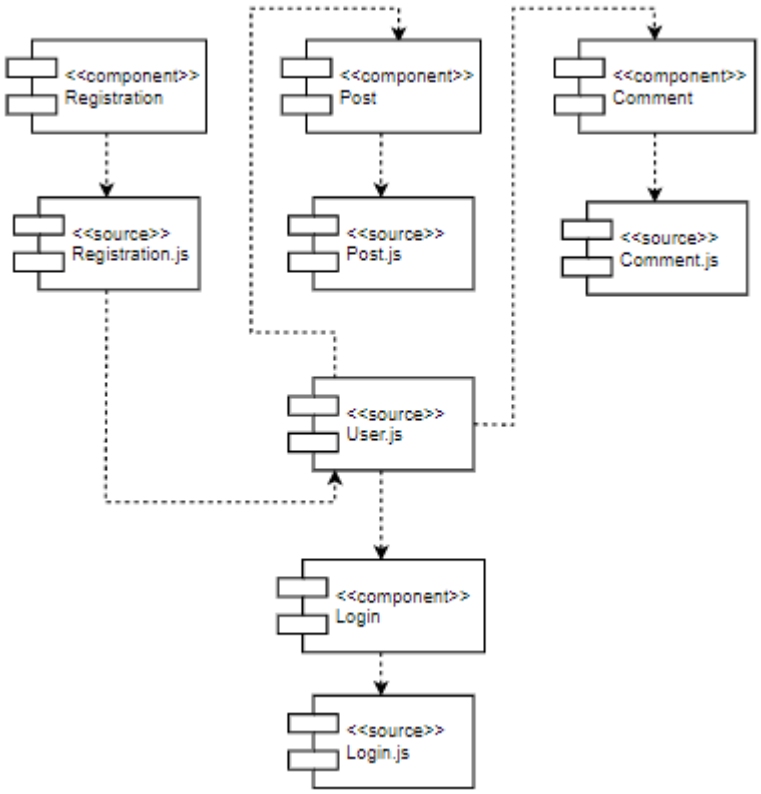


Рисунок 2.7 – Діаграми компонентів

## 2.2 База даних

База даних додатку складається з п’яти моделей, що представлені у вигляді json-документів. Кожна модель має певні опціональні та обов’язкові поля різного типу.

Модель User зберігає імейл, нікнейм, пароль, роль та масив постів, для яких користувач є автором. Таким чином User поєднується з моделлю Post, що подібно до моделі користувача зберігає усі необхідні дані постів. Найменшою є сутність Comments, адже коментарі складаються лише з

самого повідомлення та посилання на автора. Наступними спорідненими сутностями є Service та Review. Модель сервісів створена для зберігання інформації про сервіси, таку як опис, зображення та середню оцінку від клієнтів. Оцінка формується завдяки підрахунку інформації з моделі відгуків, адже кожний з них окрім повідомлення, зберігає й оцінку від 1 до 5 балів.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

## 3.1 Архітектура програмного додатку

На рисунку 3.1 зображено схему архітектури додатку.

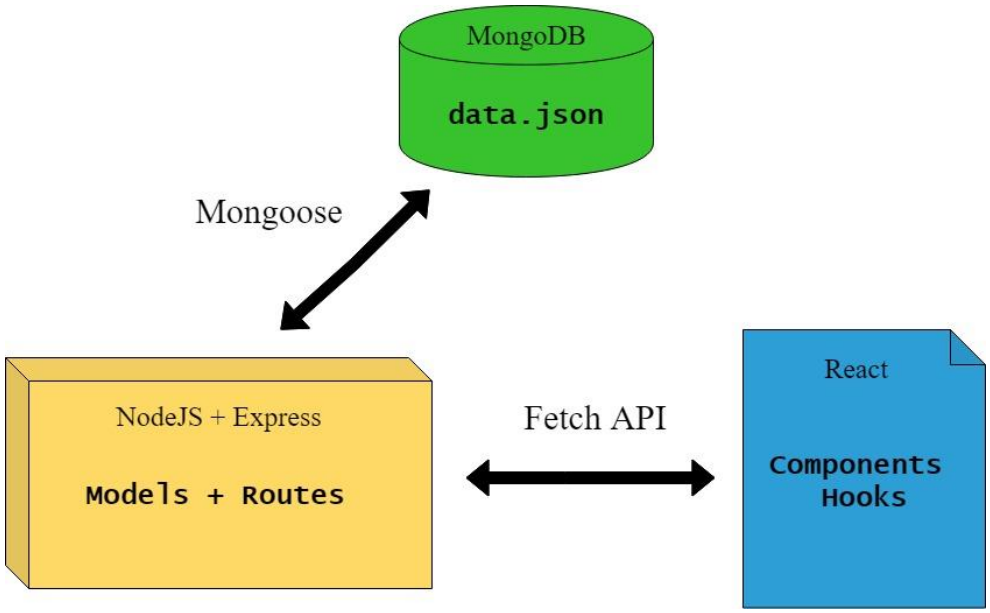


Рисунок 3.1 – Архітектура додатку

### 3.1.1 Серверна частина

За роботу сервера відповідає NodeJs у поєднанні з фреймворком ExpressJs. Саме ExpressJs виконує налаштування середовища та забезпечує канали зв'язку бази даних та клієнта завдяки роутам. Роут описує шлях, по якому можна звернутися до певної моделі та описує дії, що з нею виконуються. Для безперебійного перезавантаження серверу після внесення змін, використовується пакет Nodemon.

NodeJs – це платформа, що дозволяє компілювати Java Script в машинний код, таким чином ви можете створити будь-який додаток без обмежень, адже раніше Java Script працював лише у браузерах.

### 3.1.2 База даних

Сховищем даних для мого додатку є MongoDB. Це NoSQL система керування БД, написана на C++. Для збереження та передачі інформації використовується Json-подібні документи, саме це обумовлює швидку роботу навіть дуже великих веб-додатків. Для поєднання серверу та бази даних я використав пакет Mongoose. За його допомогою були описані усі моделі додатку. Саме моделі визначають структуру Json-файлів, котрі зберігаються у MongoDB.

### 3.1.3 Клієнтська частина

За відображення та рендеринг сторінок додатку відповідає бібліотека React. React працює на JSX – це розширення для мови Java Script. Воно допомагає бібліотеці зрозуміти, як повинен виглядати інтерфейс користувача. Саму на JSX описані усі компоненти. Після компіляції JSX стає звичайним викликом Java Script функції, що повертає об’єкт. Це дозволяє передавати JSX елементи в функції та використовувати їх багаторазово. Сайти написані на React працюють дуже швидко, завдяки технології рендеренгу, яка оновлює тільки ті DOM-елементи, що були змінені. Для реалізації такої системи необхідно працювати зі станом елементів сторінки. React компоненти є імутабельними, тобто незмінюваними, щоб оновити стан компоненту його необхідно перезаписати. Для цього використовуються React hooks – функції, що приймають як аргумент початковий стан елементу та функцію-callback, яка його оновлює. Щоб передавати дані на сервер, я використав вбудовану в Java Script FetchAPI. Це інтерфейс для отримання ресурсів, в тому числі по мережі. Головним аргументом, який приймає метод fetch(), є шлях до необхідного ресурсу. Тобто, в метод fetch ми передаємо роутери, котрі ми описали раніше, таким чином клієнт розуміє що і куди йому відправляти.

## 3.2 Програмна реалізація

### 3.2.1 Налаштування серверу

Для налаштування серверу було встановлено Express та Nodemon. Головним файлов є app.js. В ньому створена функція запуску серверу

```
const PORT = config.get('port') || 5000;

async function start() {
    try {
        await mongoose.connect(config.get('mongoUri'), {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true
        })
        app.listen(PORT, () => {
            console.log(App has been started on PORT: ${PORT}...);
        })
    } catch (e) {
        console.log('Server error', e);
        process.exit(1)
    }
}

start()
```

В тілі функції відбувається приєднання до БД через mongoose.connect() та оголошується порт, котрий імпортується з файлу конфігурації. Усі асинхронні запити бажано обробляти у конструкції try{…} catch(e){…} для легкого пошуку та аналізу помилок. Крім того, для роботи роутерів, їх необхідно ініціалізувати у головному файлі.

```
const express = require('express');
const app = express();
const mongoose = require('mongoose');
const config = require('config');
```

```
app.use(express.json());

app.use('/api/auth', require('./routes/auth.routes'))
```

### 3.2.1 Моделі даних

Наступними кроками для створення додатку були опис моделей та роутерів. Почнемо з моделей, котрі є екземплярами класу Schema. По своїй структурі вони є звичайними Java Script об’єктами, що приймають ключі, як назву поля, а значення, як типізацію цих полів. Для повноти опису полів значення також може бути представленим у вигляді об’єктів з певним набором параметрів. Як приклад, розглянемо модель User.

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    email: {type: String, required: true, unique: true},
    name:{type: String, required: true, unique: true},
    password: {type: String, required: true},
    role: {type: String, default: 'user'},
    posts: [{type: Types.ObjectId, ref: 'Post'}]
})

module.exports = model('User', schema)
```

Щоб пов’язати моделі між собою можна передати в поле моделі посилання на іншу модель, що і було зроблено з posts з використанням об’єкта Type пакету Mongoose.

Для використання моделей в інших частинах проекту їх експортують через функцію model(), котра приймає два аргументи. Перший – назва моделі, другий – екземпляр класу Schema.

Далі наведено приклад об’єкту с бази даних MongoDB (рис. 3.2).


Рисунок 3.2 – Об’єкт «user» з бази даних

### 3.2.2 Роути

Для обробки моделей необхідно описати методи, що допоможуть опрацьовувати дані та передавати їх в БД. Саме для цього і розроблені роутери. Для їх опису та створення я використовував ExpressJs. Для ініціалізації роуту необхідно імпортувати функцію Router з ExpressJs та передати її в змінну. В Java Script усе будується на об’єктах та прототипах, функція не є виключенням. Тому вона має власні методи. В нашому випадку ми визиваємо необхідний метод для опису роуту. Основними методами для роботи с даними та їх передачі є POST – створення нового ресурсу, PUT – зміна ресурсу, GET – отримання даних, та DELETE – для видалення. Як приклад, розглянемо роут реєстрації користувача.

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

router.post('/registration',
    [
        check('email', 'Email невалидный').isEmail(),
        check('password', 'Минимальная длина пароля 6 символов')
            .isLength({min: 6})
```

```
        ]
        , async (req, res) => {
        try {
            const errors = validationResult(req)
            if(!errors.isEmpty()) {
                return res.status(400).json({
                    errors: errors.array(),
                    message: 'Невалидные данные при регистрации'
                })
            }

            const {email, password, name} = req.body;

            const candidate = await User.findOne({email});

            if(candidate) {
                return  await  res.status(400).json({message:  'Такой  пользователь
уже существует'})
            }

            const hashedPassword = await bcrypt.hash(password, 15);
            const user = new User({email, name, password: hashedPassword});
            await user.save();

            await res.status(201).json({message: 'Пользователь успешно создан'});


        } catch (e) {
            await res.status(500).json({ message: 'Что-то пошло не так...'})
        }
    })
```

Функція ініціалізації може приймати безліч аргументів. В даному випадку їх три. Перший – шлях, по якому можна визвати функцію обробки даних, другий – масив, в якому визивається функція check(), для валідації полів моделі, та третій – колбек для обробки та зміни даних. В ньому і відбуваються головні дії з даними. Через об’єкт req ми отримаємо дані з форми та порівнюємо поле email через виклик User.find(). Це і є наша модель

у котрій ми шукаємо необхідне поле, адже якщо користувач з такою поштою вже існує, ми повинні припинити виконання функції та повідомити про це користувача. Якщо пошта є унікально, можна створювати новий екземпляр моделі користувача шляхом передачі в аргументи конструктора даних з форми. Зверніть увагу на те, що для безпеки та збереження даних, пароль було зашифровано за допомогою пакету bcrypt. Останніми кроками є виклик методу save() для збереження в БД нового користувача та повернення необхідного статусу запиту через об’єкт res.

Саме за такою схемою були створені усі моделі та роути.

### 3.2.3 Компоненти

Усі сторінки складаються с багатьох компонентів. Для їх розробки я використав бібліотеку React Bootstrap, що дає великий набір інструментів для роботи з різноманітними компонентами. Як було описано раніше, React використовує формат jsx для створення елементів. Кожна сторінка це набір компонентів, котрі в свою чергу складаються з елементів. Для опису компонентів необхідно створити файл формату jsx, який являє собою функцію, що повертає необхідний jsx об’єкт. Усі компоненти імпортуються в головний клієнтський файл App.js. Для формування стилів елементів я використовував комбінацію двох методів. Перший – оголошення класів з bootstrap grid та опис власних css стилів у файлі App.css.

### 3.2.4 React Hooks

Зміна станів та обробка даних з клієнту – головні задачі, що вирішуються завдяки React Hooks. Розглянемо приклад сторінки реєстрації.

```
import React, {useEffect, useState} from "react";
import {Form, Button} from "react-bootstrap";
import {useHttp} from "../../hooks/http.hook";

function Register() {
  const {loading, request} = useHttp();
```

```
  const [form, setForm] = useState({
    email: '', name: '', password: '', repeat_password: '', role: 'user'
  })

  const changeHandler = event => {
    setForm({ ...form, [event.target.name]: event.target.value })
  }
  const registerHandler = async () => {
    try {
      const data = await request ('/api/auth/registration', 'POST', {...form})
      console.log('Data ', data)
    } catch (e) {}
  }
  return (
    <div className="form">
      <Form>
        <Form.Group controlId="email">
          <Form.Label>Email</Form.Label>
          <Form.Control
            required
            size="lg"
            id="email"
            name="email"
            type="email"
            value={form.email}
            placeholder="Введите email"
            onChange={changeHandler}
          />
        </Form.Group>
...
      </Form>
    </div>
  )
}

export default Register
```

Хуки, що використані на цій сторінці, – useHttp() та useState(). useHttp – це кастомний хук, створений власноруч для передачі даних з клієнта на

сервер. В ньому викликається метод fetch(), котрий приймає ті дані, які мі вказуємо при виклику хука. Таким чином, використовуючи один хук, я зміг опрацьовувати компоненти всього проекту та передавати дані на необхідні роути. Хук useState є вбудованим в React та співпрацює з changeHandler. form, це об'єкт, котрий зберігає початкове значення функції, а функція setForm – оновлює стан елементів, котрі ми ініціалізували в об'єкті useState. changeHandler викликає у собі setForm та забирає дані з форми через атрибут name, які далі передаються у value. registerHandler викликається при натисканні кнопки реєстрації та відправляє дані у бд за роутом `'/api/auth/registration'`. Подібним чином відбувається опис інших сторінок веб-додатку.

### 3.2.4 React Router

Для правильного відображення сторінок використовується React Router. Цей компонент дозволяє встановити кожній сторінці специфічний шлях для відображення правильних даних. Я виніс налаштування роутингу для клієнту в окремий файл, а потім імпортував його до App.js.

```
export const useRoutes = (isAuthenticated, isModer, isAdmin) => {
    if(isAuthenticated && isAdmin) {
        return (
            <Switch>
                <Route path="/posts/create">
                    <AddPost />
                </Route>
                <Route path="/support">
                    <Help />
                </Route>
                <Route path="/services/:id">
                    <OneService />
                </Route>
                <Route path="/services">
                    <Services />
                </Route>
```

```
                    <Route path="/users">
                        <Users />
                    </Route>
                    <Route path="/">
                        <Home />
                    </Route>
                    <Redirect to="/" />
                </Switch>
            )
        } else if (isAuthenticated && isModer) {
. . .
}
        )}
```

В залежності від ролі, яку відіграє користувач в системі, він бачить різні сторінки додатку. Отже саме React Router відповідає за те, що бачить користувач та за яким посиланням.

Налаштування роутерів в React було останнім кроком в розробці сервісу.

### 3.3 Використання програмного додатку

Після відкриття сайту, користувач бачить головну сторінку (рис.3.3) ‘з постами інших користувачів. Неавторизований користувач має доступ майже до всього функціоналу, окрім створення постів та відгуків.

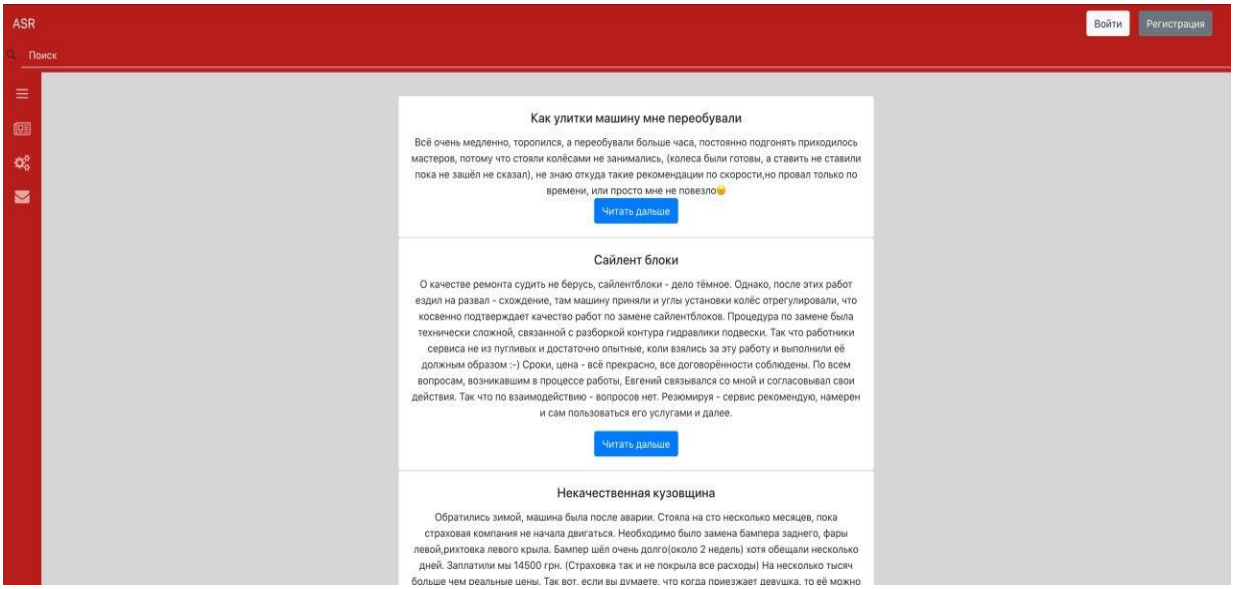


Рисунок 3.3 - Демонстрація екрану не авторизованого користувача

Без авторизації також можна переглянути рейтинг сервісів (рис.3.4).

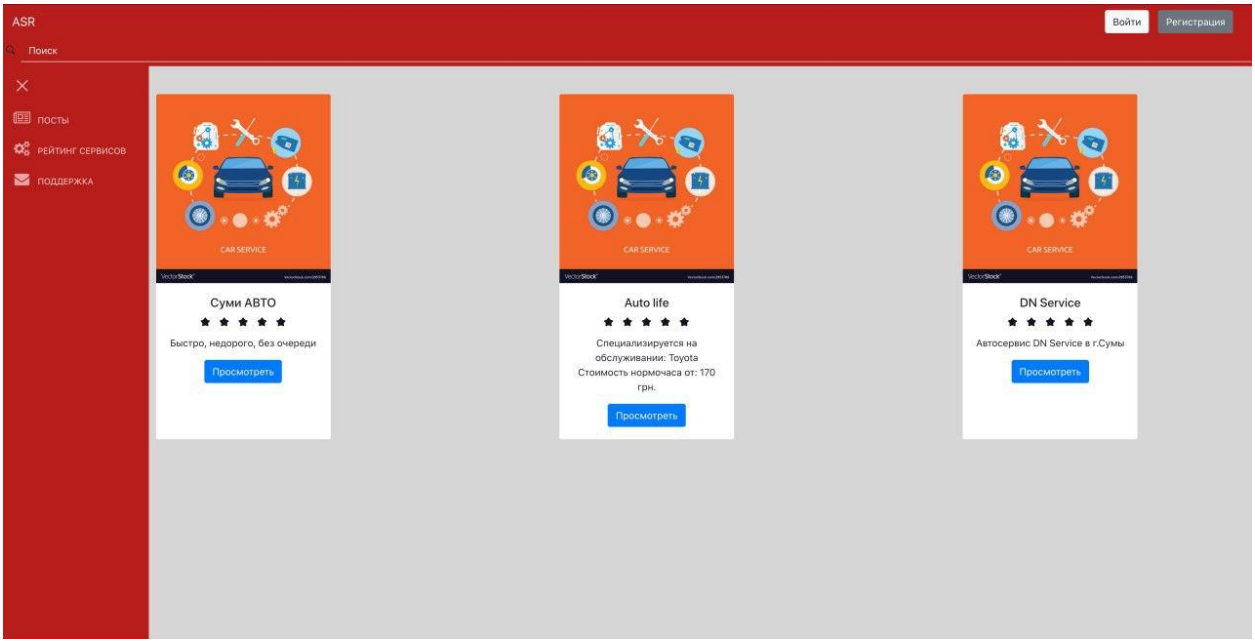


Рисунок 3.4 – Сторінка з рейтингами автосервісів

Щоб отримати доступ до всіх функцій користувачу необхідно пройти процес авторизації та реєстрації (рис 3.5) відповідно. Підказки допоможуть заповнити форму правильно (рис 3.6 – 3.8).

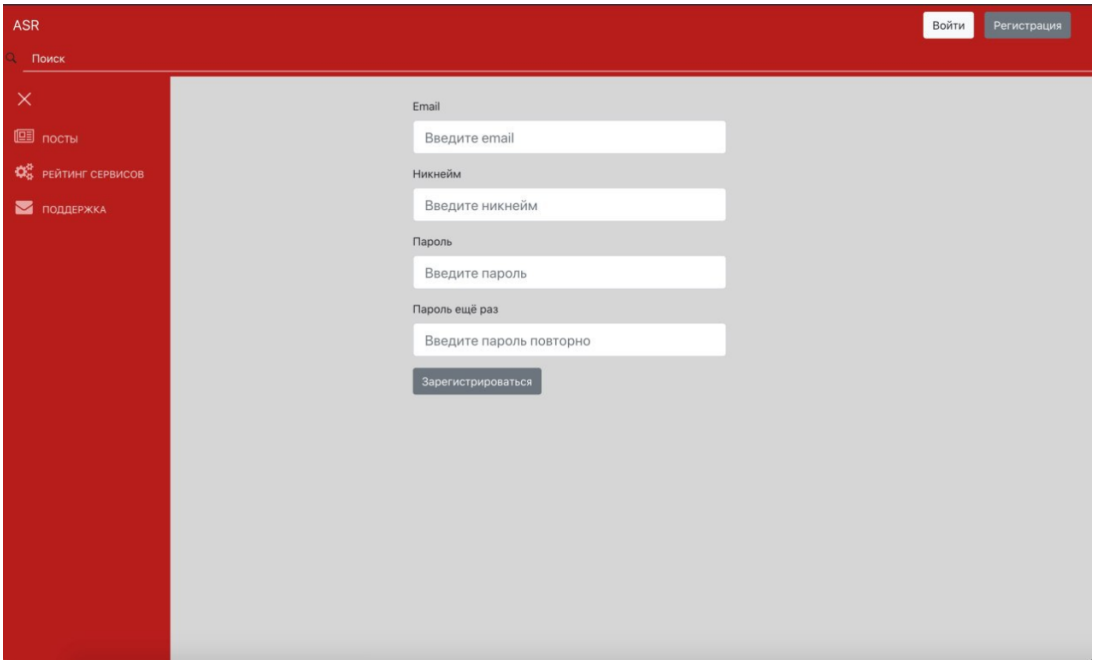


Рисунок 3.5 – Сторінка реєстрації

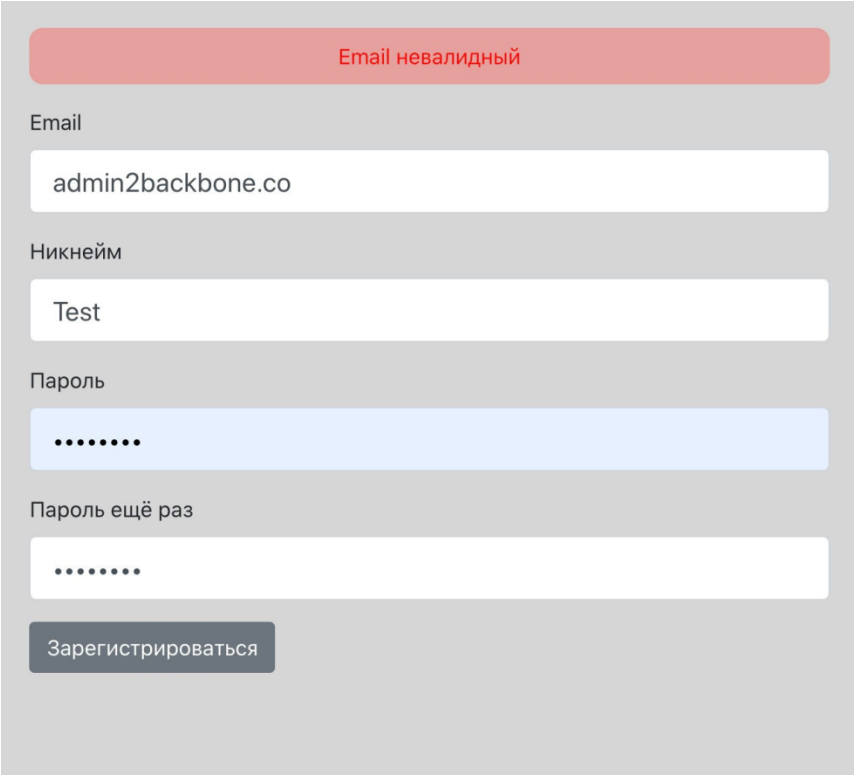


Рисунок 3.6 – Перевірка пошти користувача

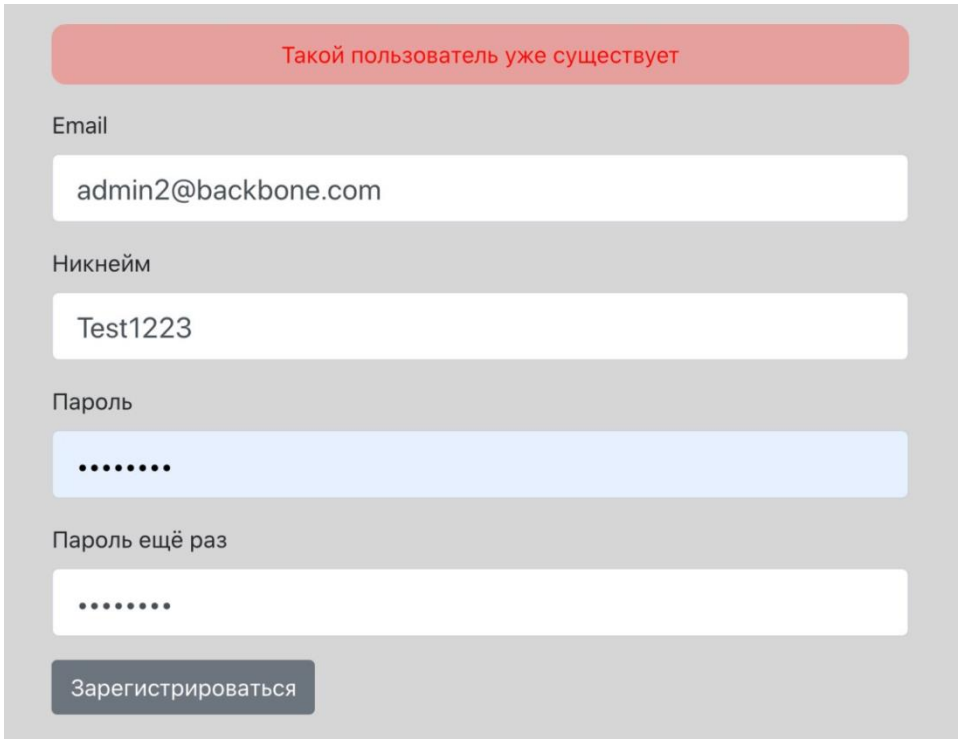


Рисунок 3.7 – Валідація створення користувача з вже зареєстрованою поштою

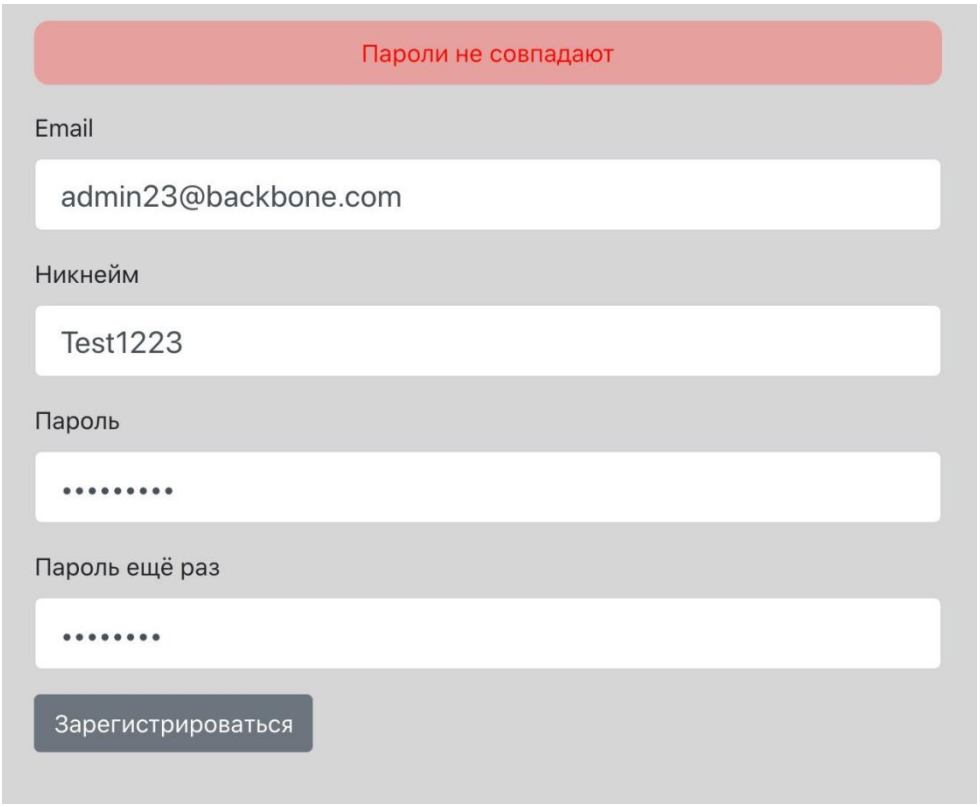


Рисунок 3.8 – Перевірка введення однакових паролів

При виниканні проблем або запитань, завжди можна надіслати лист з описом проблемної ситуації (рис. 3.9).

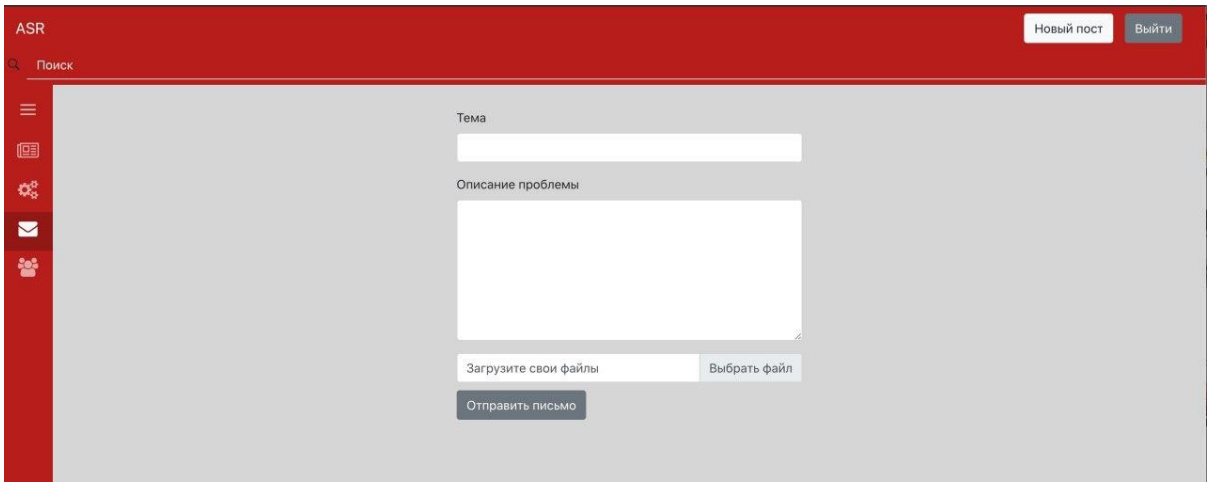


Рисунок 3.9 – Сторінка зворотного зв’язку

Після завершення авторизації, головною функцією є створення посту (рис. 3.10) для того, щоб цікаву історію вашого обслуговування прочитали всі користувачі порталу. Для цього потрібно натиснути кнопку «Новий пост» та заповнити необхідні поля. Обов’язковими є текст та заголовок (рис. 3.11).

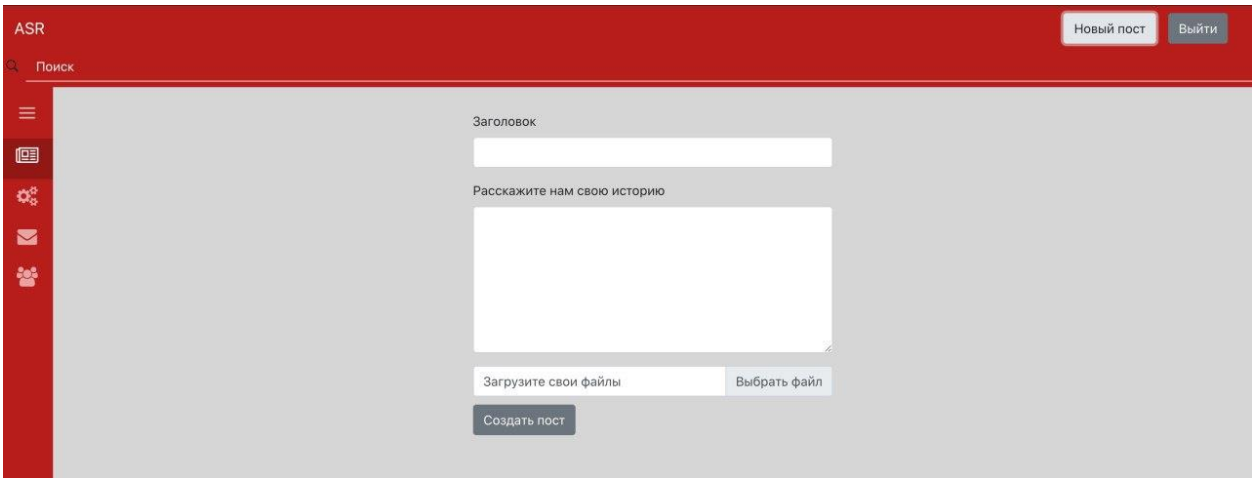


Рисунок 3.10 – Сторінка створення посту

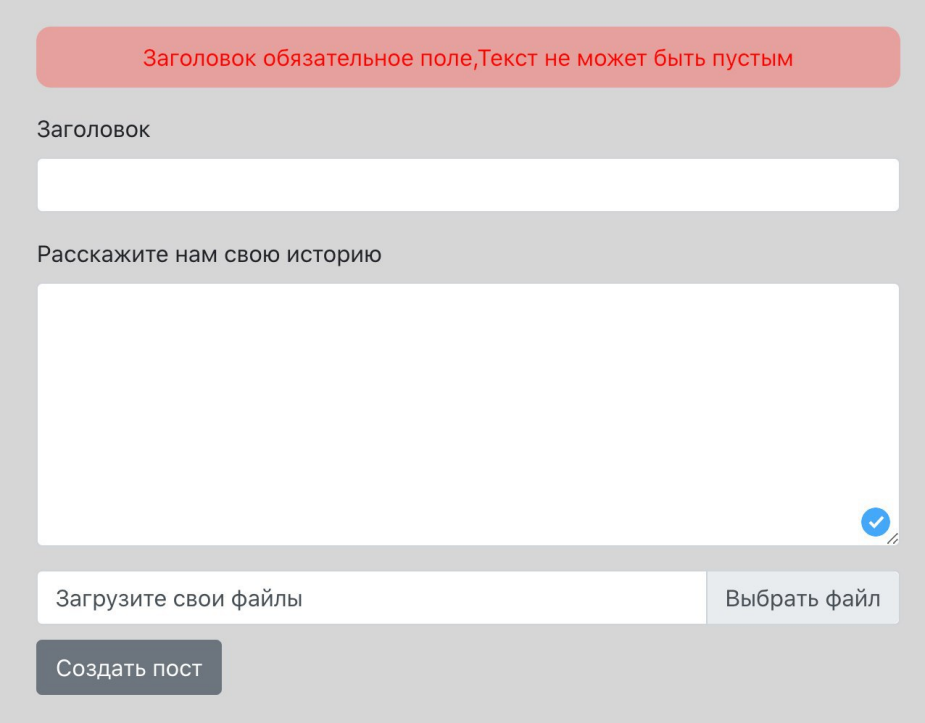


Рисунок 3.11 – Приклад валідації обов’язкових полів

В системі існує три головні ролі, Користувач, Адміністратор та Модератор. Найбільшу різницю в інтерфейсі та додатковому функціоналі має адміністратор. Його головним завданням є менеджмент користувачів та модерів. Наступний рисунок відображає саме сторінку для роботи з користувачами (рис. 3.12).

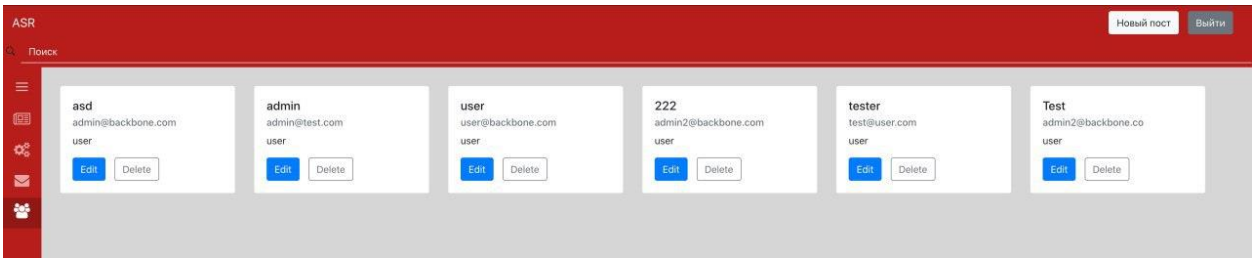


Рисунок 3.12 – Демонстрація екрану адміністратора на сторінці адміністрування користувачів

# ВИСНОВКИ

При виконанні дипломної роботи було проведено аналіз предметної області на основі літературних джерел, визначено актуальність даної роботи. Після чого було прийняте рішення про розробку вебсервісу, у зв’язку з актуальністю обраної теми на сьогоднішній день.

За результатами аналізу складено технічне завдання на виконання робіт та проведено планування робіт.

Для повного розуміння роботи інформаційної системи було створено схему роботи, діаграми формалізації процесів та модель варіантів використання, які дають чітке уявлення про роботу та структуру додатку, а також формалізує процес його розробки.

Щоб оптимізувати швидкість роботи сайту, було використано сучасний стек MERN, що базується на мові JavaScript.

Створений додаток орієнтований на автомобілістів, що прагнуть знайти відповідний автосервіс, використовуючи кращий додаток для пошуку станцій технічного обслуговування, мийок або дилерських центрів. Простий та інтуїтивно зрозумілий інтерфейс, відсутність реклами, можливість поділитися своєю історію – головні конкурентні переваги даної розробки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крокфорд Д. JavaScript: сильные стороны. – СПб.: Питер, 2012.

2. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих. Node.js в действии. — СПб.: Питер, 2014.

3. Eric Elliott. Programming JavaScript Applications —O'Reilly Media, 254 p.

4. Douglas Crockford's JavaScript [Електронний ресурс] – Режим доступу до ресурсу: http://crockford.com/javascript/.

5. Полезные статьи о JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.ipipe.ru/info/javascript.html

6. Джава скрипт для начинающих [Електронний ресурс] – Режим доступу до ресурсу: https://www.internet-technologies.ru/articles/yazyk-programmirovaniya-javascript-informaciya-dlya-nachinayuschih.html

7. Статистика количества автомобилей в Украине [Електронний ресурс] – Режим доступу до ресурсу: https://topgir.com.ua/56589905-kolichestvo-avtomobilej-na-dushu-naseleniya-v-ukraine-poslednie-dannye/

8. Список стран по количеству автомобилей [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Список_стран_по_количеству_автомобилей_на_1000_человек

9. The Software Quality Company [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index//

10. Trends in JS development [Електронний ресурс] – Режим доступу до ресурсу: https://trends.builtwith.com/docinfo/Javascript

11. Client side technologies [Електронний ресурс] – Режим доступу до ресурсу: https://w3techs.com/technologies/overview/client_side_language

12. Плюсы ReactJs [Електронний ресурс] – Режим доступу до ресурсу: https://xbsoftware.ru/blog/pochemu-stoit-ispolzovat-react-js-razrabotke-prilozhenij/

13. Особенности NodeJs [Електронний ресурс] – Режим доступу до ресурсу: https://senior.ua/articles/pochemu-node-js-osobennosti-i-preimuschestva

14. HTML&CSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/standards/webdesign/htmlcss

15. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/html401/

16. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://timeweb.com/ru/community/articles/plyusy-i-minusy-bootstrap-1

17. Блог о ReactJs [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/ruvds/blog/349452/

18. Статся про покращення якості сервісу [Електронний ресурс] – Режим доступу до ресурсу: https://service.yaware.com.ua/blog/yak-pokrashhyty-yakist-servisu/

19. Блог про качество обслуживания [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/prolan/blog/202768/

20. Как обманывают на автосервисах [Електронний ресурс] – Режим доступу до ресурсу: https://auto-tech.center/blog/deceive

21. Trip advisor [Електронний ресурс] – Режим доступу до ресурсу: https://www.tripadvisor.ru/

22. Google [Електронний ресурс] – Режим доступу до ресурсу:https://www.google.com/

23. Vse-sto [Електронний ресурс] – Режим доступу до ресурсу: https://vse-sto.com.ua/sumi/sto/

24. Most popular JS stacks [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/datadriveninvestor/most-popular-technology-stack-to-choose-from-full-stack-vs-mean-stack-vs-mern-stack-in-2019-d12c0a17439a

25. Most used languages stats [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

# ДОДАТОК А
# ТЕХНІЧНЕ ЗАВДАННЯ

## ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
«Веб-сервіс для оцінки якості обслуговування наданих автомобілістам послуг на СТО, мийках або в дилерських центрах»

**Суми 2020**

# 1 Призначення й мета створення інформаційної системи

## 1.1. Призначення інформаційної системи

Інформаційна система дає користувачу можливість залишити відгук за надані йому послуги на СТО.

## 1.2. Мета створення інформаційної системи

Прозорий сервіс оцінки якості наданих на автосервісах послуг з зручним інтерфейсом.

## 1.3. Цільова аудиторія

У цільовій аудиторії інформаційної системи можна виділити наступні групи:

- автовласники;
- власники сервісів.

# 2 Вимоги до інформаційної системи

## 2.1. Вимоги до інформаційної системи в цілому

### 2.1.1. Вимоги до структури й функціонування інформаційної системи

Інформаційна система повинна бути реалізована у вигляді сайту, доступного в мережі Інтернет під доменним іменем www.asr.io Сайт складається з головної сторінки-списку відгуків та сторінки-списку автосервісів.

### 2.1.2. Вимоги до персоналу

Для підтримки сайту та валідації відгуків або компаній персонал повинен вміти розпізнавати шахраїв та аналізувати дані на предмет корисності для аудиторії додатку.

### 2.1.3. Вимоги до збереженні інформації

Користувачі повинні мати змогу додавати свої зображення та відео файли та завжди мати до них доступ

### 2.1.4. Вимоги до розмежування доступу

Інформація, розташовувана на сайті, є загальнодоступною.

Користувачів сайту можна розділити на 3 групи відповідно до прав доступу:

1. Відвідувачі
2. Модератор
3. Адміністратор

**Відвідувачі** мають доступ тільки до загальнодоступної частини сайту. А саме відгуки та персональні сторінки авто сервісів

**Модератор** може валідувати матеріали та сторінки автосервісів і дозволяти їх публікацією.

**Адміністратор** може виконувати всі ті ж дії, що й Модератор, і крім того:

- додавати й видаляти користувачів із правами Модератора;
- видаляти користувачів\відгуки\персональні сторінки СТО.

Доступ до адміністративної частини повинен здійснюватися з використанням унікального логіна й пароля.

## 2.2. Вимоги до функцій, виконуваних сайтом

### 2.2.1. Основні вимоги

#### 2.2.1.1. Структура сайту

Сайт повинен складатися з наступних розділів:

- Головна сторінка – список усіх відгуків з можливістю їх фільтрації.
- Сторінка-список автосервісів – налічує усі автосервіси та можливість переглянути їх персональну інформацію.
- Правила написання відгуків – окрема сторінка з правилами створення нових публікацій.

#### 2.2.1.2. Навігація

Користувацький інтерфейс сайту повинен забезпечувати наочне, інтуїтивно зрозуміле представлення структури розміщеної на ньому інформації, швидкий і логічний перехід до розділів і сторінок. Навігаційні

елементи повинні забезпечувати однозначне розуміння користувачем їх змісту: посилання на сторінки повинні бути мати заголовок, умовні позначки відповідати загальноприйнятим.

Система повинна забезпечувати навігацію по всіх доступних користувачеві ресурсам і відображати відповідну інформацію. Для навігації повинна використовуватися система контент-меню. Меню повинне являти собою текстовий блок (список гіперпосилань) у лівій колонці.

#### 2.2.1.3. Наповнення сайту (контент)

Сторінки всіх розділів наповнюються користувачами, що залишають відгуки та власниками СТО, котрі створюють персональні сторінки.

Модифікація вмісту розділів повинна здійснюватися модераторами, котрі мають відповідні права та бачать не затвердженні статті.

#### 2.2.1.4. Система навігації (карта сайту)

Взаємозв'язок між розділами й підрозділами сайту (карта сайту) представлено на рисунку А.2.

### 2.2.2. Вимоги до функціональних можливостей

Як відвідувач, я хочу мати можливість продивлятись відгуки про автосервіси та їх персональні сторінки, а також створювати відгуки.

Як модератор, я маю ті ж права, що і Відвідувач, а також повинен мати можливість валідації відгуків та дозволяти їх публікацію.

Як адміністратор, я маю права модератора, а також мати можливість видаляти\редагувати користувачів, відгуки та сторінки автосервісів.

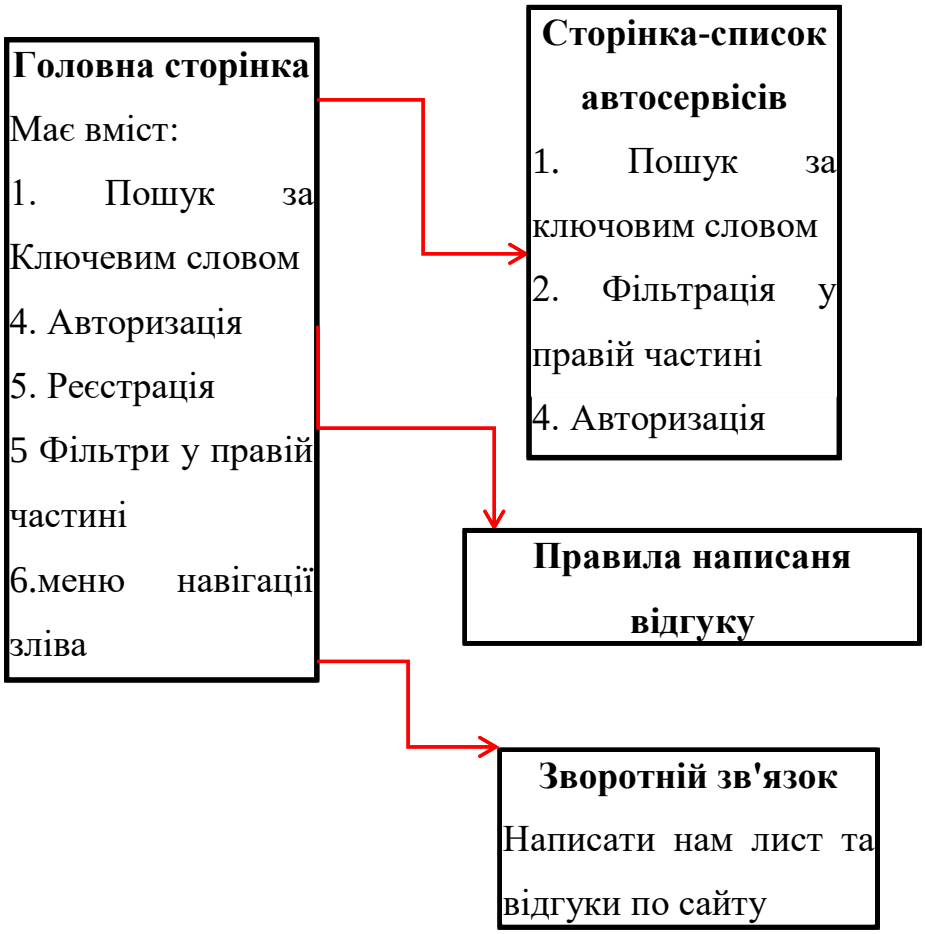


Рисунок А.2 – Карта сайту

#### 2.2.2.1. Функціональні можливості розділів

На головній сторінці будуть представлені наступні елементи:

- Пошук по ключевим словам з можливістю фільтрації за датою та рейтингом;
- Можливість створити власний відгук;
- Авторизація\Реєстрація користувачів;
- Меню навігації зліва
- Перегляд правил написання відгуків;
- Можливість зв’язатися з командою технічної підтримки сайту через зворотній зв’язок;

#### 2.2.2.2. Загальні вимоги

Головна особливість дизайну сайту – зрозумілість та мінімалізм. Сторінки не повинні бути перенавантаженими текстом та інформацією.

Кольорова гама складається з білого, червоного та чорного кольорів. Сайт повинен працювати швидко та без затримок

Розташування елементів на головній сторінці сайту схематично показано на рис. А.2.

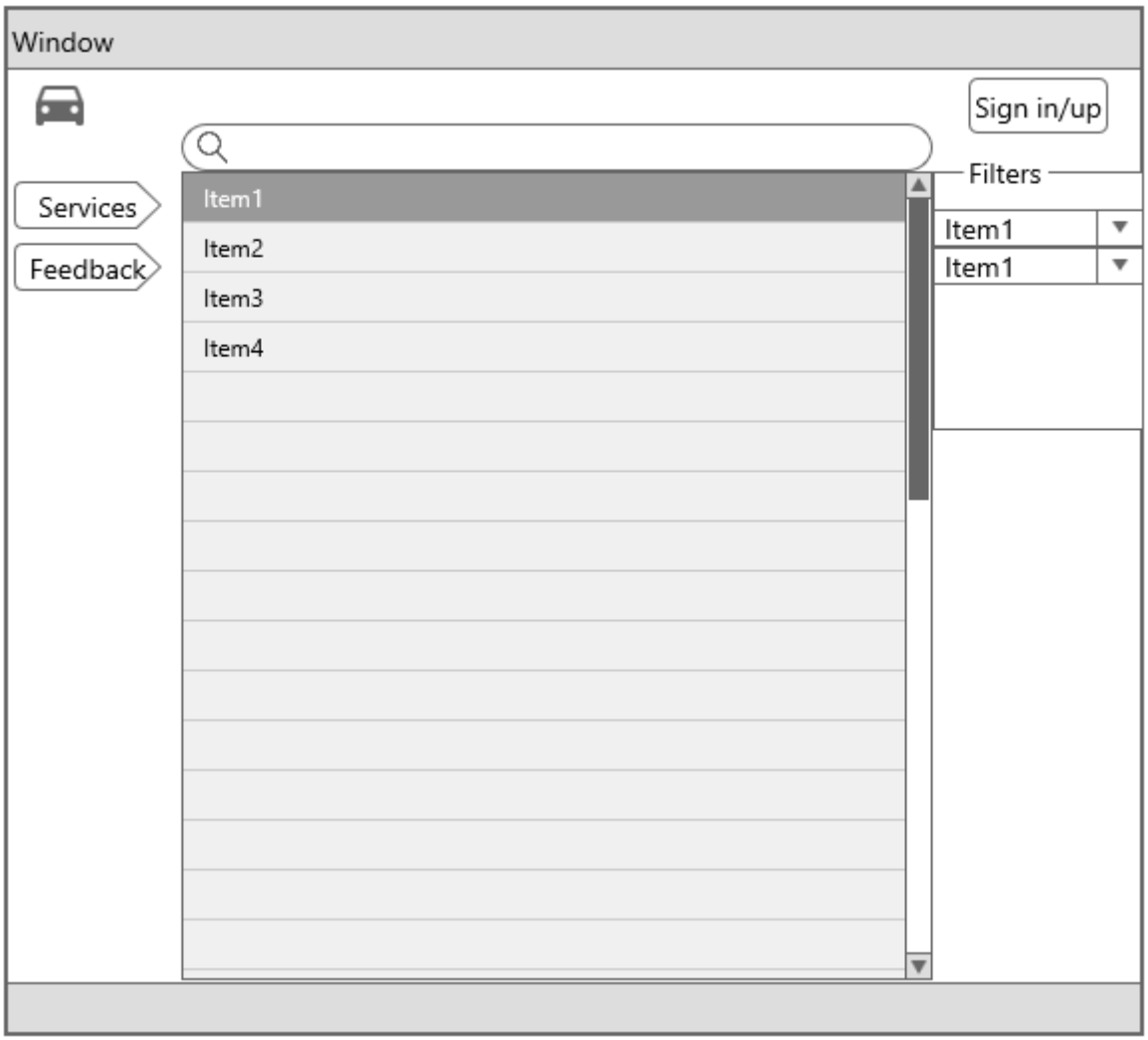


Рисунок А.2 – Типова сторінка

#### 2.2.2.3. Типові навігаційні й інформаційні елементи

- Шапка сайту
- Меню
- Блок фільтрації

– Блок авторизації

#### 2.2.2.4. Шапка сайту

Шапка сайту повинна містити логотип і назву сайту. Логотип є посиланням на головну сторінку сайту.

#### 2.2.2.5. Меню

Меню повинне розташовуватися у лівій частині вікна у вигляді окремого блоку і містити посилання на всі розділи сайту.

#### 2.2.2.6. Основне поле контенту

Основне поле контенту повинне розташовуватися в центрі сторінки. У цьому полі відображаються відгуки користувачів. Стильове оформлення матеріалів і їх елементів (посилань, заголовків, основного тексту, зображень, форм, таблиць і т.п.) повинне бути єдиним для всього веб-сайту.

## 2.3. Вимоги до видів забезпечення

### 2.3.1. Вимоги до інформаційного забезпечення

Реалізація сайту відбувається з використанням:

- Node.js
- MongoDB
- React.js

### 2.3.2. Вимоги до лінгвістичного забезпечення

На першому етапі розробки сайт повинен мати лише російську локалізацію

### 2.3.3. Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

– Веб-браузер: Internet Explorer 7.0 і вище, або Firefox 3.5 і вище, або Opera 9.5 і вище, або Safari 3.2.1 і вище, або Chrome 2 і вище;

– Включена підтримка javascript, Flash і cookies.

### 2.3.4. Вимоги до апаратного забезпечення

Апаратне забезпечення клієнтської частини повинне забезпечувати підтримку програмного забезпечення клієнтської частини, зазначеного в п. 2.2.3.

# 3 Склад і зміст робіт зі створення сайту

Докладний опис етапів роботи зі створення сайту наведено в табл. 1.

Таблиця А.1 – Етапи створення сайту

| № | Склад і зміст робіт | Строк розробки (у робочих днях) |
|---|---|---|
| 1 | Розробка каркасу:<br>Проектування розмітки та наповнення сайту | 3 дні |
| 2 | Авторизація:<br>Розроблення та реалізація блоку Авторизації на веб-сайті | 4 дні |
| 3 | Фільтрація:<br>Розробка модулю фільтрації відгуків та сто за рейтингом, датою створення (для відгуків) та містом (для автосервісів) | 5 днів |
| 4 | Правила створення відгуків:<br>Розробка сторінки з правилами | 2 дні |
| 5 | Сторінки створення автосервісів та відгуків:<br>Форми з необхідними полями для створення відгуків\сторінок автосервісів | 5 днів |
| 6 | Модуль пошуку:<br>Реалізація пошуку данних за ключовим словом | 2 дні |
| 7 | Сторінка зворотного зв’язку:<br>Форма для відправки письма розробнику сайту | 2 дні |
| 8 | Завершення роботи:<br>Проведення стилістичних виправлень веб-сайту, перевірка (тестування) реалізованого функціоналу | 3 дні |
| | Загальна тривалість робіт (з урахуванням резервного строку на налагодження й виправлення помилок) і строк закінчення проекту | 25 |

# 4 Вимоги до складу й змісту робіт із введення сайту в експлуатацію

Для створення умов функціонування, при яких гарантується відповідність створюваного сайту вимогам сьогодення ТЗ і можливість його ефективної роботи, в організації Замовника повинен бути проведений певний комплекс заходів.

Для переносу сайту на хостинг необхідно, щоб параметри хостинга відповідали вимогам, зазначеним у ТЗ. На хостинг переноситься програма (сайт), зверстаний шаблон дизайну й структура й наповнення бази даних з подальшою їх доробкою.

# ДОДАТОК Б
# ПЛАНУВАННЯ РОБІТ

1 Планування змісту робіт

Структурна декомпозиція робіт (work breakdown structure, WBS) - це ієрархічна структура робіт, побудована з метою логічного розподілу усіх робіт з виконання проекту і подана у графічному вигляді. Це сукупність декількох рівнів, кожний з яких формується в результаті розподілу роботи поперед-нього рівня на її складові. Елементом найнижчого рівня є група робіт, або так званий робочий пакет (work package).

WBS представлена на рисунку Б.1.

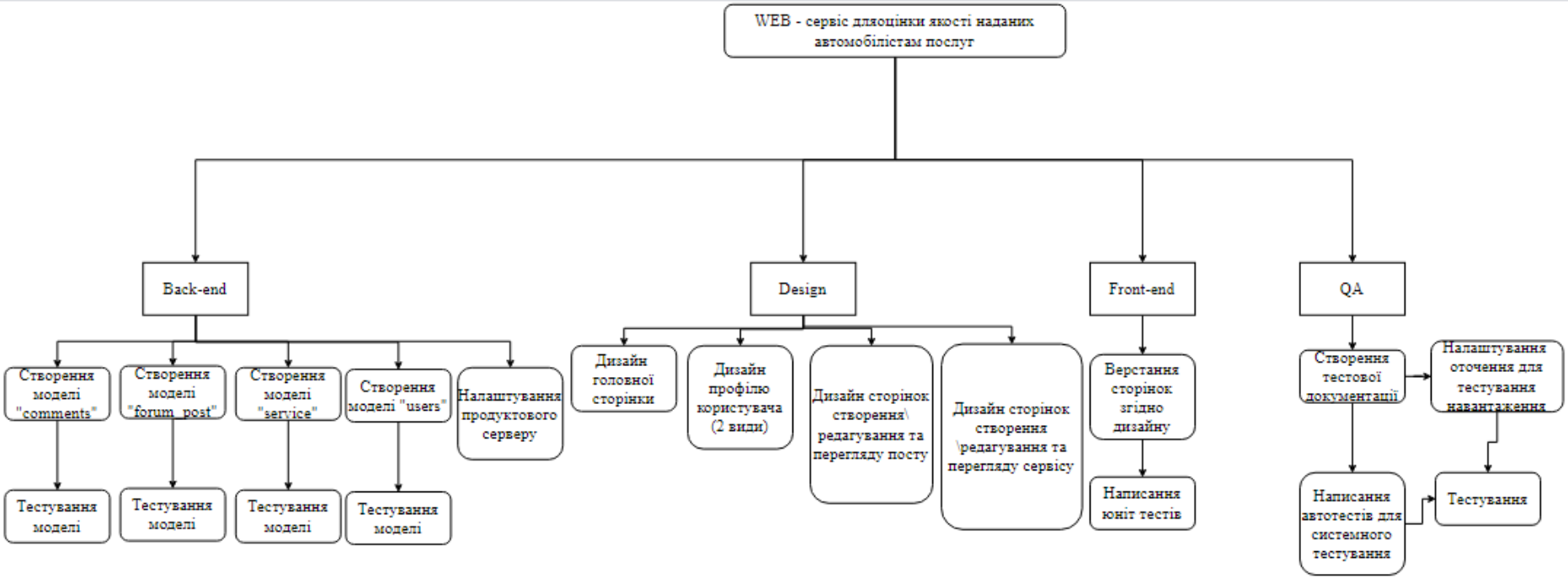


Рисунок Б.1 – WBS структура проекту

2 Планування структури виконавців

Наступним кроком розробки структури проекту є визначення організаційної структури (OBS) проекту.

Організаційна структура проекту (OBS) – є графічним відображенням учасників проекту (фізичних та юридичних осіб) та їхніх відповідальних осіб, залучених до реалізації проекту. На верхньому рівні OBS проекту знаходиться керівник та команда управління проектом; на наступному рівні – виконавці. Останнім рівнем OBS-структури є відповідальні особи виконавців. Це не обов'язково повинні бути керівники, а ті співробітники, яким доручено безпосередньо організовувати і відповідати перед виконавцем за виконання конкретного елемента WBS-структури.

OBS структура представлена на рисунку Б.2.

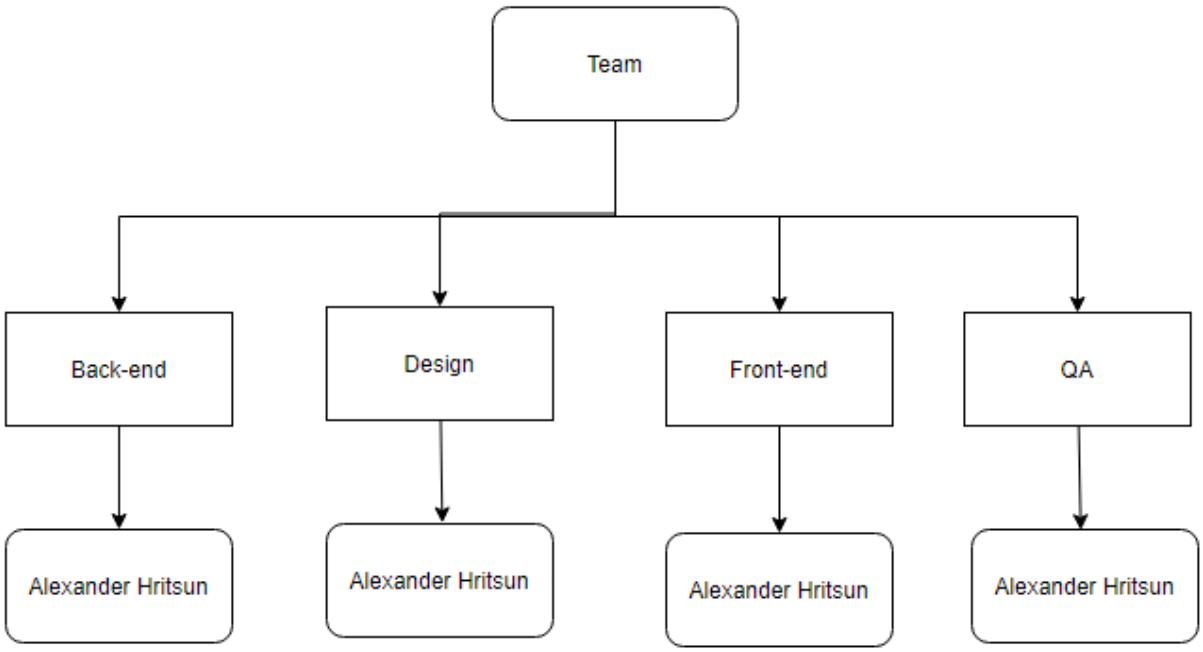


Рисунок Б.2 – OBS структура

3 Побудова матриці відповідальності

Матриця відповідальності (Responsibility Assignment Matrix) забезпечує опис і узгодження структури відповідальності за виконання пакетів робіт. Вона являє собою форму опису розподілу відповідальності за реалізацію робіт проекту із зазначенням ролі кожного з виконавців. Будується на основі WBS та OBS.

У таблиці Б.2 показано матрицю відповідальності проекту.

Таблиця Б.2 – RAM

| № | Duty | Гріцун Олександр | Back dev | Front dev | Designer |
|---|---|---|---|---|---|
| 1 | Створення моделі "comments" | C | R | C | |
| 2 | Створення моделі "forum_post" | C | R | C | |
| 3 | Створення моделі "service" | C | R | C | |
| 4 | Створення моделі "users" | C | R | C | |
| 5 | Дизайн головної сторінки | A | | C | R |
| 6 | Дизайн профілю користувачів | A | | C | R |
| 7 | Дизайн сторінок створення\редагування та перегляду посту | A | | C | R |
| 8 | Дизайн сторінок створення\редагування та перегляду сервісу | A | | C | R |
| 9 | Верстання сторінок згідно дизайну | A | I | R | |
| 10 | Створення тестової документації | R | I | I | |
| 11 | Написання автотестів | R | | | |
| 12 | Налаштування оточення для тестування навантаження | R | C | | |
| 13 | Системне тестування | R | I | I | |
| 14 | Баг фікс+ретест | R | R | R | |

# 4 Розробка PDM мережі

Мережеве планування - метод, при якому використовується графічне моделювання планованого комплексу виконуваних робіт, що відображає їх логічну послідовність, існуючу взаємозв'язок і плановану тривалість

PDM-мережі складаються з двох типів елементів: робіт, які розташовані у вузлах, та стрілок, які вказують логічні взаємозв’язки між роботами проекту

PDM-мережі представлені на рисунках Б.3

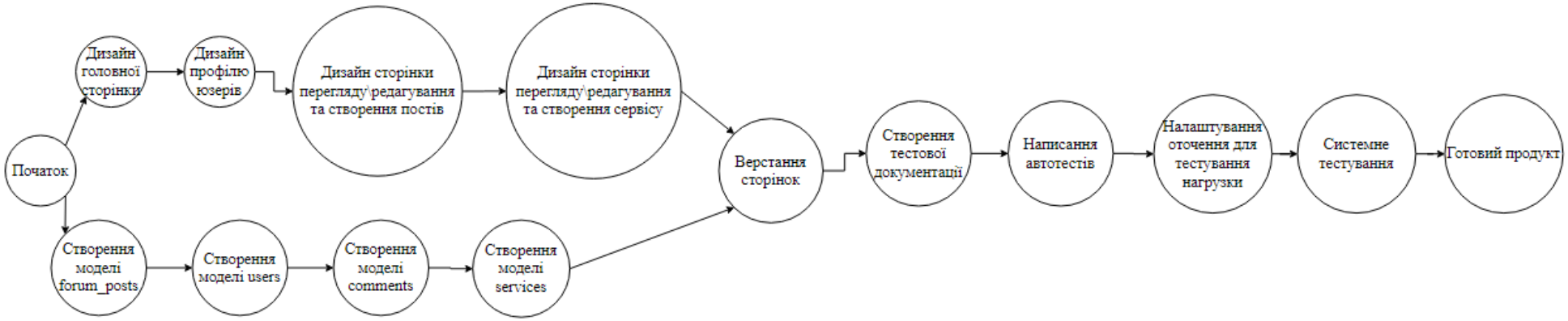


Рисунок Б.3 – Загальний вид PDM-мережі

# 5 Побудова календарного графіку

Діаграма Ганта - це популярний вид діаграми (придуманий Генрі Гант), який використовується для планування і контролю виконання проекту. Такий інтерактивний мережевий графік присутній практично у всіх системах управління проектами.

На діаграмі відображаються завдання і стадії проекту з урахуванням їх часу виконання. Завдання на діаграмі можуть бути залежними один від одного (наприклад, одна задача може починатися тільки після завершення другої). Крім того, може показуватися відсоток виконання кожного завдання і відповідальний за її виконання.

Діаграма Ганта представлена на рисунку Б.5

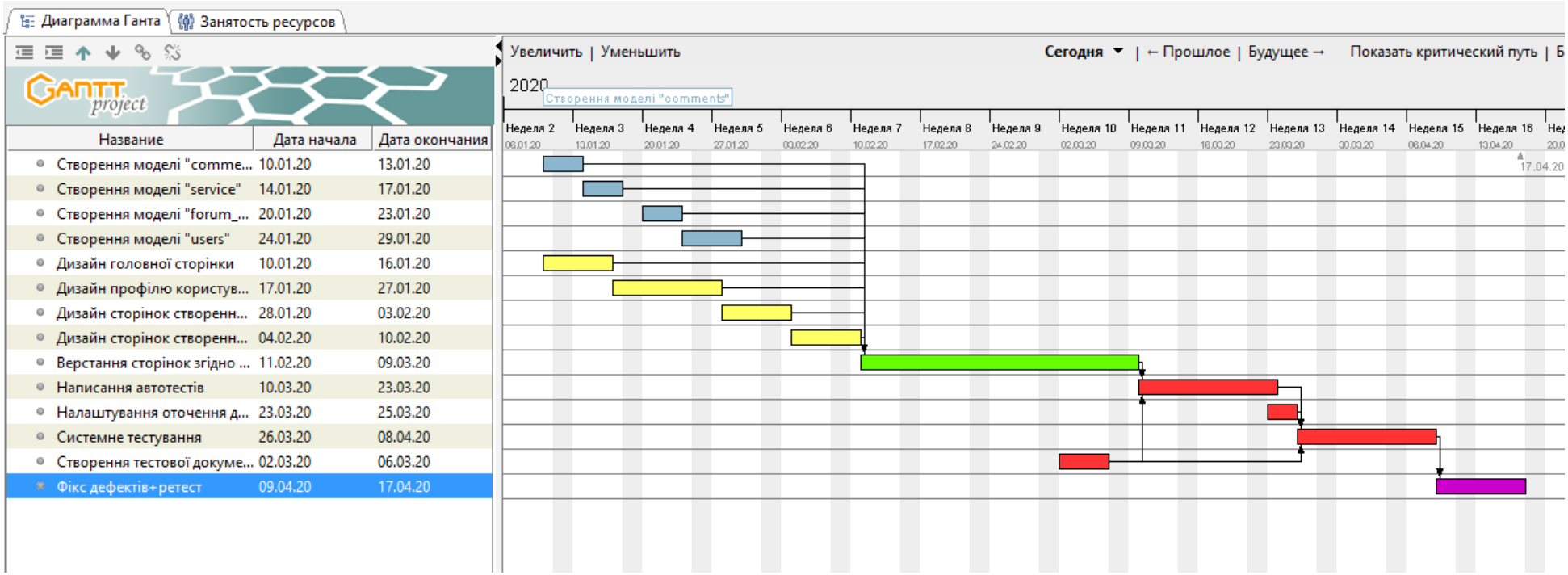


Рисунок Б.5 – Діаграма Ганта

## 2.6 Управління ризиками проекту

Ризик – це імовірнісна подія, яка у випадку своєї появи негативно або позитивно вплине на проект.

Ризики проекту представлені у таблиці Б.4.

Таблиця Б.4 – Risk Register

| № | Опис ризику | Вплив | Вірогідність | RV | Рішення |
|---|---|---|---|---|---|
| 1 | Хвороби працівників | 4 | 2 | M | Призупинити виконання задач до одужання |
| 2 | Фронт енд частина не буде виконана вчасно | 4 | 2 | M | Призупинити виконання залежних від фронтенду задач |
| 3 | Дизайн не буде виконаний вчасно | 5 | 1 | M | Призупинити виконання залежних від дизайну задач |
| 4 | Написання беку затримається | 5 | 3 | H | Призупинити виконання залежних від бек енду задач |
| 5 | Важкі задачі | 5 | 2 | M | Звільнити та замінити працівника |
| 6 | Зміна вимог | 5 | 1 | M | Оцінити зміни, виконати нові задачі |
| 7 | Припинено фінансування | 4 | 2 | M | Шукати нового інвестора, надихати працівників працювати заради ідеї, що змінить світ |
| 8 | Працівник звільнився | 4 | 2 | M | Замінити новим працівником |

Таблиця Б.5 – Probability / Impact Matrix

| | | | | | |
|---|---|---|---|---|---|
| 5 | | | | | |
| 4 | | | | | |
| 3 | | | | | 1 |
| 2 | | | | 4 | 1 |
| 1 | | | | | 2 |
| Probability / Impact | 1 | 2 | 3 | 4 | 5 |

# ДОДАТОК В
# КОДИ ОСНОВНИХ МОДУЛІВ

## User.js

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    email: {type: String, required: true, unique: true},
    name:{type: String, required: true, unique: true},
    password: {type: String, required: true},
    role: {type: String, default: 'user'},
    posts: [{type: Types.ObjectId, ref: 'Post'}]
})

module.exports = model('User', schema)
```

## Service.js

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    name: {type: String, required: true},
    description: {type: String, required:true},
    rating: Number,
    file: String,
    reviews: {
        type: Schema.Types.ObjectId,
        ref: 'Review'
    }
})

module.exports = model('Service', schema)
```

## Review.js

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    text: {type: String, required:true},
    createdAt: {
        type: Date,
        default: Date.now
    },
    updatedAt: {
        type: Date,
        default: Date.now
    },
    rating:{type: Number, required: true},
    userId: {
        type: Schema.Types.ObjectId,
```

```
        ref: 'User'
    }
})

module.exports = model('Review', schema)
```

## Post.js

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    title: {type: String, required: true},
    text: {type: String, required:true},
    file: String,
    createdAt: {
        type: Date,
        default: Date.now()
    },
    updatedAt: {
        type: Date,
        default: Date.now
    },
    author: {
        type: Types.ObjectId,
        ref: 'User'
    }
})

module.exports = model('Post', schema)
```

## Comment.js

```
const {Schema, model, Types} = require('mongoose');

const schema = new Schema({
    text: {type: String, required:true},
    createdAt:{
        type: Date,
        default: Date.now
    },
    updatedAt: {
        type: Date,
        default: Date.now
    },
    userId: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    }
})

module.exports = model('Comment', schema)
```

## auth.routes.js

```
const {Router} = require('express')
const User = require('../models/User')
const bcrypt = require('bcryptjs')
```

```
const jwt = require('jsonwebtoken')
const config = require('config')
const {check, validationResult} = require('express-validator')
const router = Router()

router.post('/registration',
    [
        check('email', 'Email невалидный').isEmail(),

        check('password', 'Минимальная длина пароля 6 символов')
            .isLength({min: 6}),

        check('repeat_password')
            .custom((value, {req}) => {
                if(value !== req.body.password) {
                    throw new Error('Пароли не совпадают')
                }
                return true
            }),

        check('name')
            .custom(async (value) => {
                try{
                    if (!value) {
                        return Promise.reject('Никнейм обязательное поле');
                    }
                    const user = await User.findOne({name: value})
                    if(user){
                        return Promise.reject('Пользователь с таким никнеймом
уже существует')
                    }
                } catch (e) {
                    console.log(e)
                }
            })
    ]
    , async (req, res) => {
    try {
        const errors = validationResult(req)
        if(!errors.isEmpty()) {
            const errorMessages = errors.array().map(el => el.msg)
            console.log(errorMessages)
            return res.status(400).json({
                errors: errors.array(),
                message: errorMessages
            })
        }

        const {email, password, name} = req.body;

        const candidate = await User.findOne({email});

        if(candidate) {
            return await res.status(400).json({message: 'Такой пользователь
уже существует'})
        }

        const hashedPassword = await bcrypt.hash(password, 15);
        const user = new User({email, name, password: hashedPassword});
        await user.save();
```

```
            await res.status(201).json({message: 'Пользователь успешно создан'});


        } catch (e) {
            await res.status(500).json({ message: 'Что-то пошло не так...'})
        }
    })

    router.post('/login',
        [
            check('email', 'Введите корректный email')
                .normalizeEmail().isEmail(),
            check('password', 'Введите пароль')
                .exists()
        ],
        async (req, res) => {
        try {
            const errors = validationResult(req)
            if(!errors.isEmpty()) {
                const errorMessages = errors.array().map(el => el.msg)
                return res.status(400).json({
                    errors: errors.array(),
                    message: errorMessages
                })
            }

            const {email, password} = req.body;

            const user = await User.findOne({email});

            if(!user) {
                return   await   res.status(400).json({message:   'Користувач   не
найден'})
            }
            const isMatch = await bcrypt.compare(password, user.password);

            if(!isMatch) {
                return   res.status(400).json({message:   'Не   верный   пароль   или
email'});
            }

            const token = jwt.sign(
                {userId: user.id},
                config.get('jwtSecret'),
                {expiresIn: '1h'}
            )

            await res.json({token, userId: user.id})



        } catch (e) {
            await res.status(500).json({ message: 'Что-то пошло не так...'})
        }
    })

    module.exports = router
```

## App.css

```
.App {
  text-align: center;
}

body {
  background-color: #d6d6d6;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

.asr-nav.navbar.navbar-expand.navbar-dark.bg-dark {
  background-color: #B71C1C !important;
}

.search-wrapper {
  padding-left: 30px;
  width: 100%;
  height: 45px;
  background-color: #B71C1C;
}
```

```
.asr-input {
  background-color: transparent;
  border-color: transparent;
  border-bottom-color: white;
  border-radius: 0;
}

.asr-input:focus {
  background-color: transparent !important;
  border-color: transparent;
  border-bottom-color: white;
}

.asr-input::placeholder {
  color: white;
}

.asr-sidebar {
  margin-top: 101px;
  background-color: #B71C1C !important;
}

.mr-s {
  margin-right: 10px;
}

.mr-m {
  margin-right: 15px;
}

.w100 {
  width: 100% !important;
}

.form {
  padding-left: 35%;
  padding-right: 35%;
  text-align: left;
  margin-top: 30px;
}

.search-icon {
  position: absolute;
  margin-left: 5px;
  background-color: #B71C1C;
  height: 35px;
}

.user {
  display: flex;
  flex-direction: row;
  justify-content: left;
  margin-top: 30px;
  margin-left: 30px;
  text-align: left;
}

.post {
  margin: 35px 30% 30px;
  overflow: hidden;
}
```

```
.post-text {
  overflow: hidden;
}

.alert {
  padding: .5rem 1rem;
  border-radius: 10px;
  text-align: center;
  margin-bottom: 1rem;
  background: rgba(255, 0, 0, .3);
  color: red;
}
```

## app.js

```
const express = require('express');
const app = express();
const mongoose = require('mongoose');
const config = require('config');

app.use(express.json());

app.use('/api/auth', require('./routes/auth.routes'))
app.use('/api/posts', require('./routes/posts.routes'))
app.use('/api/users', require('./routes/users.routes'))

const PORT = config.get('port') || 5000;

async function start() {
    try {
        await mongoose.connect(config.get('mongoUri'), {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true
        })
        app.listen(PORT, () => {
            console.log(App has been started on PORT: ${PORT}...);
        })
    } catch (e) {
        console.log('Server error', e);
        process.exit(1)
    }
}
start()
```