МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка веб-ресурсу для лізингу автотранспорту на базі фреймворку Node.js

Виконав студент групи К-18
спеціальності 122 «Комп'ютерні науки»
Курбанов Агамират _____

Керівник к.геогр.н., доцент _____
Кузніченко Світлана Дмитрівна ___

Консультант _____
_____

Рецензент к.техн.н., доцент _____
Гнатовська Ганна Арнольдівна ___

Одеса 2022

ЗМІСТ

Скорочення та умовні познаки 6
Вступ 7
1 Актуальність та порівняльний аналіз 9
1.1 Аналіз предметної області 9
1.2 Аналіз існуючих рішень та аналогів 10
1.2.1 Система «AVIS» 10
1.2.2 Система «AutoStar» 11
1.2.3 Система «Hapai» 11
1.2.4 Система «ULF» 12
1.2.5 Висновки щодо порівняння аналогів 12
1.3 Вибір архітектурного шаблону 13
1.4 Вибір мови програмування 15
1.4.1 Вибір мови програмування для клієнтської частини 15
1.4.2 Вибір мови програмування для серверної частини 16
1.5 Вибір середовища програмування 18
1.6 Вибір технологій для розробки інформаційної системи 20
1.6.1 Технології для клієнтської частини 21
1.6.2 Технології для серверної частини 24
1.7 Вибір бази даних 27
1.8 Висновки до першого розділу 29
2 Проектування інформаційної системи 30
2.1 Мета і задачі інформаційної системи 30
2.2 Типи користувачів 31
2.3 Представлення інтерфейсу користувача (UI View) 32
2.4 Логічне уявлення ІС (Logical View) 40
2.5 Уявлення розгортання ІС (Deployment View) 41
2.6 Представлення даних ІС (Data View) 42

2.7 Висновки до другого розділу 44
3 Реалізація інформаційної системи 46
3.1 Уявлення про структуру проекту 46
3.2 Уявлення про класи ІС 51
Висновки 57
Перелік джерел посилання 59
Додаток А Керівництво адміністратора ІС 62
Додаток Б Код програми 63

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

API – Application Programming Interface – програмний інтерфейс програми

IDE – Integrated Development Environment – інтегроване середовище розробки

SDK – Software Development Kit – набір засобів розробки

ВСТУП

З ростом міст та чисельності населення виникає потреба мати налагоджену галузь транспортної інфраструктури, яка надає такі функції, як переміщення людей і товарів.

Такі тенденції зумовлюють кожного обзавестися особистим транспортом або ж користуватися громадським. Однак є недоліки цих двох варіантів. У першому випадку прийдеться витратити кошти для придбання, займатися постійним обслуговуванням і ремонтом. У другому – через перевантаженість витрачати більше часу, будувати складні маршрути, а також обмежувати себе в комфорті. В якості альтернативи виступають майданчики з лізингу автотранспорту, які можуть надати широкий спектр послуг та пропозицій для клієнта.

Ці майданчики дуже популярні в Америці та Європі, так як надають людям можливість розділити на тимчасові етапи покупку авто або просто скористатися довгостроковою орендою. До того ж, для малого бізнесу дуже зручно на початкових етапах не вкладати великі кошти на транспорт.

Тому виникає потреба в інформаційних системах, за допомогою яких клієнт зможе ознайомитися з усіма пропозиціями цих майданчиків для свого міста. Швидко розрахувати термін і приблизну вартість лізингу або ж отримати консультацію у фахівців.

Таким чином, розробка програмного продукту допоможе вирішити проблему швидкого і зручного лізингу автівок для особистих потреб через веб-сервіс. Це підтверджує актуальність вибору теми для кваліфікаційної роботи.

Метою кваліфікаційної роботи є розробка веб-ресурсу для лізингу автотранспорту на базі фреймворку Node.js.

Для розробки онлайн-ресурсу необхідно виконати наступні задачі:

– проаналізувати аналогічні системи для взаємодії користувачів і майданчиків з прокату автотранспорту і виявити функціонал;

– вибір інструментарію для проектування та реалізації системи;

– проектування складових частин системи;

– здійснити реалізацію кожної частини системи;

– тестування кожної частини системи окремо, а також всієї інформаційної системи в цілому.

Структура кваліфікаційної роботи складається з вступу, трьох розділів, висновків, переліку посилань на 27 найменування. Повний обсяг роботи становить 70 сторінок і містить 26 рисунків та 3 таблиці.

1 АКТУАЛЬНІСТЬ ТА ПОРВНЯЛЬНИЙ АНАЛІЗ

1.1 Аналіз предметної області

Створення налагодженої інфраструктури сприяє високому розвитку міст і країн в цілому. Однією з таких інфраструктур є транспортна інфраструктура, яка контролює всі галузі і підприємства транспорту. Вона включає в себе безліч окремих ознак, проте в умовах даної роботи розглянемо саме автомобільний транспорт.

Як було описано вище, в умовах зростання міст виникає гостра необхідність у використанні транспорту для пересування. Людина стикається з такими ситуаціями, як: купувати власний транспорт, скористатися громадським або таксі. Кожен з цих варіантів може привести до певних незручностей.

В якості альтернативи можна виділити систему з лізингу автотранспорту, яка також допомагає вирішити проблему перевезення людей і товарів. Однак на сучасному українському ринку представлено не так багато систем, які б могли надати великий асортимент пропозицій, заточений під різні потреби. При цьому не кожна система має охоплення в кілька населених пунктів.

З урахуванням даних обставин, актуальною є тема створення онлайн-ресурсу по лізингу автотранспорту для густонаселених міст України. Це дасть можливість людям або малому бізнесу швидко знайти підходящу площадку у своєму місті, де можна отримати потрібний транспорт. Вибір буде стояти між трьома категоріями транспорту:

- легкові автомобілі;
- мототехніка;
- вантажівки.

Таким чином, основною ідеєю комплексної кваліфікаційної роботи є створення онлайн-ресурсу для лізингу автотранспорту різної специфікації. Пошук варіантів проводиться виходячи з вибору міста. Для кожного міста буде сформований свій список пропозицій. Також дана система дозволить швидко обробляти запити користувачів і настроїти зв'язок з майданчиками.

1.2 Аналіз існуючих рішень та аналогів

Насамперед потрібно провести порівняльний аналіз з існуючими аналогами, які вже представлені на ринку. Це дозволить визначити необхідність удосконалення даних систем, а також сформувати функціонал, який доцільно запозичити.

Були знайдені системи, через які можна орендувати автівки:

– AVIS;
– AutoStar;
– Hapai;
– ULF.

Далі проведемо опис кожної системи з визначенням основних можливостей та характерних ознак.

1.2.1 Система «AVIS»

Система «AVIS» [1] дозволяє брати в лізинг автомобілі різних класів у таких містах як: Київ, Дніпро та Ужгород.

Далі будуть описані пункти, які характеризують можливості та недоліки.

Можливості:

– даний сервіс надає широкий спектр пропозицій. Присутні варіанти різного типу лізингу (фінансового і оперативного).

Недоліки:

– пропонуються лише легкові автомобілі, немає можливості взяти в лізинг вантажівку або мототехніку;
– послугою можна скористатися лише трьох містах;
– відсутня можливість прямо в системі розрахувати щомісячну вартість і початковий внесок.

Як можна бачити з вищенаведених характеристик, дана система має не дуже зручний функціонал, обмеженість по населеним пунктам. Також немає пропозицій для перевезення великих вантажів, і відсутні пропозиції щодо мототехніки.

1.2.2 Система «AutoStar»

Система «AutoStar» [2] займається лізингом і продажом автомобілів в Києві та Черкасах.

Вона може похвалитися такими функціональними можливостями:

– вона також в себе включає продаж автотранспорту;

– важливою особливістю є окремий фільтр по підбору варіантів з США та Європи;

Серед можливих недосконалостей сервісу, крім як спеціалізація лише на легкових авто, можна також виділити наступні мінуси:

– немає можливості прив’язати до системи особистий аккаунт, який би дозволив зберігати історію;

– не надто значне охоплення;

– передбачає використання лізингу лише з подальшим викупом.

1.2.3 Система «Нараі»

Система «Нараі» [3] вже має значне охоплення по містах України. До того ж вона співпрацює з майданчиками з продажу автомобілів.

Далі будуть описані пункти, які характеризують можливості та недоліки.

Можливості:

– має великий розмах в автоіндустрії, тому автопарк налічує велику кількість пропозицій різних класів;

– прямо в сервісі можна розрахувати вартість для заданого періоду.

Недоліки:

– відсутня можливість оперативного лізингу, тому потрібно заключати договори для подальшого викупу;

Система передбачає величезний запас пропозицій, до того ж є спеціалізовані майданчики для цього. Однак взяти в лізинг авто на короткий термін у користувача не вийде.

1.2.4 Система «ULF»

В сервісі «ULF» [4] вже присутнє розбиття на категорії, тобто з'являється можливість взяти в лізинг вантажівки, мотоцикли та інше. Також зберігається історія замовлень.

З мінусів можна виділити наступне:

– система майже не надає власних пропозицій. Або посилається на сторонній майданчик, або змушує користувача шукати потрібний варіант у іншій системі;

– немає можливості взяти автотранспорт у недовгостроковий лізинг відразу, тільки через посередників.

1.2.5 Висновки щодо порівняння аналогів

У підсумку, після проведення аналізу аналогів інформаційної системи, були сформульовані основні вимоги для створення онлайн-ресурсу з лізингу автотранспорту. Даний сервіс повинен спростити взаємодію між клієнтами і лізинговою компанією, а також вирішити проблему довгострокової оренди автотранспорту з можливістю подальшого викупу або поверненню.

Отриманні вимоги були занесені у таблицю. У табл. 1.1 можна побачити порівняння інформаційної системи, що розробляється с існуючими на ри-

нку аналогами. Ця ж таблиця демонструє необхідність удосконалення розглянутих у пункті 1.2 аналогів.

Таблиця 1.1 – Порівняння аналогів

Назва функціоналу	«AVIS»	«AutoStar»	«Hapai»	«ULF»	Веб-ресурс, що розробляється
Підтримка усіх типів лізингу	Так	Ні	Ні	Так	Так
Розрахунок коштів	Ні	Так	Так	Так	Так
Зберігання історії користувача	Так	Ні	Ні	Так	Так
Розподіл автотранспорту по категоріям	Ні	Ні	Ні	Так	Так
Особистий автопарк	Так	Так	Так	Ні	Так
Охоплення усіх великих міст	Ні	Ні	Так	Так	Так
Можливість обробити власну пропозицію	Так	Ні	Так	Так	Так
Можливість стежити за станом балансу	Ні	Ні	Ні	Ні	Так

Отже, основними недоліками систем було або відсутність пропозицій для малого бізнесу (лізинг вантажівок та спецтехніки), або обмеженість по обслуговуванню міст. При формуванні вимог одним з важливих критеріїв було підтримати початковий та малий бізнес шляхом розширеного каталогу автотранспорту (від мототехніки до великих вантажівок). До того ж і звичайний споживач зможе знайти для себе кращу пропозицію за рахунок різних умов для лізингу.

1.3 Вибір архітектурного шаблону

Так як система, яка розробляється, передбачає використання якогось інтерфейсу для клієнта і обробника запитів для надавання даних, то в якості

архітектурного шаблону ідеально підійде клієнт-серверна архітектура [5]. Більш докладно опишемо застосування даної архітектури.

В якості клієнта буде виступати веб-браузер. Цей вибір обумовлений тим, що веб-браузер найкраще підійде для відображення всього функціоналу системи і для легкої взаємодії з ним. Він буде виконувати такі функції, як:

– надавання користувальницького інтерфейсу;
– формулювання запиту до сервера і його відправка;
– отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

В якості основного сервера буде виступати сервер додатків, який в свою чергу розміститься на хмарної PaaS-платформі. Більшість серверів додатків мають в своєму складі веб-сервер, значить він може робити все, на що здатний веб-сервер. Безкоштовний хостинг був вибраний з рахунку забезпечення вільного доступу до основної бізнес-логіки за допомогою інтернету. Даний сервер також дозволяє приймати HTTP-запити від клієнтів і видавати їм HTTP-відповіді. Всі передані дані будуть перетворені у формат JSON (JavaScript Object Notation).

Для коректної роботи сервера з базою даних буде використовуватися технологія ORM (Object-Relational Mapping). Вона дозволяє пов'язувати бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

Таким чином, сервер додатків буде виконувати наступні функції:

– обробка клієнтського запиту;
– робити запити до бази даних для формування відповіді;
– відправка результату (відповіді) клієнту.

Структура обраного для системи архітектурного шаблону представлена на рисунку 1.1. В даному випадку буде використовуватися архітектура, яка складається з трьох рівнів.

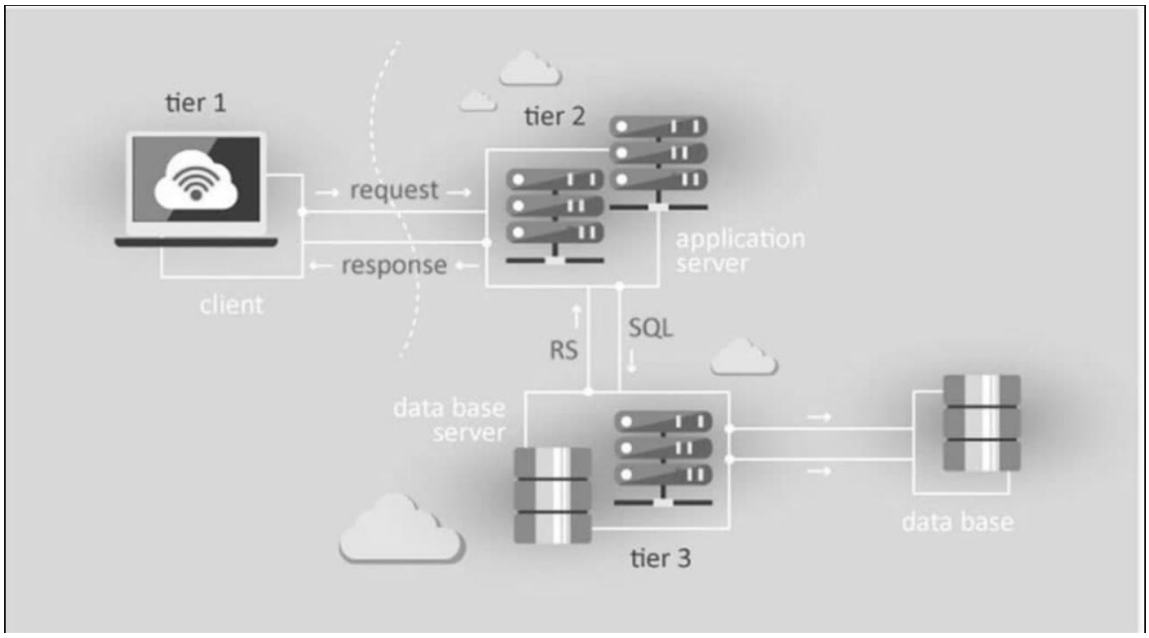

Рисунок 1.1 – Трирівнена архітектура клієнт-серверної системи

1.4 Вибір мови програмування

1.4.1 Вибір мови програмування для клієнтської частини

Виходячи с того, що інтерфейс системи буде розроблятися для підтримки в веб-браузерах, тому в якості мови програмування був обраний JavaScript [6].

JavaScript – це мова програмування, яка є прототип-орієнтованою. Він відображає мову ECMAScript, чиїм прототипом спочатку і був. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додавання інтерактивності веб-сторінкам. JavaScript – офіційно зареєстрована торгова марка компанії Oracle.

Дана мова підтримується усіма основними браузерами і включена за замовчуванням. Наприклад, в браузері JavaScript може:

- додавати новий HTML-код на сторінку;
- змінювати існуючий вміст;
- модифікувати стилі;

- реагувати на дії користувача (клацання миші, натискання клавіш);
- відправляти мережеві запити на віддалені сервера;
- отримувати і встановлювати куки, показувати повідомлення;
- запам’ятовувати дані на стороні клієнта.

Код JavaScript зазвичай або вбудований в сторінку браузера, або посилається на окремий файл .js. Це мова, яка працює на стороні клієнта, що означає, що скрипт закачується на комп’ютер відвідувача і обробляється там.

На сьогоднішній день існує чимало модифікацій мови JavaScript. Найпоширеніші з них: CoffeeScript, TypeScript і JSX. Фреймворк, який передбачає розробку на TypeScript – це Angular, а для CoffeeScript – фреймворк Ruby on Rails.

Оскільки у подальшому буде використовуватися фреймворк React, тому для створення клієнтської частини використовуватиметься модифікована версія JSX, який також називають синтаксичним "цукром" мови JavaScript [7].

1.4.2 Вибір мови програмування для серверної частини

Серверні мови програмування потрібні для реалізації бізнес-логіки, тобто розробник за допомогою мови програмування описує можливі сценарії використання сайту або додатку.

Мов програмування, використовуваних для серверної веб-розробки, досить багато. Однак в рамках даної роботи проведемо порівняння мов: PHP, Java та JavaScript [8].

PHP використовують близько 78% всіх сайтів. Мова з’явилася в 1995 році, коли було не так багато можливостей для створення динамічних веб-сторінок. Він динамічний типізований, і один і той же фрагмент коду може вести себе по-різному в залежності від контексту, що робить програми на PHP складними для масштабування і іноді повільними.

PHP – гарна мова для розробки проектів на CMS. Але екосистема мови і особливості кадрового ринку роблять цю мову не найкращим вибором для розробки складних проектів.

Java – одна з найпопулярніших об'єктно-орієнтованих мов програмування. Універсальність даної мови забезпечується віртуальною машиною JVM (Java Virtual Machine). У багатьох мовах під час компіляції переводиться в код, який може працювати по-різному на різних пристроях або платформах. В Java цієї проблеми немає. JVM грає роль проміжного рівня – з програми на Java вона робить код, який може виконуватися на будь-якому комп'ютері незалежно від того, де код був скомпільований.

Компанія «Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензію GNU General Public License

У Java велика спільнота, і в інтернеті можна знайти відповіді практично на будь-які питання про мову.

JavaScript – гнучка мова, яку можна використовувати як для фронтенду, так і для бекенду. Ця мова підходить для тих, хто починає свій шлях у серверної розробці, оскільки в неї мало налаштувань, і можна писати код прямо в браузері.

Це динамічна мова, тому, працюючи на неї, не можна використовувати щось на зразок статичної перевірки типів при компіляції. У JavaScript велика спільнота, і для його вивчення в мережі є багато корисних матеріалів.

Значно розширює можливості JS програмна платформа Node.js. З її допомогою код, написаний на JS, можна запускати без браузера на бекенд. А наявність величезної кількості готових рішень в пакетної екосистемі npm (node packages manager) дозволяє розробнику не витрачати час на створення більшості типових рішень. Застосовуючи JS на клієнті і на сервері, можна створювати веб-додатки, які можуть рендериться і в браузері, і на сервері. При цьому такі системи зазвичай працюють дуже чітко і зрозуміло.

Далі проведемо порівняльний аналіз щодо аспектів, які є дуже важливими при створенні системи для кваліфікаційної роботи (порівняння мов програмування представлено в таблю 1.2).

Таблиця 1.2 – Порівняння мов програмування для серверу

Функціональні можливості	Java	PHP	JavaScript
Легкість налаштування	–	+	+
Підтримка потоків	+	–	+
Неблокуючі введення/виведення	+	–	+
Висока швидкість написання робочого сервера	–	+	+
Наявність зручного вбудованого менеджера пакетів	–	–	+

Так як система передбачає використання тільки HTTP протоколу передачі даних, необхідно розглядати ту мову, яка в достатній мірі орієнтована на веб-додатки. До того ж, дуже зручно, коли клієнтська і серверна частини підтримують одну мову програмування (можна сказати, що мова створена саме для full-stack розробника). Це дозволяє швидко побудувати програмну архітектуру системи, спираючись на принципи однієї мови.

Тому в якості мови програмування для серверу ідеально підходе JavaScript.

1.5 Вибір середовища програмування

Виходячи с того, що для клієнтської і серверної частини буде використовуватися одна мова програмування (JavaScript), то для налагодженої роботи необхідно підібрати сумісну для цих двох частин середу розробки. Потрібно розглянути всі можливі IDE, які дозволяють без проблем виконувати ро-

зробку на JavaScript, а зокрема, з платформою Node.js. Порівняємо такі IDE, як: Visual Studio Code, WebStorm та Brackets.

Visual Studio Code позиціонується як редактор вихідного коду, розроблений Microsoft, Linux і macOS. Він підходить для кросплатформеної розробки веб і хмарних додатків. Це середовище передбачає грамотного налаштування усіх плагінів для повноцінної роботи (починаючи зі звичайних імпорту модулів і закінчуючи технологією автодоповнення). Всі ці маніпуляції доведеться виконати вручну.

Можливості, які можна виділити при роботі з Visual Studio Code:

- підтримка різних мов;
- при правильному налагоджені дана платформа перетворюється у потужну платформу;
- вбудований термінал;
- підтримка GIT.

WebStorm – інтегроване середовище розробки від компанії JetBrains, розроблена на основі платформи InteliJ IDEA. Він знаходить застосування всіх можливостей сучасної JavaScript-екосистеми. Дозволяє налагодити додаток прямо в IDE. До того ж його не потрібно налаштовувати, він уже в собі містить необхідні інструкції для зручної розробки. Можна виділити наступні особливості даного середовища:

- вбудовані тестування і трасування;
- підтримка Node.js;
- вбудований термінал;
- підтримка систем контролю;
- пропонує надійну навігацію та рефакторинг коду.

Brackets – вільний текстовий редактор для веб-розробників, який орієнтований на роботу з HTML, CSS та JavaScript. Сумісний з операційними системами macOS, Windows і Linux. Brackets створений і розвивається Adobe Systems, тому його часто використовують для творчих розробок. Однак він

також потребує налагодження – установки різних плагінів. Виділимо основні можливості цього редактору:

- підтримка препроцесора;
- підтримка Node.js;
- забезпечує попередній перегляд.

Проведемо порівняння цих платформ в табл. 1.3.

Таблиця 1.3 – Порівняння середовищ програмування

Можливості	WebStorm	Visual Studio Code	Brackets
Можливість тестування і трасування	+	–	–
Вбудований командний рядок	+	+	+
Плагіни з екосистеми та локальної історії	+	–	+
Інтеграція з іншими редакторами	+	+	–
Вбудований контроль версіями	+	+	+

Таким чином, ідеальним вибором буде використання саме середовища розробки WebStorm, завдяки великому вбудованому функціоналу.

1.6 Вибір технологій для розробки інформаційної системи

Наступним важливим етапом при проектуванні інформаційної системи є вибір стека технологій. Його вибір обумовлений тим, які саме функціональні задачі потрібно вирішити. Також він служить для того, щоб полегшити сам процес розробки системи.

1.6.1 Технології для клієнтської частини

Почнемо з пошуку технології для клієнтської частини системи (він же front для веб-додатків). Основна частина сучасних frontend фреймворків концептуально побудовані з використанням мови програмування JavaScript. Розглянемо найпопулярніші з них, а саме: Vue.js, Angular, Ember.js та React.js [9].

Vue.js – це JavaScript-фреймворк, заснований у 2013 році, який ідеально підходить для створення складних односторінкових додатків і високоадаптованих користувальницьких інтерфейсів. Підтримкою даного фреймворку займається відносно невелика спільнота, тому для вирішення проблем може знадобитися більше часу. Отже, можна виділити такі основні особливості даної технології:

- двостороння реактивна система прив’язки даних;
- крихітний розмір. Vue.js важить близько 20 Кбайт, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти набагато кращої продуктивності в порівнянні з іншими платформами;
- нечіткі архітектурні шаблони;
- компонентний підхід у Vue.js не такий гнучкий і очевидний, як в React.

Таким чином, цю технологію рекомендується використовувати для гнучких конструкцій, що дозволяють створювати все з нуля і успішно розробляти великі проекти. Однак через нестабільність роботи компонентів і відносно невеликим співтовариством, розробка може стати досить складною і незручною.

Angular – це каркас для розробки веб-додатків від Google. Добре підходить для динамічних веб-додатків, з використанням HTML для статичних сторінок.

Майже всі поточні рішення використовують парадигму компонентного підходу, і Angular не виняток. Даний фреймворк написаний на TypeScript, тому для розробки також потрібно використовувати цю мову програмування. Для цієї технології можна виділити наступні особливості:

- наявність CLI (Command Line Interface) системи;
- MVVM (Model-View-ViewModel), яка дозволяє розробникам працювати окремо над одним і тим же розділом програми, використовуючи один і той же набір даних;
- впровадження залежностей від компонетів, пов'язаних з модулями і модульністю в цілому;
- велика масштабованість;
- відносно повільна продуктивність;
- великий обсяг результуючого коду.

Отже, Angular збільшує продуктивність браузерних додатків, шляхом динамічного оновлення вмісту в найкоротші терміни, за допомогою двостороннього зв’язування даних. Він відмінно підходить для створення динамічних веб-додатків.

Однак в плані побудови правильної структури, він найскладніший з існуючих. Ця технологія вимагає великого запасу знань в окремих складових (Injectables, Components, Pipes, Modules та інше). До того ж, для оптимізації і збільшення продуктивності на невеликих і середніх проектах виникає необхідність позбавлятися від зайвих модулів. Тому дана технологія не підходить для системи, яка розроблюється.

Ember.js – це добре структурований JavaScript-фреймворк, що дозволяє розробникам створювати клієнтську частину масштабованих веб-додатків на основі MVC (Model-View-Controller). Виконує першокласну роботу на стороні DOM-рендерингу на стороні сервера. Підтримується сайтами Discourse, Groupon, LinkedIn, Vine.

Для даної технології можна виділити наступні особливості:

- підтримує двосторонню прив'язку даних;

- серверний рендеринг;
- відміна маршрутизація;
- простий в налаштуванні;
- немає стандартного набору UI-елементів;
- багато важить;

Розглядати Ember.js краще в тих випадках, коли розробник вже має непоганий рівень в організації сучасних веб-додатків. Так як обмежена документація може сильно загальмувати процес розробки. Через велику вагу для деяких систем рендеринг на початкових етапах може виявитися досить довгим.

React.js – один з найпростіших у вивченні фреймворк. Він був розроблений в Facebook для виправлення проблем з підтриманням коду через постійне додавання функцій в додаток. Сучасний React з відкритим вихідним кодом відрізняється своєю віртуальною об’єктною моделлю документів (DOM), яка пропонує виняткові функціональні можливості.

Перерахуємо можливості цієї технології:

- стабільна і безперебійна робота з використанням віртуального DOM;
- додаток витримує великі навантаження;
- сумісний з великою кількістю бібліотек;
- невпорядкованість документації;
- можливість створення компонентів без написання класів;
- React реалізує концепції функціонального програмування, створюючи простий в тестуванні і багаторазово використовуваний код;
- перехід між версіями, як правило, дуже простий.

React.js дуже простий у вивченні, так як в основі лежать прості мови програмування. Це найнадійніший frontend фреймворк для розробки інтерактивного інтерфейсу з меншими витратами часу, зважаючи на можливість повторного використання компонентів. Для коректної роботи з сервером допоможе асинхронне програмування, яке дана технологія підтримує. Завдяки концепції Virtual DOM по організації документів у вигляді дерева, це самий

кращий формат для веб-браузера при аналізі компонентів будь-якого веб-додатку.

Проаналізував найпопулярніші технології для створення інтерактивного інтерфейсу системи, який би підтримував сучасні веб-браузери, вибір припав на використання фреймворку React.js. Цей вибір обумовлений також і іншими важливими ознаками для розробки системи, серед яких можна ще виділити:

- сумісність з великою кількістю бібліотек;
- швидка оптимізація рендеринга;
- застосування функціонального програмування.

Для роботи з даним фреймворком нам буде потрібно використовувати деякі бібліотеки. Перерахуємо їх:

- бібліотека Redux призначена для управління станом додатку. Вона містить рід інструментів, що дозволяють значно спростити передачу даних сховища через контекст. Це сильно спрощує масштабованість і дозволяє швидко діагностувати вузькі місця, асинхронні проблеми або помилки;
- бібліотека Axios. Дана бібліотека надає один з найпопулярніших HTTP клієнтів для браузерів. У Axios є підтримка запитів, отримання відповідей від сервера, їх трансформація і автоматична конвертація в JSON. Ще важливим параметром є можливість перехоплювати і скасовувати запити і в клієнті вбудований захист від підробки запитів між сайтами.

1.6.2 Технології для серверної частини

Для створення серверної частини системи проведемо аналіз актуальних на сьогоднішній день технологій з розробки backend-у на мові програмування JavaScript. Серед них можна виділити такі Node.js фреймворки: Meteor, Koa, LoopBack, Express. Існують і інші популярні технології для сервера, проте в рамках даної роботи буде проводитися вибір серед цих чотирьох [10].

Meteor є MVC фреймворком з відкритим вихідним кодом, за допомогою якого ви можете створювати веб-додатки реального часу. Одна з найважливіших особливостей платформи полягає в тому, що вона дозволяє використовувати один і той же код як на стороні сервера, так і на стороні клієнта. Між сервером і клієнтом, як правило, передаються дані, а не HTML-код. Його реактивна модель програмування дозволяє створювати додатки, використовуючи менше JavaScript коду.

Для даної технології можна виділити наступні особливості:

– можливість управління великими проектами;
– велике і організоване співтовариство по розробці документації;
– використання реактивної парадигми програмування;
– використання стека даних Facebook GraphQL.

Даний фреймворк не сильно підходить під концепції системи, тому що він передбачає одночасну роботу з шаблонизаторами для клієнтської частини, хоча для нас важливо саме обробка запитів і взаємодія з базою даних.

Koa – це потужний фреймворк HTTP для Node.js, що дозволяє створювати веб-додатки і API-інтерфейси, більш зручні для розробки. Його стек програмного забезпечення працює в вигляді стека, що дозволяє реалізувати завантаження даних, а потім уточнити і управляти вивантаженням даних. Це включає в себе такі елементи, як узгодження вмісту, нормалізація невідповідності вузлів, перенаправлення і багато іншого.

Виділимо особливості даної технології:

– використання необхідних генераторів для управління і обробки зворотних викликів;
– ефективні процедури по обробці помилок;
– блоки на основі компонентів;
– недостатня підтримка спільноти.

Застосування генераторів ES6 в Koa дозволяє виключити зворотні виклики (callbacks), зменшує для розробників обсяг роботи з кодом і знижує

ймовірність помилок. Однак через відсутність достатньої підтримки спільноти, розробка на цій технології може затягнутися.

LoopBack – це розширюваний фреймворк Node.js, що дозволяє розробникам легко налаштовувати моделі і створювати потужні API REST в найкоротші терміни, які вимагають мінімального кодування. Він підтримує безпечну аутентифікацію і настройки дозволів, а також підтримує додаткові сегменти, численні сховища даних бекенда і спеціальні запити. Також має гарну сумісністю з різними службами REST і типами баз даних, включаючи MySQL, MongoDB, Oracle, Postgres і багато інших.

Виділимо особливості даної технології:

– швидке створення динамічних REST API;
– гкраще з’єднання між різними пристроями і браузерами;
– покращена кореляція між різними типами даних і сервісами;
– працює як в on-promises, так і в хмарі.

Ця технологія включає в себе велику кількість вбудованого функціоналу, який не так просто налаштувати, тому для невеликих проектів ця багатофункціональність може виявитися надмірною.

Express – це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків. Надає тонкий шар фундаментальних можливостей для роботи з функціями Node.js. Він дозволяє використовувати реляційну і нереляційну базу даних, різні модулі для верифікації та форматування даних. Його особливості:

– легкість і швидкість написання коду;
– добре задокументовано;
– хороша масштабованість;
– величезне співтовариство розробників;
– наявність npm (node package manager). Величезна кількість бібліотек, які можуть бути встановлені в один рядок).

Даний фреймворк дозволяє миттєво вирішити безліч завдань, з якими можна зіткнутися при розробці. Маючи в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів, створити надійний API (Application Programming Interface) можна швидко и легко.

Проаналізував найпопулярніші технології для створення серверної частини, які б дозволили швидко та легко налаштувати базовий функціонал серверу, вибір припав на використання фреймворку Express.js.

Для роботи з даним фреймворком нам буде потрібно використовувати деякі бібліотеки. Перерахуємо їх:

– бібліотека Sequelize. Це ORM для СУБД Postgres, MySql, MariaDB, Sqlite і Microsoft Sql Server. Вона здійснює зіставлення таблиць в БД і відносин між ними з класами. При використанні Sequelize ми можемо не писати sql-запити, а працювати з даними як зі звичайними об'єктами;

– бібліотека Faker. Вона дозволяє генерувати великі обсяги фіктивних даних для перевірки роботи;

– бібліотека Passport. Єдине завдання, на вирішення якого спрямована ця бібліотека, полягає в аутентифікації запитів. Процедура розгляду заяв виконується з використанням плагінів, відомих як «стратегії».

1.7 Вибір бази даних

Далі виберемо одну з можливих моделей для зберігання та надавання даних. Серед найбільш часто використовуваних СУБД виділяють реляційні (SQL) і нереляційні (NoSQL).

Спочатку зробимо порівняльний аналіз цих двох технологій, щоб визначити найбільш підходящу для розроблюваної інформаційної системи [11]. У таблиці 1.4 виділимо основні параметри, за якими виконаємо порівняння SQL та NoSQL.

Таблиця 1.4 – Порівняння SQL та NoSQL

Параметр	SQL	NoSQL
Структура і тип збережених даних	Вимагають наявності строго певної структури зберігання даних	Обмежень не ставлять
Мова запитів	Мова структурованих запитів (SQL)	Немає декларативної мови запитів
Масштабованість	Підтримують лише вертикальну масштабованість	Підтримує вертикальну і горизонтальну масштабованість
Надійність	Однозначно попереду	Відстають від реляційних
Тип	Бази даних на основі таблиць	Можуть бути застосовані на документах, парах ключ-значення, графових БД
Підтримка	Має багато років розвитку, тому вирішувати проблеми простіше	Не так все добре, як з реляційними
Основний пріоритет	Підтримка усіх пристроїв і безпека	Простота у використанні і висока продуктивність

У підсумку, переважили такі фактори, як надійність, допустимість зв’язувати таблиці і можливість мати розширені функції з управління даними. Тому для розроблюваної системи будемо використовувати реляційну базу даних (SQL).

Серед найвідоміших реляційних СУБД виділимо MySQL та PostgreSQL. Так як в рамках курсу вже були ознайомлені з СУБД PostgreSQL, тому для розробки інформаційної системи будемо використовувати саме її.

1.8 Висновки до першого розділу

Таким чином, в першому розділі кваліфікаційної роботи було визначено бачення системи. Шляхом пошуку аналогів і порівняння з ними, вдалося сформулювати функціонал, щоб реалізований продукт став унікальним і виключав всі недоліки.

Для розроблюваної системи визначили архітектурний шаблон, а саме, клієнт-серверну архітектуру. Де в якості клієнта виступає веб-браузер, а в якості серверу – сервер додатків. Від клієнта будуть передаватися запити по HTTP-протоколу на сервер, який в свою чергу буде їх обробляти, шляхом взаємодії з базою даних, та повертати відповіді.

Для кожної частини системи був зроблений вибір актуальної технології для розробки. Шляхом порівняльного аналізу визначили, які саме з існуючих технологій підходять для клієнту і для серверу.

Сервер буде написано на JavaScript з використанням фреймворку Express.js, а клієнтська частина на JSX (розширення мови JavaScript) з використанням фреймворку React.js. Для фреймворку сервера виділили основні бібліотеки: для роботи з базою даних спеціальна ORM (бібліотека Sequelize) і бібліотека Passport (для аутентифікації запитів). Фреймворк для клієнту передбачає використання бібліотеки Redux, яка призначена для управління станом додатку.

В обох випадках передбачається використання одного середовища для розробки – це WebStorm.

Виділили основні переваги застосування реляційних СУБД. Вибір пав на використання PostgreSQL. І також з’ясували, який список патернів доцільно використовувати (Factory method, Observer та Singleton).

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Мета і задачі інформаційної системи

Розроблювальна система являє собою клієнт-серверний додаток, основною функцією якого є швидка і зручна взаємодія між звичайними, корпоративними користувачами і майданчиками з лізингу автотранспорту.

Мета ІС: інформаційна система повинна за допомогою user’s interface обробляти заявки клієнта з лізингу автотранспорту і надавати усі можливі пропозиції. Також надавати розрахунок лізингової ставки для конкретної пропозиції в режимі реального часу.

Цільова аудиторія: Можна виділити дві групи. Перша – звичайні клієнти, яким потрібен транспорт для особистого користування. Друга – корпоративні клієнти, які займаються перевезенням людей і товарів.

Основні задачі ИС:

– надання інформації по всім пропозиціям автотранспорту;
– розподіл автотранспорту по категоріям;
– створення заявки по лізингу;
– створення аккаунту користувача;
– збереження історії заявок зареєстрованого користувача;
– можливість стежити і погашати заборгованості;
– отримати консультацію фахівця;
– розрахунок ставки по лізингу в режимі реального часу.

Опис вхідних та вихідних інформаційних потоків ИС.

Вхідні:

– ПІБ та контактні дані клієнта (якщо він незареєстрований);
– обраний транспорт;
– назва міста;
– дані про автотранспорт (модель, вартість);
– тип лізингу;

– термін фінансування;

– відсоток для авансу.

Вихідні:

– повідомлення про успішне або неуспішне створення заявки;

– запис у базі даних інформації про створення заявки та її власника;

– розрахунок щомісячного платежу.

Інформаційні потоки інформаційної системи зображені на рисунку 2.1.

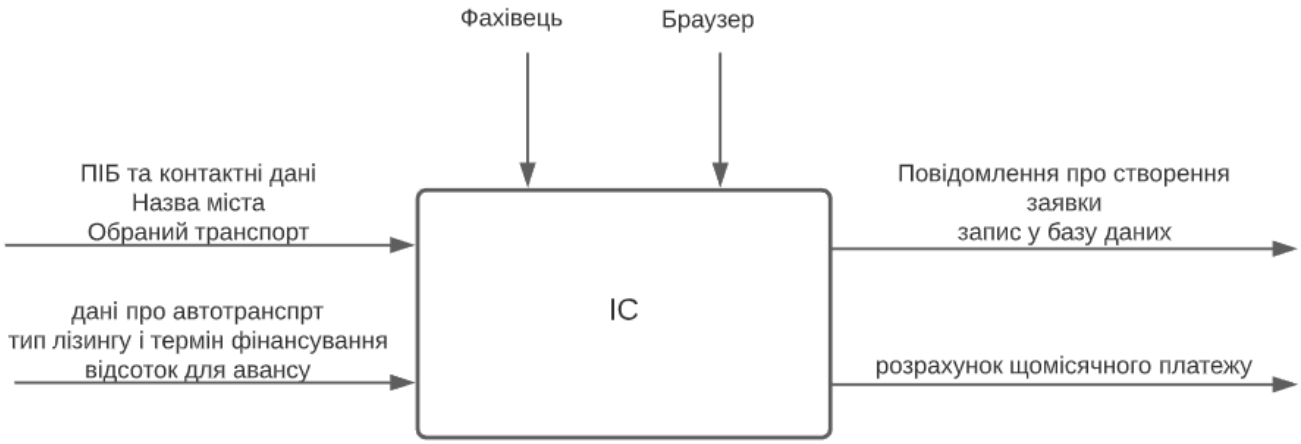

Рисунок 2.1 – Інформаційні потоки розроблювальної ІС

2.2 Типи користувачів

Гість: може розглянути всі пропозиції та оформити заявку, вказавши контактні дані. Низький рівень відповідальності;

Зареєстрований користувач: має такі ж можливості, що і Гість, але контактні дані вже перевірені. Зберігається вся історія заявок (активних чи вже неактивних). Низький рівень відповідальності;

Фахівець: обробляє заявки, які були отримані від користувачів. За вказаними контактними даними зв'язується з користувачем для підтвердження статусу заявки. Високий рівень відповідальності;

Адміністратор: реєструє у системі всіх можливих фахівців, може змінювати особисту інформацію. По мірі необхідності додає нові пропозиції до каталогу. Може отримувати статистику заявок. Високий рівень відповідальності.

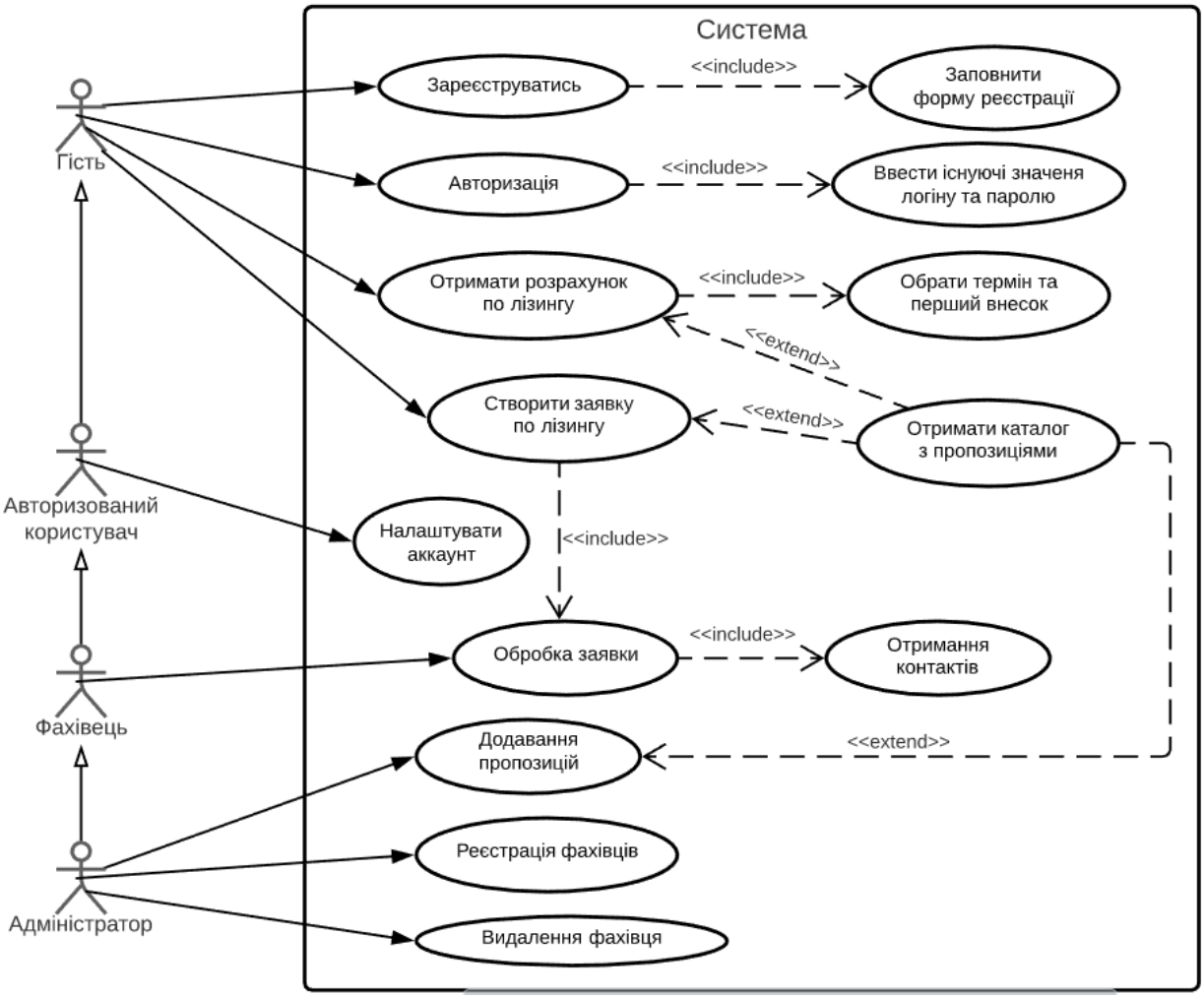

Рисунок 2.2 – Діаграма прецедентів ІС

2.3 Представлення інтерфейсу користувача (UI View)

Перш ніж створювати веб-додаток, необхідно розробити приблизні макети веб-сторінок з певним дизайном і компонуванням. Потрібно позначити

переходи між сторінками та визначити їх вміст і функціонал. Для цього створимо мокапи цих сторінок за допомогою програми Balsamiq Wireframes [13].

Передбачається, що клієнтська частина системи буде складатися з наступних веб-сторінок:

– веб-сторінка реєстрації;

– веб-сторінка авторизації;

– головна сторінка яка в себе включає короткий опис даної платформи та пропонує зробити якісь дії (зайти в свій профіль або зареєструватись; перейти до сторінки зі складанням заявки);

– веб-сторінка з пропозиціями автотранспорту (користувач може обрати місто та категорію транспорту);

– веб-сторінка для перегляду створених заявок (доступно для фахівців);

– веб-сторінка для складання заявки, яка буде містити в собі деякі етапи (вибір транспорту, типу лізингу, терміну і авансу, введенню персональних даних);

– веб-сторінка для розрахунку щомісячної ставки по лізингу для обраної моделі.

На початковому етапі користувач при відвідуванні інформаційної системи потрапляє на головну веб-сторінку. Тут для користувача запропоновані різні варіанти подій, починаючи від реєстрації та входу в систему під своїм профілем і, закінчуючи складанням заявки по лізингу транспорту. Мокап головної веб-сторінки представлений на рис. 2.3.

На цій веб-сторінці користувач може отримати загальну початкову інформацію про систему. Переглянути всі останні пропозиції, які були додані.

За допомогою навігаційного меню переходити на інші веб-сторінки:

– натиснувши на кнопку «1» користувач потрапить на сторінку, яка зображена на рисунку 2.4;

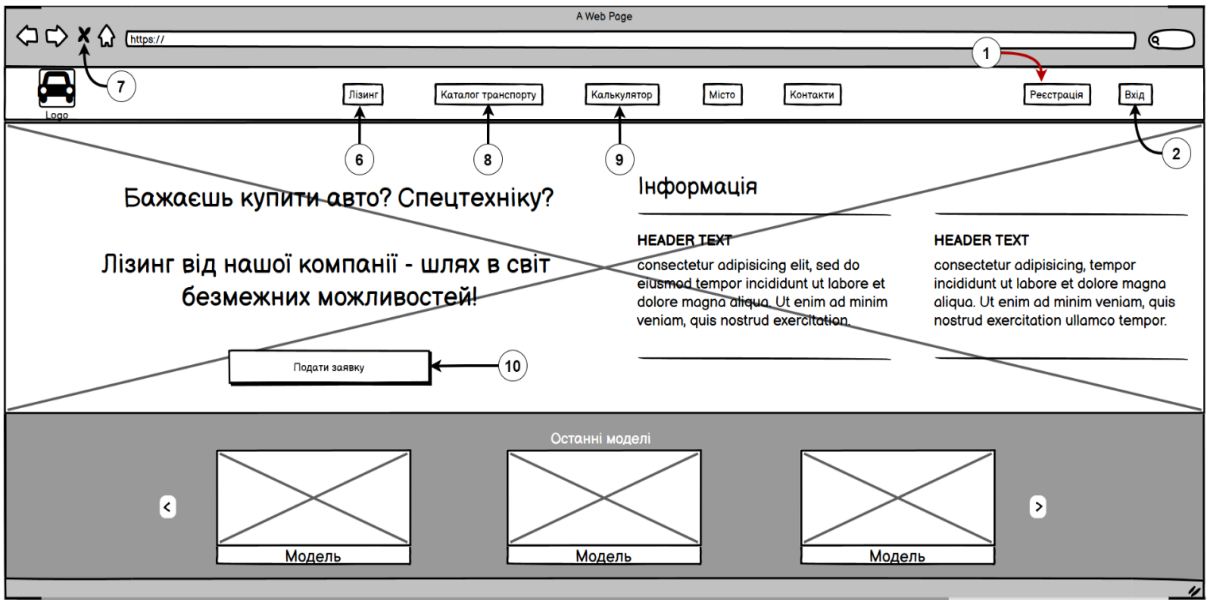

Рисунок 2.3 – Головна веб-сторінка

– натиснувши на кнопку «2» він потрапить на сторінку, яка зображена на рис. 2.5;

– натиснувши на кнопку «8» користувач потрапить на сторінку з усіма пропозиціями. Вона зображена на рис. 2.6;

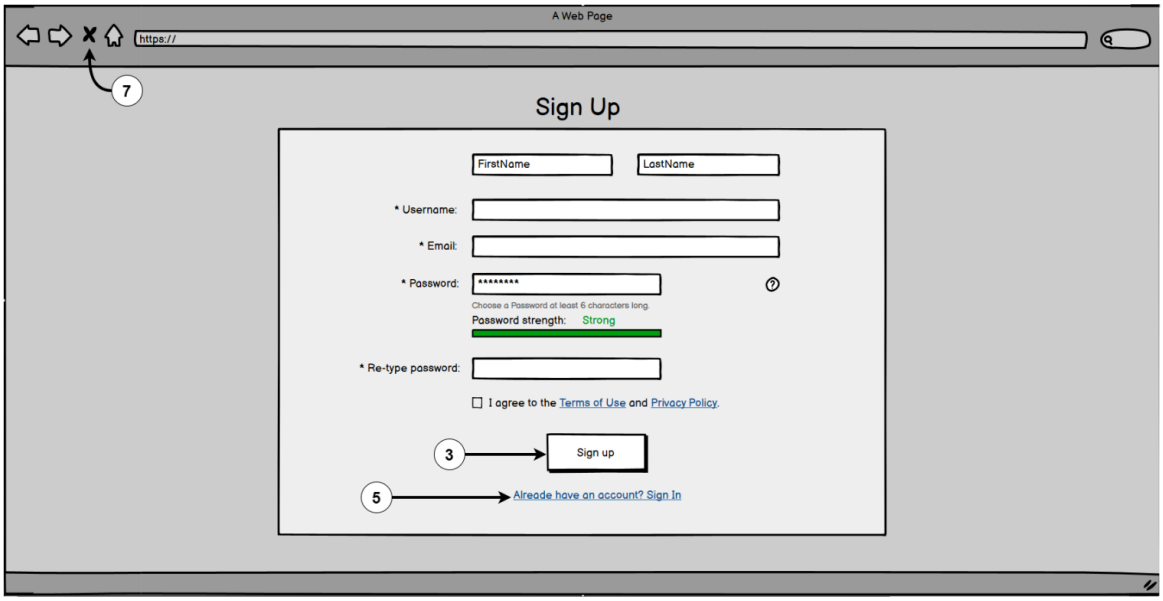

Рисунок 2.4 – Веб-сторінка реєстрації

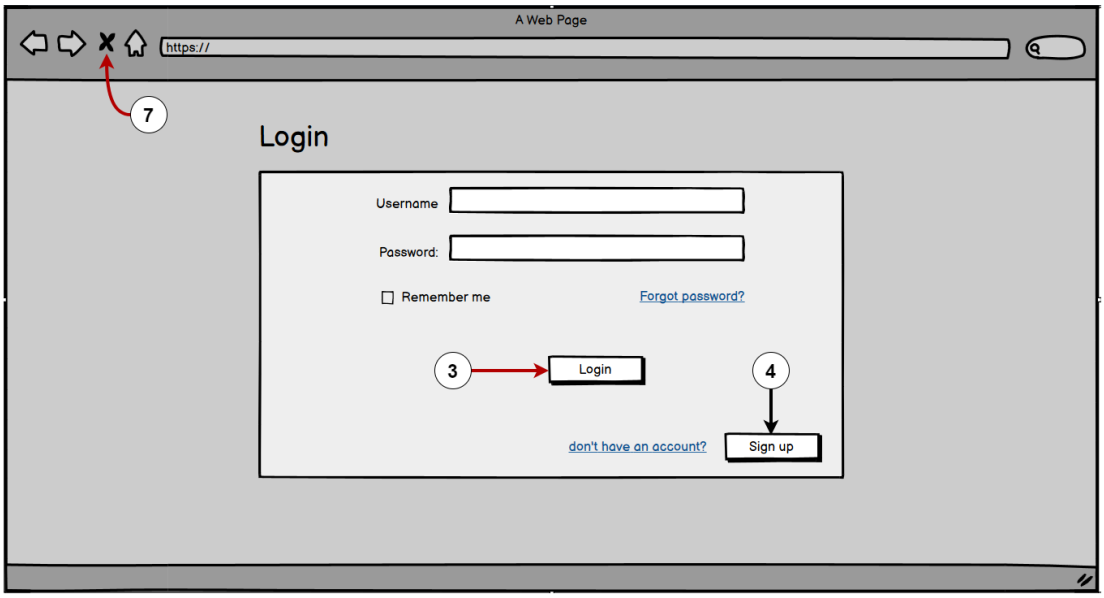

Рисунок 2.5 – Веб-сторінка авторизації

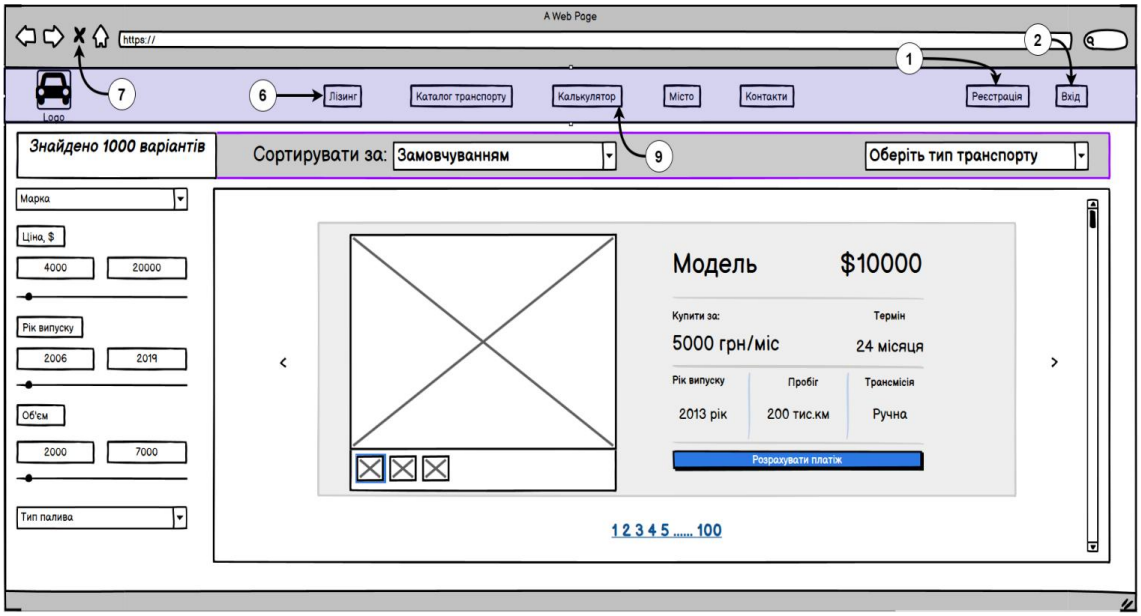

Рисунок 2.6 – Веб-сторінка з пропозиціями

– якщо користувач вирішив створити заявку, він попаде на сторінку, яка зображена на рис. 2.7. Для цього треба натиснути на кнопку «10»;

– натиснувши на кнопку «7» користувач покине сайт;

– в кінцевому підсумку користувач може отримати розрахунок місячної ставки, натиснувши на кнопку «9». Сторінка з цим розрахунком зображена на рис. 2.8.

–

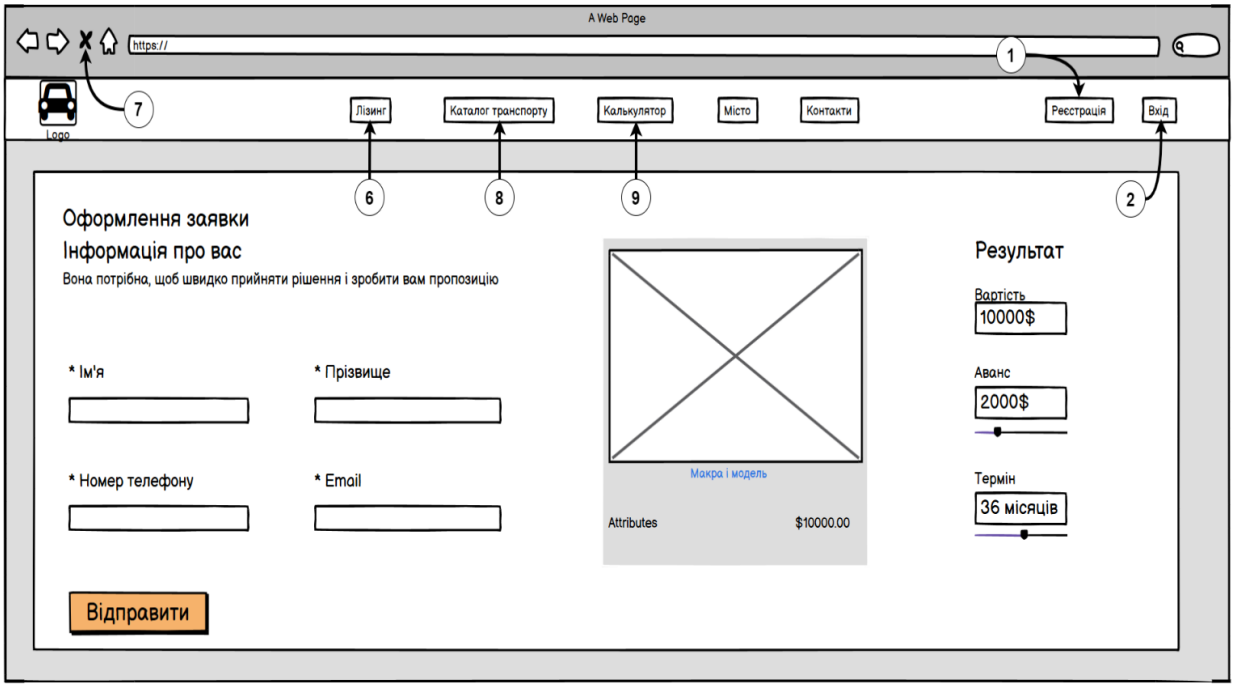

Рисунок 2.7 – Веб-сторінка складання заявки

Опинившись на сторінці реєстрації, користувач може створити акаунт. Для цього йому потрібно заповнити всі поля, дотримуючись усіх правил. Якщо все буде виконане правильно, при створенні система поверне його на головну сторінку (рисунок 2.3) під створеним профілем, якщо він натисне кнопку «3».

Якщо користувач вже має обліковий запис, він може перейти з цієї сторінки на сторінку "Авторизації" на рисунку 2.5. Для цього необхідно натиснути кнопку «5».

Натиснувши на кнопку «7» користувач покине сайт.

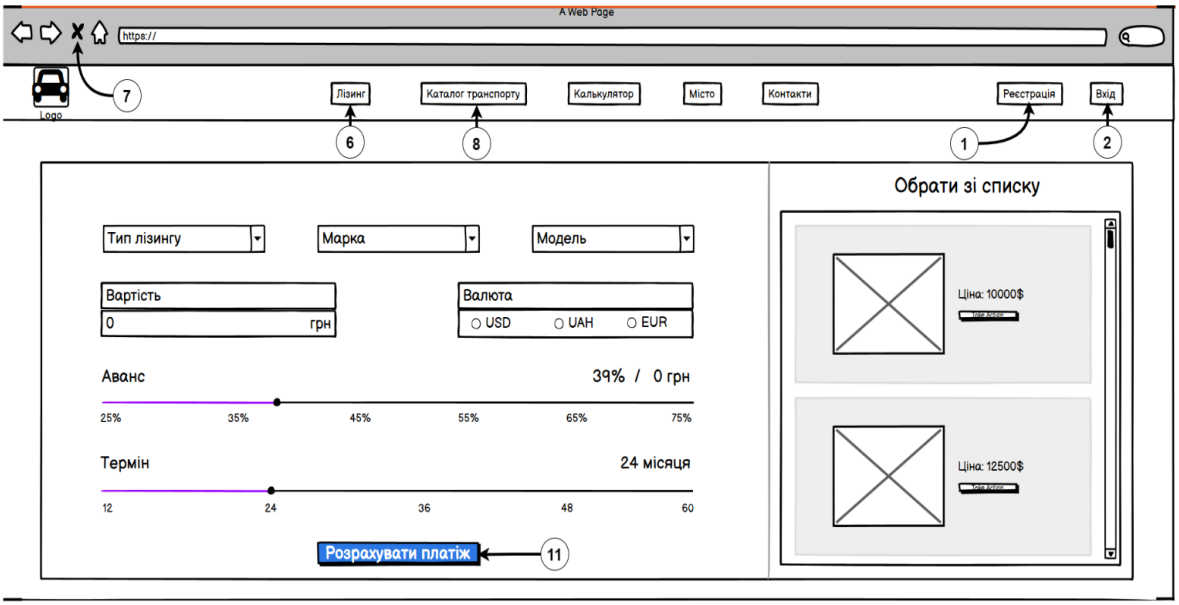

Рисунок 2.8 – Веб-сторінка розрахунку щомісячного платіжу

Потрапивши на веб-сторінку авторизації, користувач може заповнити поля, вказавши існуючий логін і пароль. Якщо все введено правильно, і такий користувач є в системі, то він потрапляє на головну сторінку (рис. 2.3) під своїм профілем, натиснувши на кнопку «3».

Якщо немає дійсного облікового запису, користувач може перейти до веб-сторінці з реєстрації (рис. 2.4), натиснув на кнопку «4».

Натиснувши на кнопку «7» користувач покине сайт.

Опинившись на сторінці з пропозиціями, користувач може ознайомитися з асортиментом. Він може налаштувати потрібні йому параметри (ціна, рік, тип палива). Також є можливість вибрати тип транспортного засобу і виконати сортування. Всі пропозиції представлені у вигляді списку.

За допомогою навігаційного меню можна переходити на веб-сторінки:

– головна (рис. 2.3). Для цього потрібно натиснути на кнопку «6»;
– реєстрації (рис. 2.4). Для цього потрібно натиснути на кнопку «1»;
– авторизація (рис. 2.5). Для цього потрібно натиснути на кнопку «2»;
– розрахунок (рис. 2.8). Для цього потрібно натиснути на кнопку «9»

Натиснувши на кнопку «7» користувач покине сайт.

На веб-сторінці з складанням заявки користувач повинен вказати свої персональні дані для зв'язку. Якщо він знаходиться під особистим підтвердженим профілем, то ці поля заповнюються автоматично. Також ця сторінка містить обрану пропозицію, і є можливість підправити параметри для лізингу.

Користувач може відправити заявку, тоді він отримає статус заявки і потрапить на головну (рис. 2.3), а заявка потрапить в список до фахівців. Цей список відображається у фахівців (рис. 2.9). Або ж користувач може припинити створення заявки і повернутися на головну без відправлення.

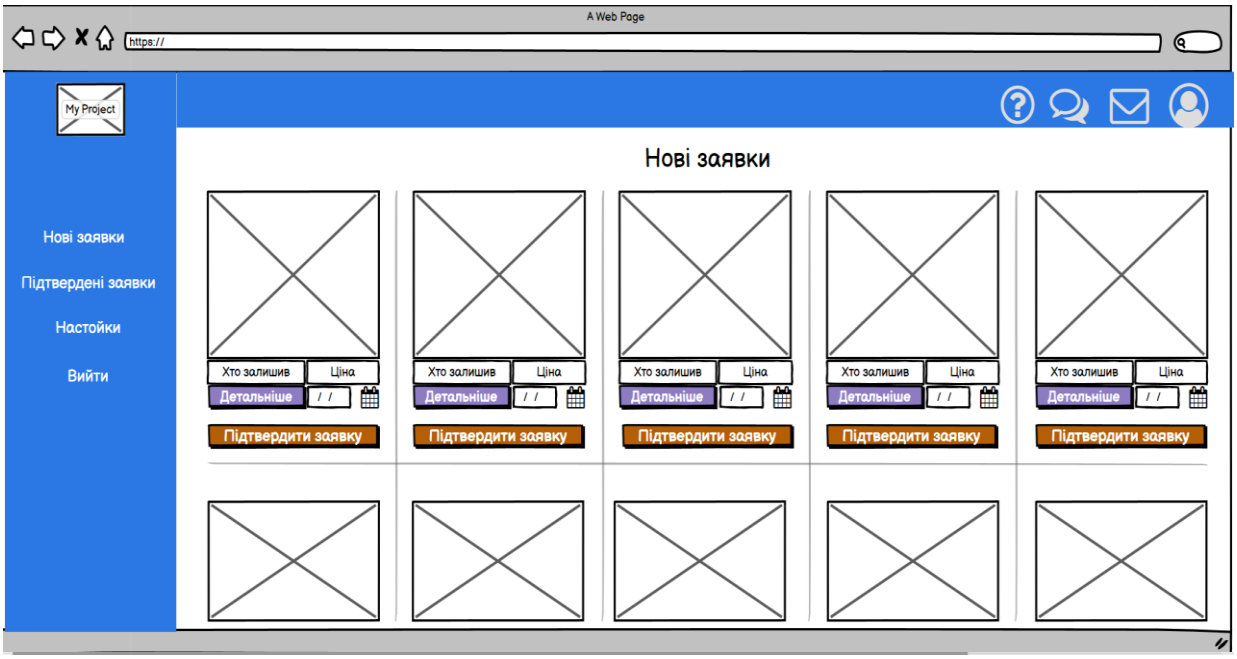

Рисунок 2.9 – Веб-сторінка зі списком заявок (для фахівців)

За допомогою навігаційного меню можна переходити на веб-сторінки:

– головна (рис. 2.3). Для цього потрібно натиснути на кнопку «6»;
– реєстрації (рис. 2.4). Для цього потрібно натиснути на кнопку «1»;
– авторизація (рис. 2.5). Для цього потрібно натиснути на кнопку «2»;
– пропозиції (рис. 2.6). Для цього потрібно натиснути на кнопку «8»;
– розрахунок (рис. 2.8). Для цього потрібно натиснути на кнопку «9».

Натиснувши на кнопку «7» користувач покине сайт.

Для розрахунку щомісячної ставки по лізингу користувачу необхідно налаштувати деякі параметри. З запропонованого списку можна вибрати пропозицію або ж вказати вартість. Після чого потрібно вказати аванс, термін фінансування і тип лізингу.

Якщо все поля відрегульовані, можна отримати приблизну щомісячну вартість.

Якщо все влаштовує, користувач відразу може перейти до оформлення заявки (рис. 2.7), натиснув на кнопку «11».

За допомогою навігаційного меню можна переходити на веб-сторінки:

– головна (рис. 2.3). Для цього потрібно натиснути на кнопку «6»;
– реєстрації (рис. 2.4). Для цього потрібно натиснути на кнопку «1»;
– авторизація (рис. 2.5). Для цього потрібно натиснути на кнопку «2»;
– пропозиції (рис. 2.6). Для цього потрібно натиснути на кнопку «8».

Натиснувши на кнопку «7» користувач покине сайт.

Ця сторінка призначена для працівників компанії. Тут розташовуються всі заявки, які були створені користувачами за допомогою форми (рис. 2.7).

Фахівець переглядає заявки і зв'язується з їх власниками. Якщо власник підтвердив заявку, він змінює її статус на активний. В іншому випадку він її видаляє.

Після створення мокапів системи складемо UML діаграму переходів між сторінками. Ця діаграма представлена на рисунку 2.10.

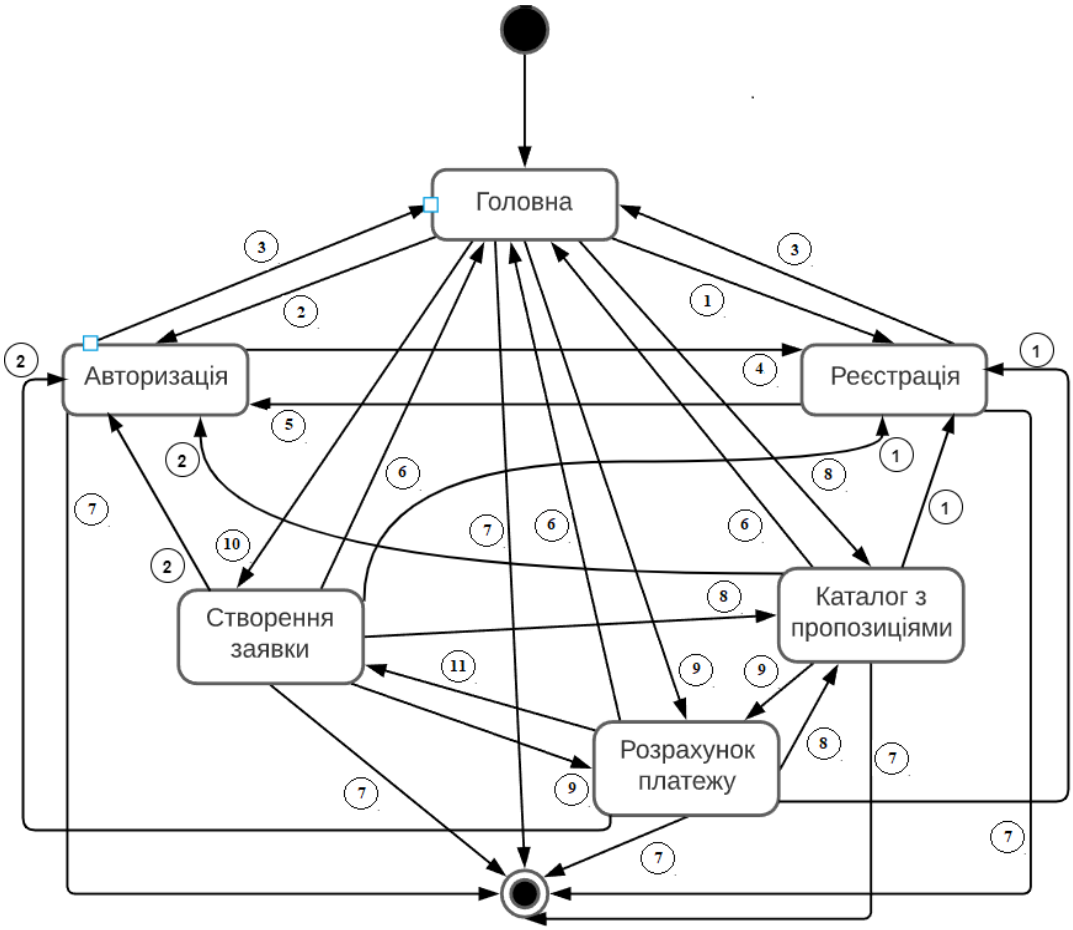

Рисунок 2.10 – Діаграма переходів між сторінками

2.4 Логічне уявлення ІС (Logical View)

Щоб отримати повне уявлення про систему, необхідно відобразити її логічне уявлення у вигляді діаграми, вказавши основні взаємодіючі компоненти (рис. 2.11).

Як можна бачити з діаграми, система взаємодіє з двома акторами. Перший – це користувач, який може створювати заявки і розрахунки, а також отримувати список пропозицій. Другий – фахівець, який обробляє заявки користувачів.

Модель складається з веб-додатку, написаного за допомогою фреймворку React, який відправляє запити на сервер і відображає отримані результати.

Сервер ж включає в себе такі модулі, як контролери і ORM. Контролери обробляють запити і, реалізуючи бізнес-логіку, генерують відповіді. ORM дозволяє пов'язувати бази даних з концепціями об'єктно-орієнтованих мов програмування.

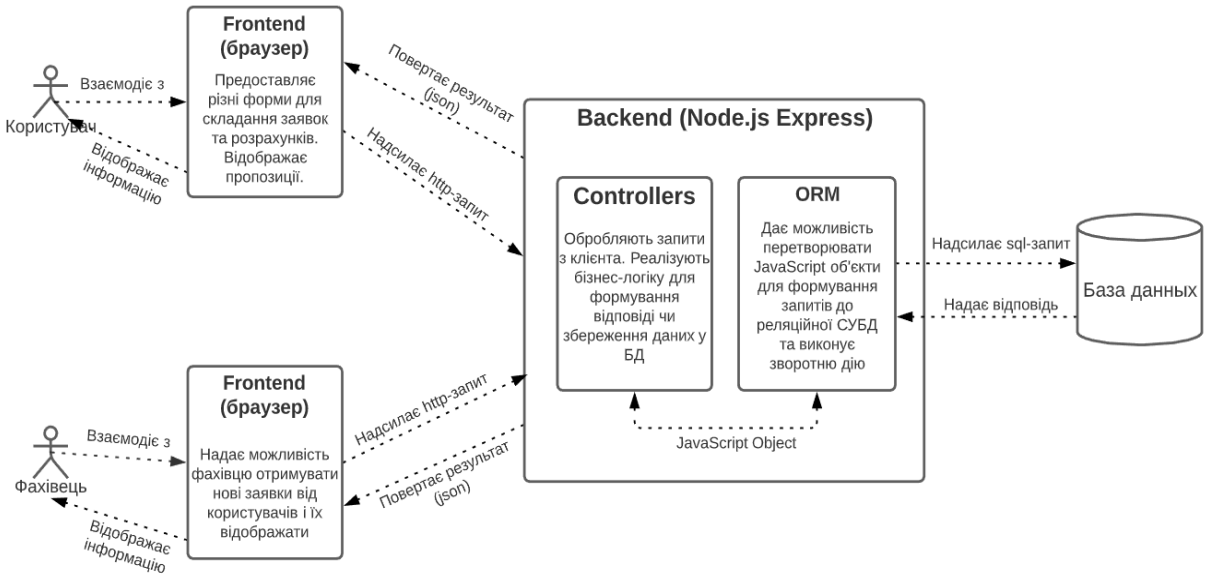

Рисунок 2.11 – Діаграма логічного уявлення ІС

2.5 Уявлення розгортання ІС (Deployment View)

Побудуємо діаграму розгортання, де вкажемо інформацію про просторове розташування обчислювальних контейнерів системи, на яких будуть «хоститься» модулі ІС.

ІС складається з чотирьох контейнерів, кожен з яких включає в себе деякі модулі. Вона включає в себе такі контейнери: ПК користувача, сервер додатків, frontend-частину і сервер бази даних. Повне уявлення зображено на діаграмі розгортання (рис. 2.12).

Клієнтська частина буде реалізована за допомогою фреймворку React. Вона буде розгорнута на безкоштовній хмарної PaaS-платформі Heroku. Одним з головних переваг даної платформи – це підтримка великої кількості мов програмування. За допомогою спеціального домену, сервер Heroku роздаватиме браузерам користувачів сайт з необхідним вмістом.

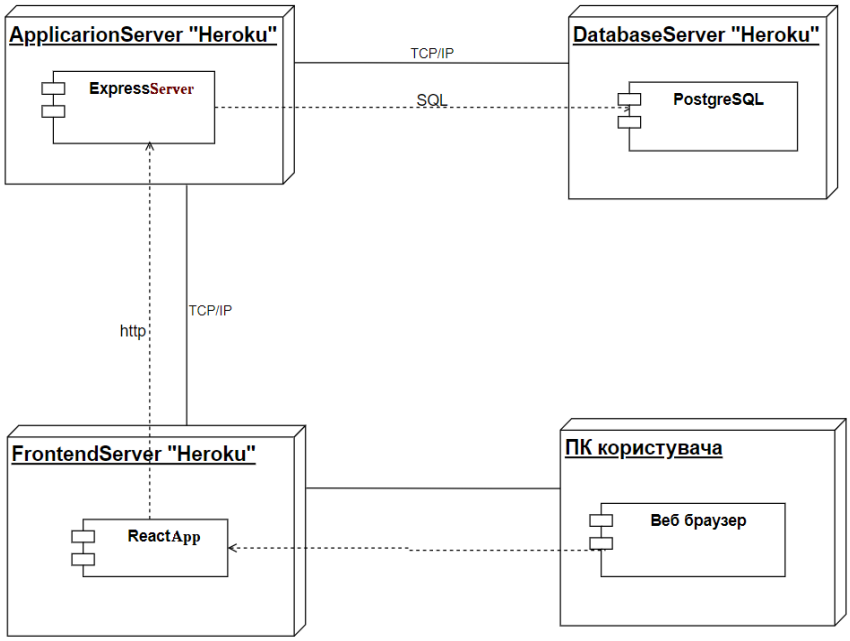

Рисунок 2.12 – Діаграма розгортання ІС

Основний сервер буде реалізований за допомогою фреймворку Express на платформі Node.js. Він буде обробляти запити, отримані від клієнта. За архітектурний підхід для створення API будемо використовувати REST. Таким чином, сервер буде повертати дані в форматі JSON.

Сам сервер також буде розміщений на хмарної платформі Heroku.

В якості серверу бази даних була взята об'єктно-реляційна СУБД PostgreSQL на сервері Heroku. Дані в ній будуть зберігатися у вигляді таблиць, а основний сервер за допомогою ORM зможе здійснювати sql-запити і змінювати її вміст та отримувати необхідну дані.

2.6 Представлення даних ІС (Data View)

Важливим аспектом для повноцінної роботи будь-якої інформаційної системи є можливість зберігання даних. Також виникає необхідність отримувати дані в режимі реального часу або ж перезаписувати їх. Для цього в сис-

темі, що розробляється, передбачено використання однієї з популярних об'єктно-реляційних СУБД, а саме, PostgreSQL.

Її головними перевагами вважається вільність у використанні (безкоштовність) і можливість пов'язувати з концепціями об'єктно-орієнтованих мов програмування за допомогою ORM-бібліотек. До того ж хмарна платформа Heroku надає можливість розгортати PostgreSQL бази даних на своєму сервері.

Для здійснення зіставлення таблиць в базі даних і відносин між ними з класами була взята ORM-бібліотека Sequelize. При використанні Sequelize можна не писати SQL-запити, а працювати з даними як зі звичайними об'єктами. Вона надає високий рівень абстракції над БД схемою і підтримує основні SQL діалекти.

Далі необхідно визначити сутності, які будуть використовуватися в ІС. Всі сутності представлені на рисунку 2.13.

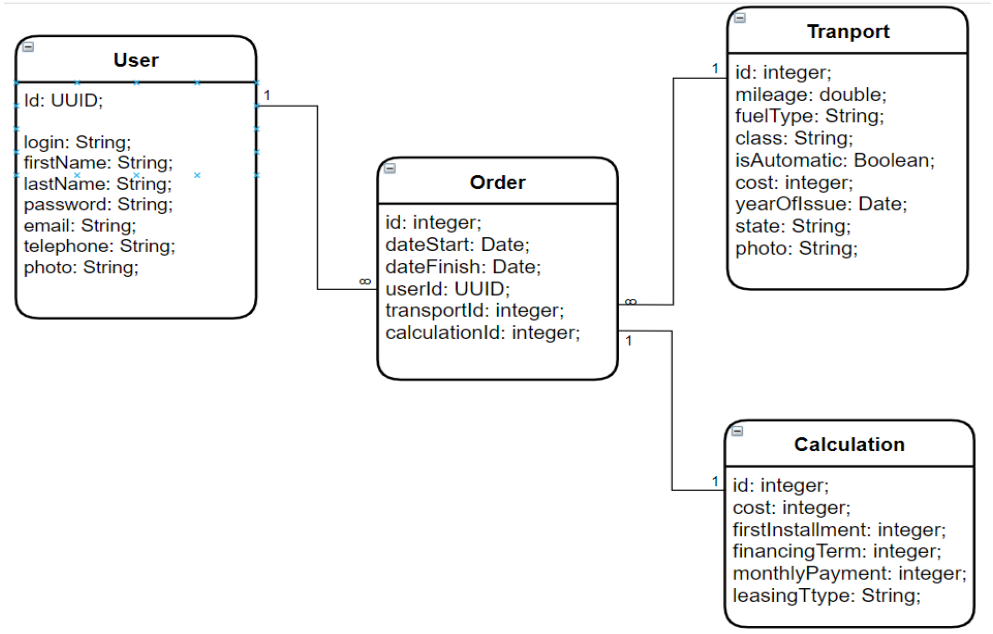

Рисунок 2.13 – Схема об'єктно-реляційної БД

В нашому випадку передбачається використовувати такі сутності:

– User. Дана сутність буде в собі містити загальну інформацію про зареєстрованого користувача і ідентифікувати його при вході. Вона буде пов'язана з сутністю "Заявки" з використанням зв'язку один до багатьох (тобто у одного користувача може бути кілька заявок). В якості первинного ключа виступає поле Id.

– Order. Ця сутність призначена для прив'язки до користувача певної заявки, яка включає в себе деяку пропозицію. Вона в собі буде містити дані користувача, який її створив, а також розрахунок щомісячного платежу. В якості зовнішнього ключа буде виступати поле UserId. Так як в заявці передбачається наявність якоїсь пропозиції, то вона повинна бути пов'язана з сутністю "Транспорт" зв'язком один до багатьох (тобто один транспортний засіб може значитися в різних заявках). Тут зовнішнім ключем виступає поле TransportId.

– Transport. Ця сутність представлятиме всі пропозиції по транспорту, які будуть розміщені в системі. Вона буде пов'язана з сутністю "Заявка" і в якості первинного ключа виступає поле Id.

– Calculation. Дана сутність розширюватиме сутність "Заявка" і міститиме в собі повний розрахунок щомісячного платежу для обраної пропозиції.

2.7 Висновки до другого розділу

В даному розділі виконали проектування інформаційної системи. В ході проектування були сформульовані основні функціональні і нефункціональні вимоги. Визначено типи користувачів і їх можливості, після чого змоделювали діаграму прецедентів. Був визначений архетип системи.

Для кожної окремої сторінки системи був складений мокап з описом основного функціоналу і переходів між сторінками.

Основним етапом стало проектування діаграм. Отримали повне уявлення про систему і основні взаємодіючі компоненти завдяки діаграмі логіч-

ного уявлення. За допомогою діаграм послідовності та діяльності визначили основний сценарій системи. Також були спроектовані такі діаграми, як: розгортання, слоїв, даних та комунікацій. В уявленні безпеки були розглянуті всі заходи безпеки, передбачені для системи.

Розглянуто стек технологій, який доцільно використовувати для подальшої розробки системи.

Таким чином, визначивши основний стек технологій і зробивши проектування, можна перейти до етапу розробки системи.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Уявлення про структуру проекту

В цьому пункті розглянемо структуру проекту, яка буде складатися з веб-частини та основного серверу. Для кожної окремої частини системи представимо діаграму, яка представляє з себе розбиття на модулі. В нашому випадку в якості модулів виступають директорії.

Структура проекту веб-частини представлена на рисунках 3.1 та 3.2.

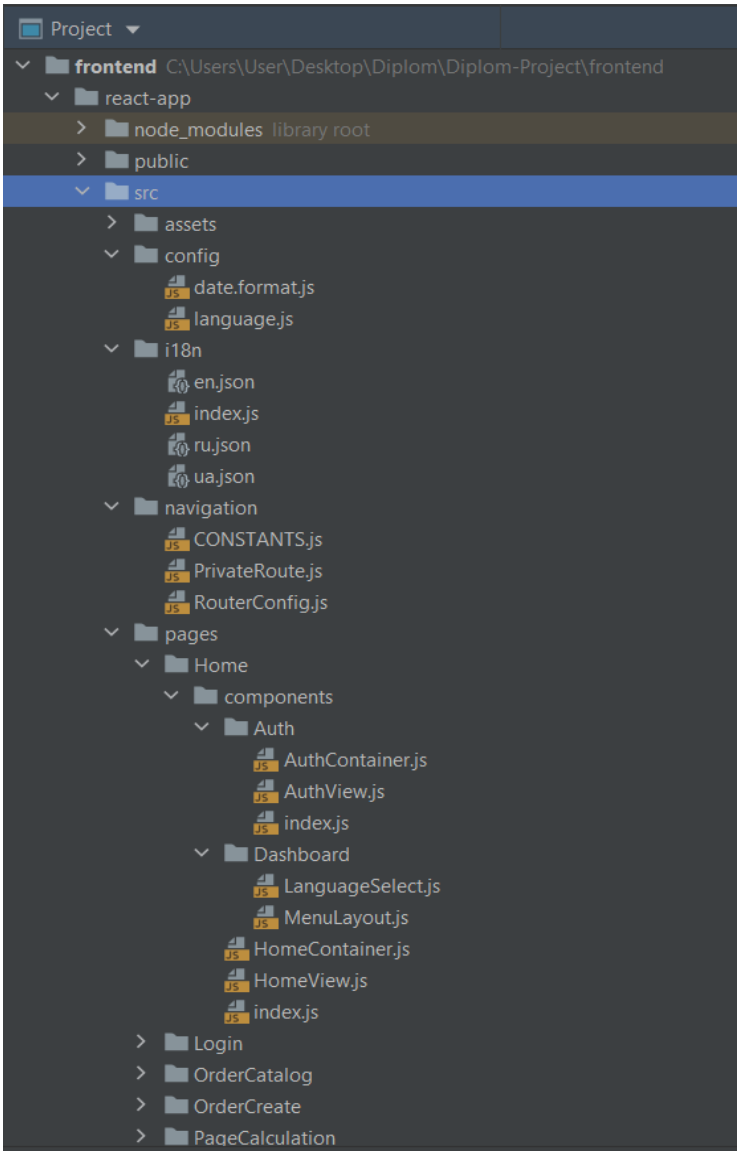

Рисунок 3.1 – Структура проекту веб-частини

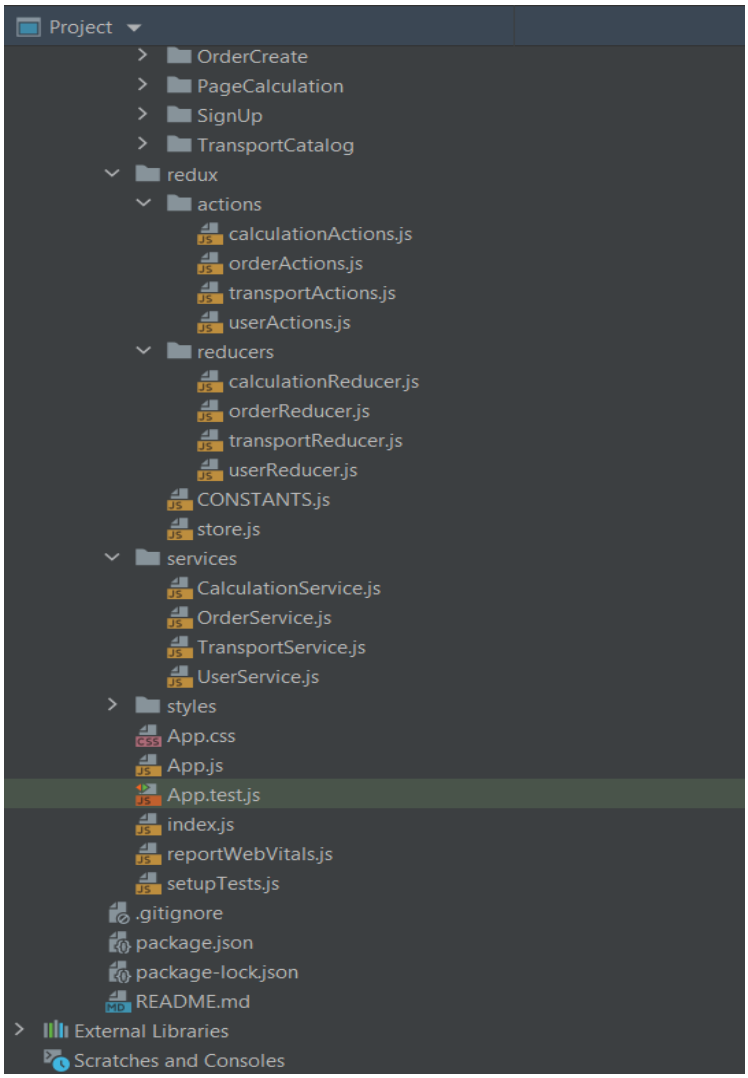

Рисунок 3.2 – Продовження структури проекту веб-частини

Веб-частина системи складається з таких основних директорій, як:

– «assets» директорія. Тут знаходяться всі статичні файли, такі як фото, логотипи, векторні іконки;

– «config» директорія. У даній директорії розташовані конфігурації програми, такі як формат дати і налаштування мови за умовчанням;

– «i18n» директорія. Інтернаціоналізація або ж багатомовна підтримка. Вона складається з файлу конфігурації index.js і всіх перекладів з інших мов в незалежних файлах language.json;

– «navigation» директорія. Вона в собі містить всю логіку маршрутизації. Включені два типу маршрутів: публічний і приватний, з яких приватні – це ті, які вимагають аутентифікації. RouterConfig.js матиме всі маршрути додатку, які визначені в одному місці. PrivatRoute.js – це компонент для додавання перевірки аутентифікації користувача для приватних маршрутів. CONSTANTS.js складається з усіх констант для різних доступних маршрутів в нашому додатку;

– «pages» директорія. У неї будуть включені окремі директорії, такі як: «Home», «Login», «OrderCatalog», «OrderCreate», «PageCalculation», «SignUp» і «TransportCatalog». Вони в свою чергу будуть відповідати за відображення основних сторінок додатку. Так сторінка Home містить в собі ті компоненти, які необхідні тільки їй. Файл HomeContainer та інші компоненти всередині директорії Home забезпечують файл HomeView усіма необхідними даними, який в свою чергу їх відображає;

– «redux» директорія. Ця частина допомагає налаштувати роботу з бібліотекою Redux, яка дозволяє стежити за станом додатку. Ключовими модулями тут можна виділити дії (actions), редуктори (reducers), і сховище (store). Кожен файл дій включає в себе творців дій на основі функцій. Так наприклад файл userActions.js матиме всі дії, пов'язані зі станом користувача. У свою чергу userReducer.js містить чисті функції, які будуть відповідати за оновлення стану користувача. У файлі store.js зберігається об'єкт, який містить стан додатку. Він дає оповіщення всім слухачам, які на нього підписані про те, що він змінився. Тим самим дозволяє робити запити на сервер або ж відображати новий контент;

– «services» директорія. Для даної директорії буде інтегрована бібліотека Axios. За допомогою неї будуть управлятися всі запити до сервера. Запити будуть виконуватися щодо кожного типу даних (користувача, заявки, транспорту і розрахунку). При цьому кожен новий стан тієї чи іншої моделі

буде зберігатися в redux-файлі store.js для подальшого використання в компонентах. А вони в свою чергу будуть відображати всі зміни;

– «styles» директорія. У ній визначені основні стилі додатку. Може включати певні теми для інтерфейсу додатку, а також глобальні або часто використовувані стилі.

– Файл index.js, який знаходиться в директорії «src/» відповідає за точку входу за замовчуванням для кожної програми.

Наступним етапом розглянемо структуру проекту серверу. Вона зображена на рис. 3.3.

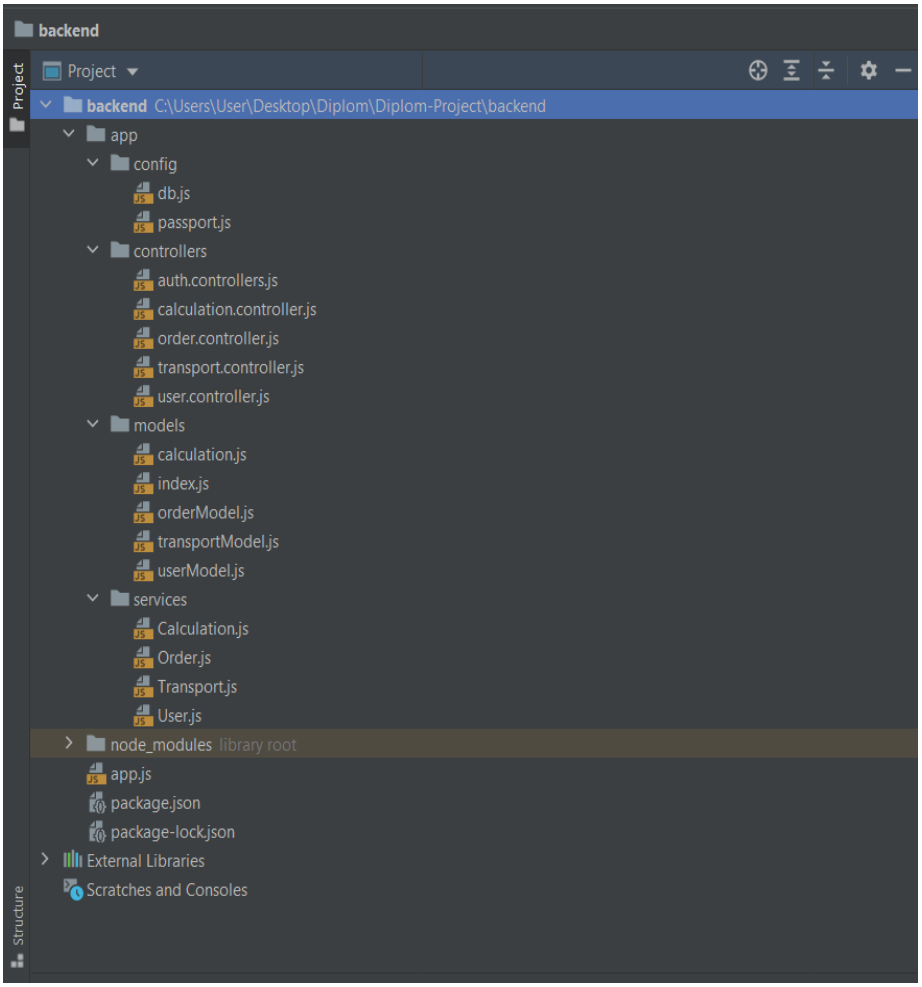

Рисунок 3.3 – Структура проекту серверу

Також для даної частини системи виділимо основні директорії і їх призначення. Отже, маємо наступні директорії:

– «config» директорія. Вона призначена для збору конфігурацій для додатку. В файлі db.js містяться загальні відомості для підключення до бази даних (порт, хост, назву самої бази даних, а також ім'я і пароль користувача, який до неї підключився). Що стосується файлу passport.js, він відповідає за настройку аутентифікації користувача в системі. Вона дозволяє убезпечити доступ до деяких ресурсів серверу. Таким чином, роблячи запит на такий ресурс, система також буде вимагати спеціальний токен для ідентифікації користувача.

– «controllers» директорія. Тут будуть визначені основні маршрути, за якими надаватиметься доступ до API. Контролери викликатимуть відповідні запитам функції з модуля «services», які будуть обробляти дані з запиту і генерувати відповідь. А контролери в свою чергу передаватимуть ці відповіді на веб-частину системи.

– «models» директорія. Дана директорія описує всі моделі системи. У неї включені такі моделі : User – користувач, Order – заявка, Transport – транспорт та Calculation – розрахунок. Також за допомогою ORM бібліотеки Sequelize встановлюється підключення з базою даних PostgreSQL. Вона в свою чергу дає можливість встановити зв'язки між таблицями. Все це дає можливість працювати з даними тільки у вигляді об'єктів.

– «services» директорія. У даній директорії для кожної окремої моделі створюється своя бізнес-логіка. Ця логіка в свою чергу відповідає за обробку даних, які були отримані з клієнта, і, шляхом звернень до бази даних, формує відповіді для передачі назад на сам клієнт. Таким чином, для моделі transportModel в файлі Transport.js буде формуватися логіка, пов'язана з додаванням нової пропозиції, редагування параметрів та інше.

– «node_modules» директорія. В цій директорії як для серверної частини, так і для веб-частини містяться всі сторонні бібліотеки, які вбудовані або ж окремо завантажені через консоль за допомогою команди npm.

– Файл app.js, який знаходиться в директорії app/ відповідає за налаштування веб-серверу Express.

3.2 Уявлення про класи ІС

Для кожної частини системи (сервера і веб-частини) наведемо діаграми класів і їх безпосередню взаємодію. Також в даних діаграмах відобразимо ті патерни, які використовуються в проекті.

Сервер передбачає використання такого патерну, як Factory method (фабричний метод). Даний патерн дозволить нам створювати об'єкти розрахунку за допомогою фабричного методу, тим самим дозволить змінювати тип цих об'єктів. У нашому випадку це потрібно для створення двох типів розрахунку: фінансового та оптимального.

Для веб-частини вибір припав на використання таких патернів, як: Observer (спостерігач) і Singleton (сінглтон).

Патерн Singleton буде застосовуватися до слою Store, де буде створюватися об'єкт store і зберігати в собі початковий стан додатку.

Патерн Observer також буде застосовуватися до об'єкта store. Однак в цьому випадку, на всі зміни, що будуть в ньому відбуватися, підписуватимуться компоненти (що відповідають за відображення контенту) і сервіси (що відповідають за запити на сервер).

Для початку покажемо діаграму класів для сервера. Вона представлена на рис. 3.4.

Класи UserModel, TransportModel та OrderModel визначають базові моделі для роботи з PostgreSQL і дозволяють перетворювати дані в об'єкти.

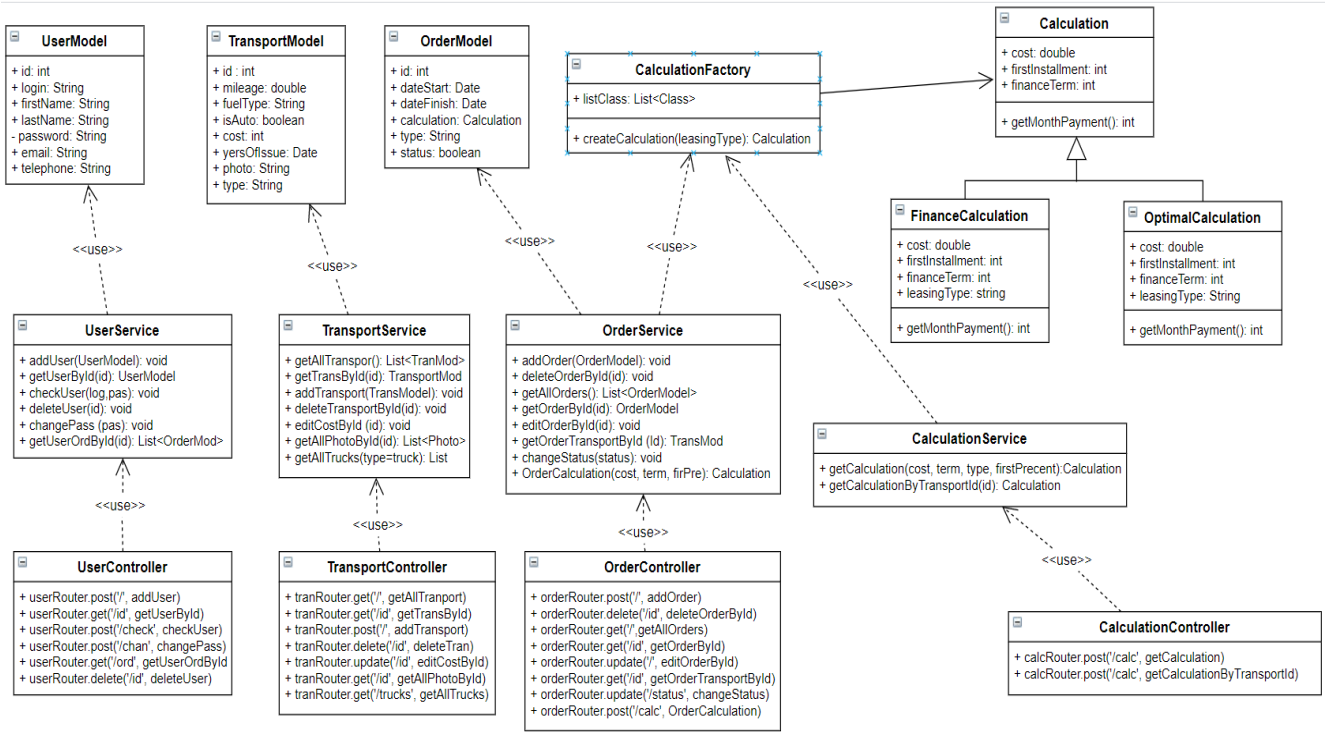

Рисунок 3.4 – Діаграма класів серверу

Кооперація класів Calculation, FinanceCalculation, OptimalCalculation та CalculationFactory дозволяє реалізувати патерн фабричного методу. Клас CalculationFactory містить метод createCalculation(leasingType): Calculation, який дозволяє створити розрахунок того чи іншого типу.

Клас UserService відповідає за основну бізнес-логіку, пов'язану з об'єктами користувачів. Він має наступні методи:

– addUser(UserModel): void – метод, який добавляє нового користувача;

– getUserById(id): UserModel – метод, який повертає конкретного користувача за заданим id;

– checkUser(login, password): status – метод, який перевіряє наявність користувача у системі;

– deleteUser(id): void – метод, який видаляє інформацію щодо користувача по наведеному id;

– changePassword(password): void – метод для зміни пароля користувача;

– getUserOrdersById(id): List<OrderModel> – метод, який повертає усі заявки користувача.

Клас TransportService відповідає за основну бізнес-логіку, пов'язану з об'єктами транспорту. Він має наступні методи:

– getAllTransports(): List<TransportModel> – метод, який повертає усі пропозиції транспорту;

– getTransportById(id): TransportModel – метод, який повертає інформацію щодо обраного транспорту;

– addTransport(TransportModel): void – метод, який додає нову пропозицію у кталог;

– deleteTransportById(id): void – метод, який видаляє обрану пропозицію;

– editCostById(id, cost): void – метод для редагування вартості пропозиції;

– getAllPhotoById(id): List<Photo> – метод, який повертає усі фото обраної пропозиції;

– getAllTrucks(type=’truck’): List<TransportModel> – метод, що повертає усі вантажівки.

Клас OrderService відповідає за основну бізнес-логіку, пов'язану з об'єктами заявок. Він має наступні методи:

– addOrder(OrderModel): status – метод, який додає нову заявку;

– deleteOrderById(id): void – метод, що видаляє обрану заявку;

– getAllOrders(): List<OrderModel> – метод, який повертає усі нові заявки;

– getOrderById(id): OrderModel – метод, який повертає дані конкретної заявки;

– editOrderById(id, OrderModel): void – метод, який редагую параметри для обраної заявки;

– getOrderTransportById(id): TransportModel – цей метод повертає пропозицію, що уключена у заявку;

– changeStatus(id, status): void – метод, який змінює статус у обраної заявки;

– orderCalculation(id, cost, term, firstPrecent): Calculation – метод, який робить розрахунок для обраної заявки.

Клас CalculationService відповідає за основну бізнес-логіку, пов'язану з об'єктами розрахунків. Він має наступні методи:

– getCalculation(cost, term, type, firstPrecent): Calculation – метод, який повертає розрахунок щомісячного платежу для конкретних параметрів;

– getCalculationByTransportId(id): Calculation – цей метод повертає розрахунок для конкретної пропозиції.

Класи UserController, OrderController, TransportController та CalculationController відповідають за обробку запитів, отриманих з веб-частини. Вони в собі містять два параметри:

– перший містить маршрут, по якому здійснюється доступ до API;

– другий – виклик відповідного методу з сервісу, який здійснює бізнес-логіку роботи з вхідними даними.

Також для веб-частини додатку складемо діаграму класів, що буде містити базові каркаси класів. Ця діаграма представлена на рис. 3.5.

Класи UserViewComponent, OrderViewComponent, TransportViewComponent та CalculationViewComponent відповідають за відображення контенту, для цього використовується для кожного з них метод render().

Клас Store містить в собі стан додатку щодо усіх моделей. Для всієї програми створюється лише один екземпляр store даного класу (реалізується патерн Singleton), до якого можна звертатися і постійно оновлювати стан з

будь-якого місця. Також він повідомляє інші класи, які відповідають за передачу даних для відображення (ContainerComponent) та за комунікацію з сервером (Services), про те, що він змінився. Для цього використовується патерн Observer. Завдяки методу subscribe() даний екземпляр буде підписувати інші класи на оновлення свого стану.

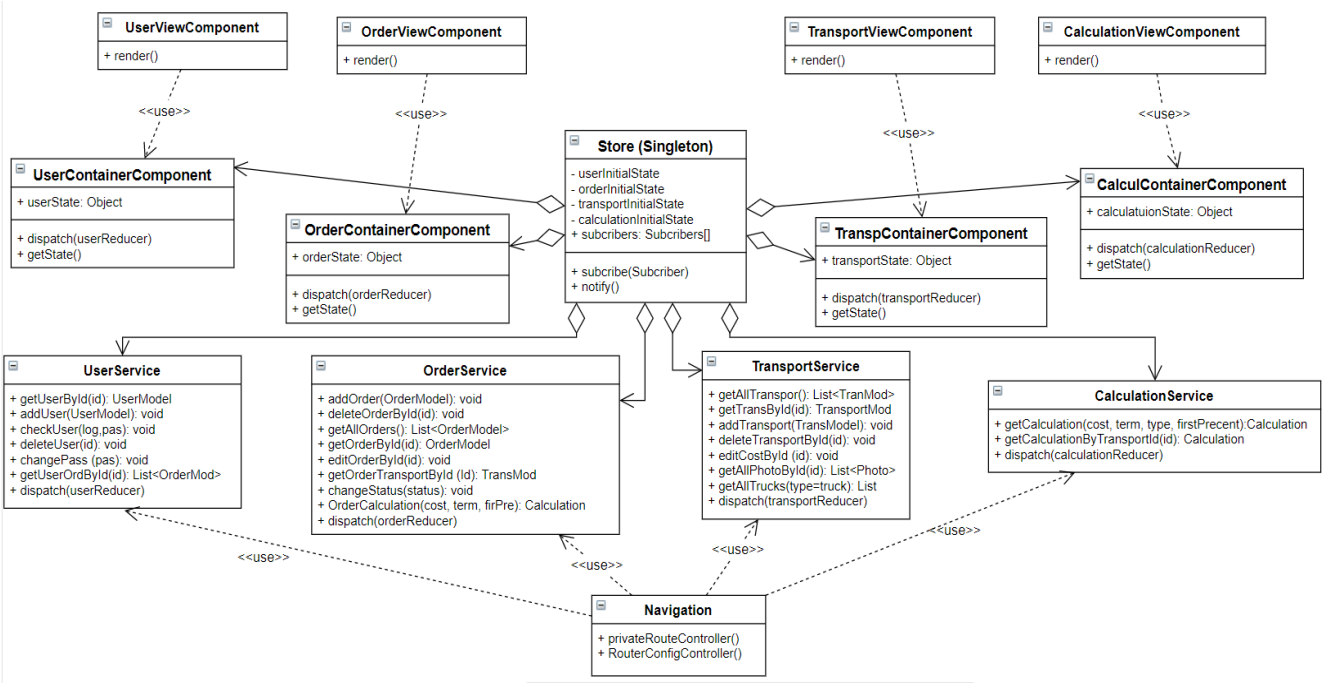

Рисунок 3.5 – Діаграма класів веб-частини

Класи UserContainerComponent, OrderContainerComponent, TransportContainerComponent та CalculationContainerComponent служать для передачі даних в компоненти для відображення. Для них визначено стан своєї моделі, яка може оновлюватися. Тому кожен компонент, підписаний на оновлення стану екземпляра store в класі Store, може отримати цей стан за допомогою методу getState(). Також ці компоненти можуть змінювати стан даного екземпляра, якщо на сторінці відбулися зміни. Для цього викликається метод dispatch(reducer).

Класи UserService, OrderService, TransportService та CalculationService, як і для серверу, відповідають за основну бізнес-логіку комунікації між цими

двома частинами. Вони реалізують майже всі ті ж самі методи, що й слой бізнес-логіки на сервері. Але вони ще мають можливість постійно оновлювати екземпляр store за допомогою методу dispatch(reducer) і викликатися в той момент, коли він змінився без їх впливу.

Клас Navigation служить для створення запитів на сервер. Він в собі містить маршрут і необхідну дію, яке береться з класів Services. Також ці запити будуть розділятися на два типи:

– з використанням аутентифікації користувача для доступу до ресурсів;

– без використання правил аутентифікації користувача.

ВИСНОВКИ

У даній кваліфікаційній роботі був розроблений веб-ресурс для лізингу автотранспорту. Веб-сервіс надає користувачам великий функціонал для взаємодії з лізинговою компанією. Система дозволяє швидко визначити каталог пропозицій для обраного міста, каталог не обмежений тільки легковими автомобілями, також можна швидко отримати розрахунок щомісячного платежу для обраного типу лізингу та пропозиції. Сам додаток реалізовано за допомогою клієнт-серверної архітектури. В якості клієнту виступає веб-додаток, написаний з використанням мови програмування JavaScript і фреймворку React.js. Серверна частина додатку реалізована на базі платформи Node.js з використанням фреймворку Express.js. В якості сховища для збереження даних виступає СУБД PostgreSQL. Таким чином, мета, що була поставлена, досягнута в повному обсязі.

Для досягнення цілі були вирішені наступні задачі.

У першому розділі провели порівняльний аналіз з аналогічними рішеннями і визначили основний функціонал, який дозволить удосконалити і зробити систему більш універсальною. Визначили архітектурний принцип, за яким будуть взаємодіяти компоненти сервера і клієнта (архітектурний стиль REST).

У другому розділі виконали проектування інформаційної системи, де були сформовані функціональні та нефункціональні вимоги. Побудована діаграма прецедентів, в якій визначені усі типи користувачів та їх функції. Також був розроблений користувацький інтерфейс, який дозволить легко орієнтуватися на сайті системи. Вся робота додатку описується в наступних діаграмах: логічного уявлення, уявлення слоїв, уявлення процесів та уявлення комунікацій. Спроектували об'єктно-реляційну модель бази даних. Побудували діаграму розгортання, де вказали інформацію про просторове розташування обчислювальних контейнерів, на яких будуть «хоститься» модулі системи.

У третьому розділі виконали реалізацію інформаційної системи, яка включає такі етапи, як: розробка структури проекту, створення діаграм класів для веб-частини та серверу, які демонструють використання патернів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. AVIS. Авто у лізинг для юридичних осіб та представництв. URL: https://avisleasing.com.ua (дата звернення 14.05.2022)
2. Автовикуп в Києві AutoStar. [Електронний ресурс] / auto-star. URL: https://auto-star.net.ua (дата звернення 14.05.2022)
3. Дружній автолізинг Хапай. URL: https://hapai.kiev.ua/ua/city-odessa (дата звернення 14.05.2022)
4. Лізинг від ULF Finance – шлях у світ безмежних можливостей! URL: https://ulf.ua/city-odessa (дата звернення 14.05.2022)
5. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення 14.05.2022)
6. Використання JavaScript на веб-сторінках. URL: https://professorweb.ru/my/javascript/js_theory/level2/2_1.php (дата звернення 14.05.2022)
7. Введення в JSX. URL: https://ru.react.js.org/docs/introducing-jsx.html (дата звернення 14.05.2022)
8. Основні мови для бэкенда. URL: https://tproger.ru/translations/7-basic-languages-for-backend-development/ (дата звернення 14.05.2022)
9. Best Frontend Frameworks of 2021 for Web Development. URL: https://www.simform.com/best-frontend-frameworks/ (дата звернення 14.05.2022)
10. 10 кращих фреймворков Node.js. URL: https://medium.com/nuances-of-programming/10-лучших-фреймворков-node-js-a8b6c42c3e2c (дата звернення 14.05.2022)
11. SQL против NoSQL. URL:https://coderlessons.com/tutorials/bazy-dannykh/osnovy-subd/17-sql-protiv-nosql (дата звернення 14.05.2022)

12. Каталог паттернів проектування. URL: https://refactoring.guru/ru/design-patterns/catalog (дата звернення 14.05.2022)
13. Balsamiq Wireframes is a graphical user interface website wireframe builder application. URL: https://balsamiq.com (дата звернення 14.05.2022)
14. The 4 Layers of Single Page Applications You Need to Know. URL: https://hackernoon.com/architecting-single-page-applications-b842ea633c2e (дата звернення 14.05.2022)
15. Bulletproof node.js project architecture. URL: https://dev.to/santypk4/bulletproof-node-js-project-architecture-4epf (дата звернення 14.05.2022)
16. Authentication and Authorization with JWTs in Express.js. URL: https://stackabuse.com/authentication-and-authorization-with-jwts-in-express-js/ (дата звернення 14.05.2022)
17. Node.js. URL: https://nodejs.org/uk/ (дата звернення 14.05.2022)
18. Express.js. URL: https://expressjs.com/ru/ (дата звернення 14.05.2022)
19. Sequelize. URL:https://sequelize.org (дата звернення 14.05.2022)
20. Faker. URL:https://faker.readthedocs.io/en/master/ (дата звернення 14.05.2022)
21. Passport. URL: http://www.passportjs.org (дата звернення 14.05.2022)
22. React.js. URL: https://ru.reactjs.org (дата звернення 14.05.2022)
23. Redux. URL: https://redux.js.org (дата звернення 14.05.2022)
24. Axios. URL: https://axios-http.com/docs/intro (дата звернення 14.05.2022)
25. PostgreSQL. URL: https://www.postgresql.org (дата звернення 14.05.2022)
26. WebStorm. Умная IDE для JavaScript. URL: https://www.jetbrains.com/ru-ru/webstorm/ (дата звернення 14.05.2022)

27. Monitoring Heroku Apps. URL: https://devcenter.heroku.com/articles/monitoring-apps (дата звернення 14.05.2022)

ДОДАТОК А
КЕРІВНИЦТВО АДМІНІСТРАТОРА ІС

Для веб-частини додатку досить того, щоб той хостинг-сервіс, на який вона буде розгорнута, просто міг повертати в будь-який час відповідні файли сторінок для обробки браузером. При цьому обов'язковим параметром є наявність доменного імені, по якому користувачі зможуть отримати доступ до сайту.

Вимоги до розгортання серверної частини додатку наступні:

– безперебійна робота 24/7;

– підтримка Node.js додатків;

– обов'язкове підключення до віддаленої або локальної бази даних PostgreSQL.

Незалежно від того, де саме розташовується база даних (на локальній машині або видаленої), для підключення до неї потрібно виконати наступні кроки:

– знайти директорію серверної частини /config і відкрити файл db.js;

– вказати відповідні параметри імені БД, імені користувача та його пароля, а також порт і хост;

– перезапустити сервер.

Після чого в підключеної базі даних створяться нові таблиці з усіма типами зв'язків. У тому випадку, якщо таблиці вже існують, вони видаляться і заново створяться.

ДОДАТОК Б

КОД ПРОГРАМИ

Файл urls.py

```
from django.contrib import admin
from django.urls import path,include
urlpatterns = [
path('admin/', admin.site.urls),
#path to djoser end points
path('auth/', include('djoser.urls')),
path('auth/', include('djoser.urls.jwt')),
#path to our account's app endpoints
path("api/accounts/",include("accounts.urls"))
]
```

Файл models.py

```
from django.db import models
from django.contrib.auth.models import User
class AuthUserModel(models.Model):
us-
er=models.OneToOneField(User,on_delete=models.CASCADE,related_name=
"profile")
description=models.TextField(blank=True,null=True)
location=models.CharField(max_length=30,blank=True)
date_joined=models.DateTimeField(auto_now_add=True)
updated_on=models.DateTimeField(auto_now=True)
is_organizer=models.BooleanField(default=False)
def __str__(self):
return self.user.username
```

Файл signals.py

```
from django.contrib.auth.models import User 90
```

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import userProfile
@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
if created:
userProfile.objects.create(user=instance)
from django.apps import AppConfig
class AccountsConfig(AppConfig):
name = 'accounts'
def ready(self):
import accounts.signals
```

Файл serializers.py

```
from rest_framework import serializers
from .models import userProfile
class userProfileSerializer(serializers.ModelSerializer):
user=serializers.StringRelatedField(read_only=True)
class Meta:
model=userProfile
fields='__all__'
```

Файл views.py

```
from rest_framework.generics import
(ListCreateAPIView,RetrieveUpdateDestroyAPIView,)
from rest_framework.permissions import IsAuthenticated
from .models import userProfile
from .permissions import IsOwnerProfileOrReadOnly
from .serializers import userProfileSerializer 91
```

```
class UserProfileListCreateView(ListCreateAPIView):
queryset=userProfile.objects.all()
serializer_class=userProfileSerializer
permission_classes=[IsAuthenticated]
def perform_create(self, serializer):
user=self.request.user
serializer.save(user=user)
class userProfileDetailView(RetrieveUpdateDestroyAPIView):
queryset=userProfile.objects.all()
serializer_class=userProfileSerializer
permission_classes=[IsOwnerProfileOrReadOnly,IsAuthenticated]
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from .views import UserProfileListCreateView, userProfileDetailView
urlpatterns = [
#gets all user profiles and create a new profile
path("all-profiles",UserProfileListCreateView.as_view(),name="all-
profiles"),
# retrieves profile details of the currently logged in user
path("profile/<int:pk>",userProfileDetailView.as_view(),name="profi
le"),
]
```

Файл license.py

```
from rest_framework.permissions import BasePermission,SAFE_METHODS
class IsOwnerProfileOrReadOnly(BasePermission):
def has_object_permission(self, request, view, obj):
if request.method in SAFE_METHODS:
return True
return obj.user==request.user 92
```

Файл test.py

```
class userProfileTestCase(APITestCase):
profile_list_url=reverse('all-profiles')
def setUp(self):
# создайте нового пользователя, отправив запрос к конечной точке
djoser
self.user=self.client.post('/auth/users/',data={'username':'mario',
'password':'i-keep-jumping'})
# получить веб-токен JSON для вновь созданного пользователя
re-
sponse=self.client.post('/auth/jwt/create/',data={'username':'mario
','password':'i-keep-jumping'})
self.token=response.data['access']
self.api_authentication()
def api_authentication(self):
self.client.credentials(HTTP_AUTHORIZATION='Bearer '+self.token)
# получить список всех профилей пользователей во время аутентифика-
ции пользователя запроса
def test_userprofile_list_authenticated(self):
response=self.client.get(self.profile_list_url)
self.assertEqual(response.status_code,status.HTTP_200_OK)
# получить список всех профилей пользователей, пока запрос пользо-
вателя не прошел проверку подлинности
def test_userprofile_list_unauthenticated(self):
self.client.force_authenticate(user=None)
response=self.client.get(self.profile_list_url)
self.assertEqual(response.status_code,status.HTTP_401_UNAUTHORIZED)
# проверьте, чтобы получить данные профиля аутентифицированного
пользователя
def test_userprofile_detail_retrieve(self):
response=self.client.get(reverse('profile',kwargs={'pk':1}))
# print(response.data)
self.assertEqual(response.status_code,status.HTTP_200_OK)
# заполнить профиль пользователя, который был автоматически создан
с использованием
```

```
сигналов
def test_userprofile_profile(self):
profile_data={'description':'I am a very famous game
character','location':'nintendo world','is_creator':'true',}
re-
sponse=self.client.put(reverse('profile',kwargs={'pk':1}),data=prof
ile_data)
print(response.data)
self.assertEqual(response.status_code,status.HTTP_200_OK)
```