МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему: Розробка мобільного Android-додатку «Mobil» для допомоги водіям

Виконав: студент 4 курсу, групи МФ-41

спеціальність 122 «Комп’ютерні науки»

освітня програма «Інформатика»

Єременко Іван Максимович

Керівник: ст. викладач Бережна Наталія Ігорівна

Рецензент:

Харків 2023

ЗМІСТ

1.ВСТУП

1.1 Постановка задачі

Опис задачі:
Була поставлена ціль розробити застосунок для автомобілістів, який допоможе користувачам зручно керувати даними про своє авто(додавання/редагування), а також надасть можливість отримання різноманітної інформації про своє авто та автосвіт загалом.

Основні технічні вимоги:

1. Можливості незареєстрованих користувачів:
1.1) Надання можливості переглядати новини застосунку;
1.2) Надання можливості слідкувати за новинами автомобільного світу;

2. Для користувачів які зареєструвалися в додатку, попередні можливості будуть розширені:
2.1) Надання можливості додавання власного автомобіля за характеристиками, такими як:
    2.1.1) Марка і модель автомобіля;
    2.1.2) Рік випуску;
    2.1.3) Об’єм двигуна;
    2.1.4) Тип трансмісії;
    2.1.5) Пробіг;
2.2) Надання можливості перегляду власного профілю;
2.3) Надання можливості приймати участь у форумі автомобілістів;
2.4) Надання можливості пошуку автосервісів за містами та оцінками;
2.5) Надання можливості отримувати інформацію по стану автомобіля, та рекомендації щодо його обслуговування.

2.6) Надання можливості проходити тестування зі знань правил дорожнього руху, та отримувати оцінку своїх знань.

Підсумовуючи застосунок має взаємодіючи з користувачами надавати необхідний функціонал для комфортного користування та забезпечити онлайн обговорення проблем, пошуку порад та отримання досвіду від інших користувачів. В той же час за допомогою рейтингів та відгуків інших користувачів, додаток має надати можливість оцінити якість та надійність ремонтних станцій.

1.2 Обґрунтування актуальності теми та можливостей застосування додатку

Ми живемо в час швидкого розвитку технологій. Більшість нових технологій допомагають нам в повсякденному житті, багато хто вже не уявляє свого життя без них, тому що ці технологію спрошують наше з вами життя. З кожним днем з’являється все більше застосунків, які беруть частину наших завдань на себе, як приклад - застосунки для нагадування важливих для нас подій, або які допомагають нам стежити за станом нашого здоров'ям. Всі вони економлять наш з вами час і дають змогу отримувати більше насолоди від життя. Саме через це ми і вирішили створити застосунок для автомобілістів. Ми прагнемо допомогти кожному автомобілісту в отриманні нових знань щодо автомобіля, скоротити його час пошуку надійного та якісного автосервісу, а також, звичайно, допомогти у своєчасному обслуговуванні улюбленого транспортного засобу.

Актуальність створення даного застосунку для автомобілістів, або просто автолюбителів, обґрунтовується тим, що в наш час транспортний засіб став невід'ємною частиною нашого життя. З кожним новим днем на дорогах країн світу з'являється все більше транспортних засобів і кожен з них потребує своєчасного обслуговування. Ми ж в свою чергу пропонуємо користувачам нашого застосунку ефективне нагадування щодо обслуговування свого транспортного засобу, що в свою чергу забезпечує додаткову надійність їхнього транспортного засобу і в той самий момент життя.

Ефективне нагадування щодо обслуговування, також надає можливість заощадити свій час і бути впевненим в надійності власного авто. Разом з тим наш застосунок надасть можливість користувачам здобувати нові знання щодо автомобілів, а також отримувати допомогу від інших

користувачів у якихось цікавих для них питаннях. Кожен з користувачів отримає можливість знайти для себе щось нове та цікавляче саме його.

1.3 Огляд аналогів та аналіз недоліків

На даний момент існують наступні аналогічні застосунки:

- *Застосунок Car Scanner ELM OBD2* - цей застосунок розроблений для того, щоб допомагати визначати автомобілістам неполадки їхнього авто. Він допоможе знайти неполадки і визначить в якій саме деталі авто є неполадки.
- *Застосунок ПДР України* - цей застосунок розроблений для того, щоб допомогти автомобілісту вивчити правила дорожнього руху, а також надає можливість проходження тестів і оцінки своїх знань правил.
- Застосунок My fishka - цей застосунок розроблений для того, щоб дати можливість постійному відвідувачу АЗС Okko накопичувати відсотки з витрат на пальне.

Розглянувши застосунки вище, ми зробили висновок, що кожен із них вузьконаправлений. Для користування кожним із них ви маєте мати при собі смартфон і бути авторизованим користувачем. При цьому, ви маєте встановити кожен із цих додатків і кожен раз перемикатися між ними, через що ви витрачаєте більше часу на роботу з ними.

1.4 План розвитку застосунку у майбутньому

Проект розробки застосунку для автомобілістів, має декілька шляхів розвитку у майбутньому:

- Удосконалення існуючого застосунку;
- Розробка додатку, які будуть зроблені виключно під якісь системи, такі як Android, IOS.

Розглянемо перший варіант розвитку застосунку:

Застосунок для автомобілістів має значний потенціал розвитку у майбутньому.

Першочергово, можна додати більше функціональності застосунку, як приклад:

- По-перше у наш застосунок можна додати навігацію, для більш зручного продумування маршруту поїздки;
- По-друге можна додати рейтинг АЗС з відгуками користувачів
- По-третє у застосунок можна додати монетизацію, а разом і з нею; додати додатковий функціонал, який буде робити систему більш зручною і функціональною;

При цьому завдяки системі монетизації можна буде в майбутньому домовлятися з різноманітними джерелами та компаніями, які будуть надавати додаткові знижки для користувачів нашого застосунку.

З розвитком застосунку також можна буде додати різноманітні форуми, по різним темам, таким як:

- Допомога на дорозі;
- Обмін інформацією;
- Поради по обслуговуванню різноманітних марок авто;
- Види палива, тощо;

До цих нововведень також можна додати співпрацю з різними інтернет ресурсами, які спеціалізується на продажі товарів для авто, і додати функціонал, який буде визначати найбільш гарні ціни на ту чи іншу запчастину і надавати інформацію про неї користувачу.

Тепер перейдемо до розгляду второго варіанту розвитку застосунку, а саме створення додатку під визначену систему, таку як Android, або IOS:

Першочергово треба буде поєднати користувачів, які вже зареєстровані в застосунку і додати їх до додатку, при цьому зберегти основні функціональні і стилістичні вимоги. Після цього можна розширювати паралельно застосунок і додаток, якщо розширення функціоналу застосунку ми вже розглянули, давайте зробимо це і для додатку:

- По-перше ми можемо зробити більш зручну онлайн-підтримку, додавши до додатку спеціальні онлайн чати і людей, які будуть мати всю необхідну інформацію залежно від тематики чату;
- По-друге можна додати відстеження знижок на різних інтернет магазинах запчастин, та розсилка їх через спеціальних ботів як в особисті повідомлення, так і на пошту;

Підсумовуючи ми розуміємо, що наш застосунок має багато варіантів розвитку у майбутньому, що знову ж таки підкреслює його актуальність.

1.5Огляд та обґрунтування використаних технологій

Враховуючи, що серверна частина розроблена на Java Spring Boot з використанням технологій Rest API, мій вибір пав на React JS.

Почнемо з JavaScript, цього року найпопулярнішою мовою програмування став JavaScript і набравши 19% проектів комерційних проектів.

Отже, давайте подивимося основні переваги JavaScript:

- Виконання логіки на стороні клієнта забезпечує більш швидку взаємодію з користувачем. Коли код працює безпосередньо в браузері, потреба у викликах сервера абстрагується, отже, скорочується час завантаження. Навіть за наявності сервера той факт, що JS є асинхронним, означає, що він може спілкуватися з сервером у фоновому режимі, не перериваючи взаємодії користувача, що відбувається у інтерфейсі.
- JavaScript стоїть за будь-яким адаптивним веб-дизайном. Дедалі більше розробникам потрібно адаптувати свій дизайн для декількох браузерів і пристроїв.Використовуючи JavaScript розробники можуть поєднувати HTML5, CSS3 і JavaScript в одній кодовій базі.
- З самого початку JavaScript приніс інтерактивність інтерфейсу користувача в Інтернет. Тепер він робить те саме для програм усіх типів, допомагаючи розробити найбільш привабливий UX. Сьогодні такі фреймворки, як React.js, Vue.js, виводять переходи та анімацію на новий рівень.

Але повернемося до React JS який в той самий час є одним із найбільш популярних фреймворків для JavaScript та одночасно чудово підтримує технологію Rest API.

Тож давайте подивимося основні переваги React JS:

- Максимально зрозумілий синтаксис, основні задачі виконуються на синтаксисі JavaScript, Html, CSS
- Функціональний підхід до написання компонентів.
- Основним з найважливіших підходів, які пропагує React, н є принцип Immutability. Тобто все, що робиться в межах компонентів, в жодному випадку не має мутувати стану або даних, а лишень створювати нові екземпляри на основі вже наявних.
- Компонентний підхід до написання застосунку
- Підхід до створення екосистеми застосунку — від меншого до більшого. Тобто тут лише ви вирішуєте, що потрібно брати з додаткових бібліотек і пакетів, а що — ні.
- Стан DOMу зберігається й обчислюється у віртуальному DOMі, що робить React дуже швидким в плані рендерінгу.
- Регулярно оновлюється і вдосконалюється.
- Підтримується Facebook.
- Підхід до створення екосистеми застосунку — від меншого до більшого. Тобто тут лише ви вирішуєте, що потрібно брати з додаткових бібліотек і пакетів, а що — ні.

Ось деякі з переваг React JS, до цього також можна додати велике ком'юніті, а також зразкову документацію та велику кількість в вільному доступі рекомендацій, щодо написання коду.

Тепер розглянемо обрану базу даних, нами була обрана реляційна база даних, яка називається PostgreSQL. Використання PostgreSQL як бази даних для додатку також має свої переваги і обґрунтування.PostgreSQL технічно є системою керування об’єктно-орієнтована реляційною базою даних. Це означає, що вона має такі ж реляційні можливості, як і Об'єктно-реляційна база даних, але додатково має деякі об’єктно-орієнтовані функції.

Іншою сферою, де PostgreSQL виділяється серед інших систем реляційних баз даних, є дотримання стандартів SQL.

Стандарти SQL були розроблені групами ANSI та ISO з метою визначення мінімальних вимог до функціональності та сумісності для реалізацій SQL. Хоча специфікації, надані цими органами, призначені для визначення функцій, які мають надавати системи SQL, через складність і тривалий розвиток мови суворе дотримання не завжди можливо. Згідно з документацією PostgreSQL, наразі жодна база даних не задовольняє всім вимогам, викладеним у специфікації.

ACID — це ініціалізм в інформатиці, який означає атомарність, послідовність, ізоляцію та довговічність. Вони представляють ключові гарантії, які повинні підтримувати транзакції бази даних, щоб уникнути помилок дійсності та підтримувати цілісність даних.

Відповідність ACID є основною проблемою для реляційних баз даних, оскільки вона представляє типові очікування щодо зберігання та модифікації високо структурованих даних. Нереляційні бази даних часто намагаються відповідати своїм власним стандартам, які часто представлені конкуруючим ініціалізмом BASE, який означає в основному доступний, м’який стан і можливу узгодженість. PostgreSQL надає широкий спектр функцій та можливостей, що дозволяють ефективно управляти та оптимізувати роботу з даними. Вона підтримує розширення за допомогою функцій, типів даних, мов програмування та інших компонентів. Це дозволяє адаптувати базу даних до специфічних потреб додатку та забезпечити гнучкість розробки.

2. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ДОДАТКУ

2.1 Опис функціональності системи

Функціонал системи був розроблений для адміністратора, незареєстрованого користувача та зареєстрованого користувача.

Розглянувши функціонал застосунку для незареєстрованого користувача, ми вирішили, що доцільно буде додати наступний функціонал:

- Система має давати змогу переглядати основний контент застосунку, такий як новини застосунку та новини автосвіту загалом.
- Також система має давати змогу зареєструватися, щоб отримати більш розширений функціонал.

Use case діаграма для незареєстрованого користувача:

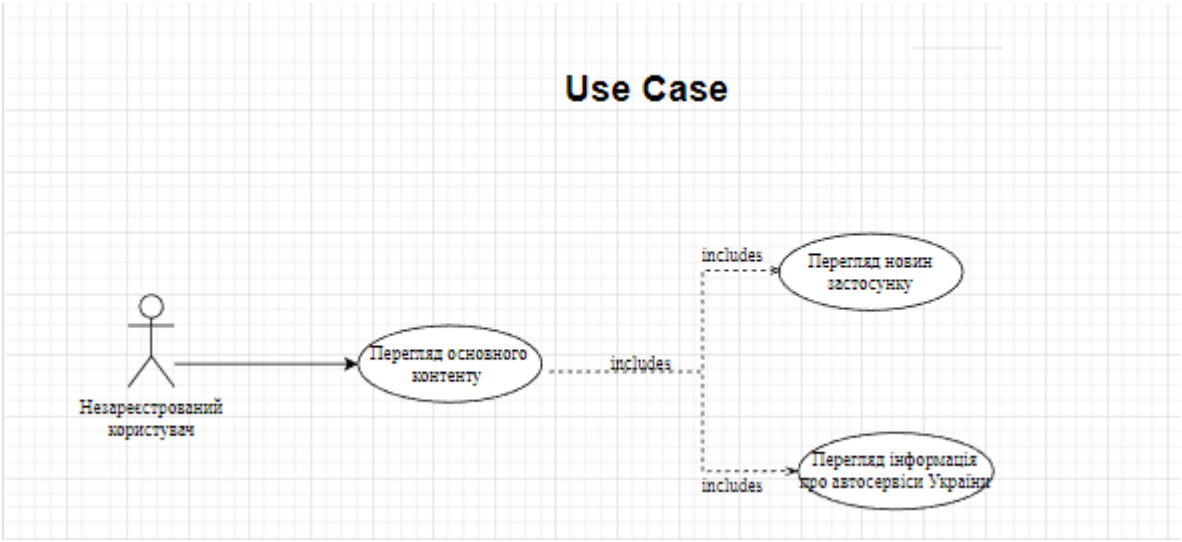

*Рис 1 Use Case(*незареєстрованого *користувач)*

Після цього був розглянутий функціонал застосунку для зареєстрованого користувача. Його функціонал мав отримати ті самі можливості як і незареєстрований користувач, а також додатково функціонал мав бути розширений, тому доцільно було розширити його таким чином:

- Насамперед зареєстрований користувач має отримати можливість авторизуватися, що в подальшому має надати йому можливість взаємодіяти з додатковим функціоналом.
- Після авторизації користувачу має стати достнупним його профіль, в якому він отримає доступ до своїх власних даних.
- Також має стати доступна форма додавання автомобіля за характеристиками і після додавання інформація про власне авто має бути доступна для перегляду в профілі.
- Користувач також має отримати можливість спілкуватися в онлайн-форумі.
- Система також має надати можливість створювати новини автосвіту, які будуть відображені на основній сторінці сайту.
- А також в окремому вікні користувач має отримати доступ до перевірки знань ПДР.Use Case діаграма для зареєстрованого

користувача

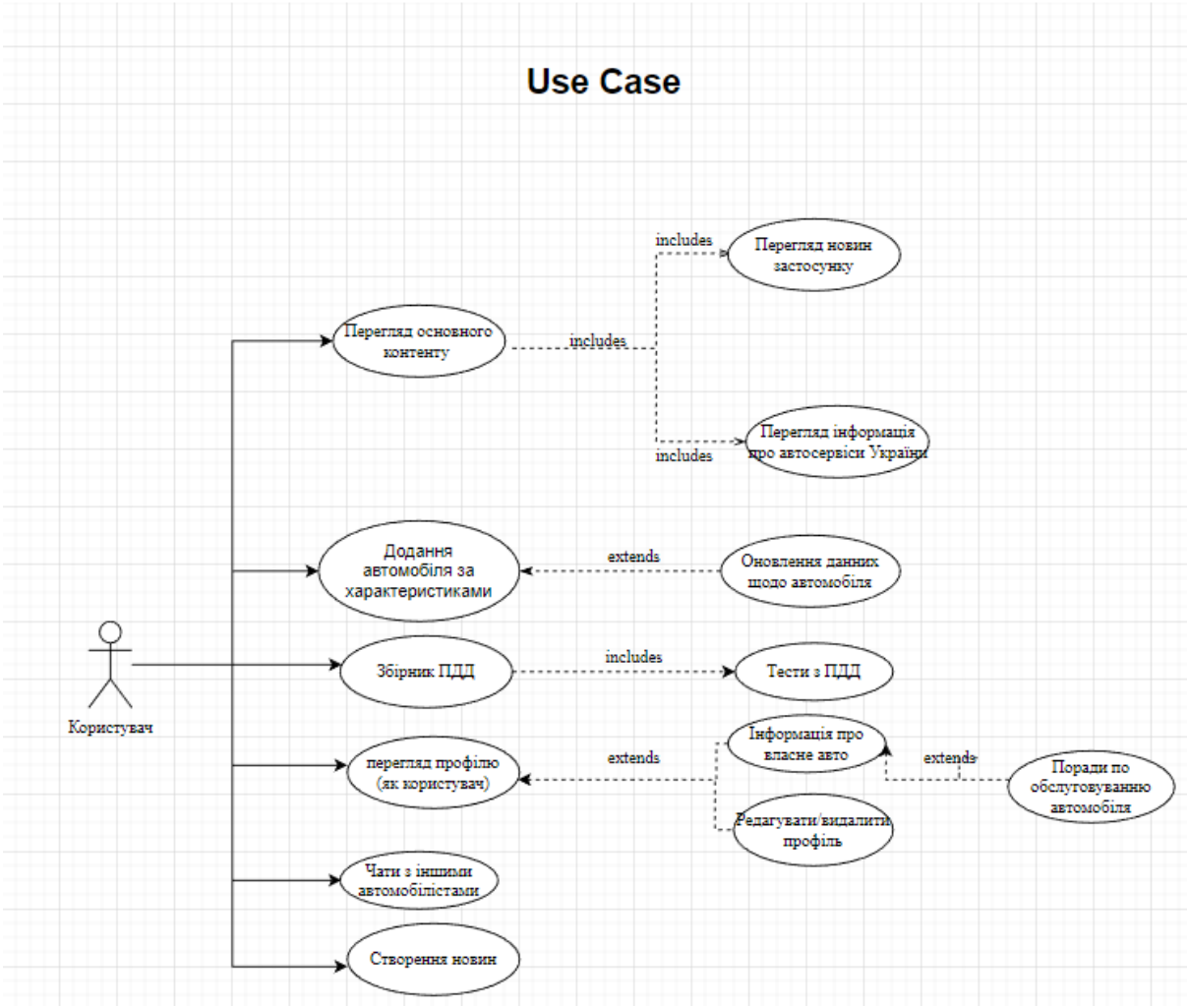

*Рис 2 Use Case(*зареєстрований *користувач)*

Також в нашому застосунку мають бути присутні адміністратори, які отримають власний функціонал, а саме:

- Оновлення інформації.
- Перегляд профілів користувачів.
- Створення/редагування новин застосунку.

Use Case діаграма для адміністратора:

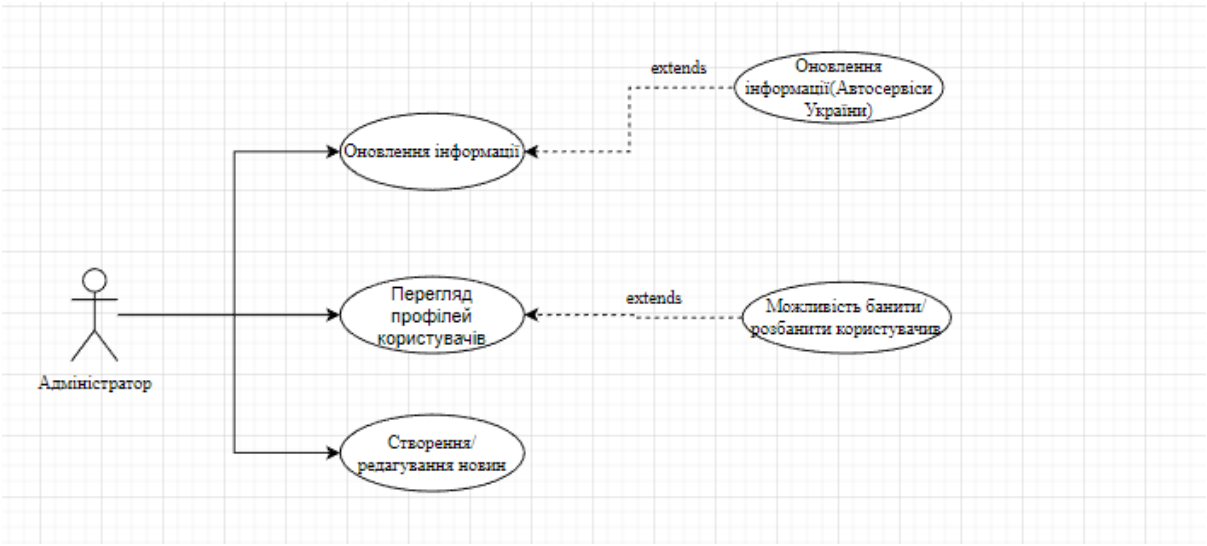

*Рис 3 Use Case(*адміністратор*)*

2.2 Покроковий опис роботи застосунку

Посилаючись на пункт 2.1 був розроблений покроковий опис роботи застосунку. Знаючи всі функціональності системи, робота кожної функціональності була продумана для кожного користувача, почнемо з незареєстрованого:

При вході у застосунок незареєстрований користувач має змогу переглянути основний контент, або перейти до вікна реєстрації/авторизації.

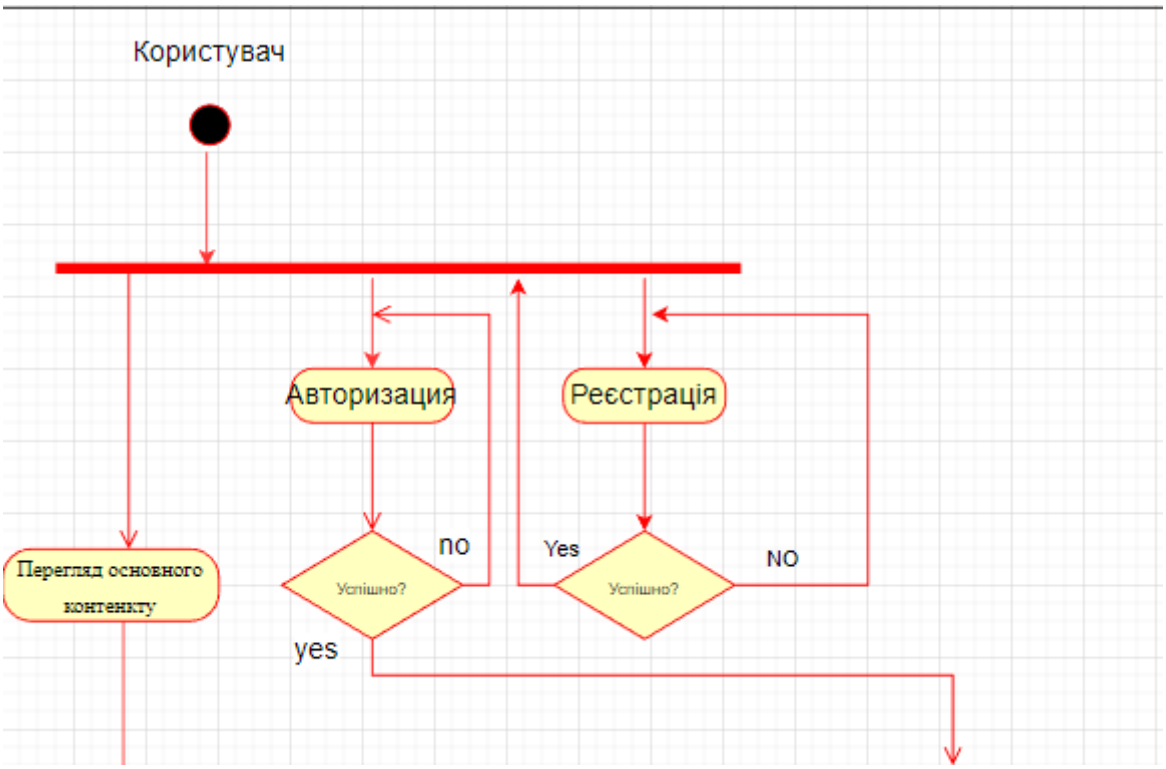

Рис 4 Activity(для неавторизованного користувача)

Після цього подивимося роботу кожної функціональності для користувача, який зареєстрований у системі:

При вході у застосунок зареєстрований користувач отримує туж саму функціональність як і незареєстрований, щоб перейти до функціональності зареєстрованого, йому потрібно авторизуватися у застосунку. Натискаючи на авторизації, він отримує вікно, де вводить дані свого аккаунту, при

успішності вводу, він стає авторизованим користувачем і отримує можливість користуватися усім функціоналом зареєстрованого користувача, таким як додавання автомобіля за характеристиками, перегляд профілю, форум з іншими користувачами, тощо.

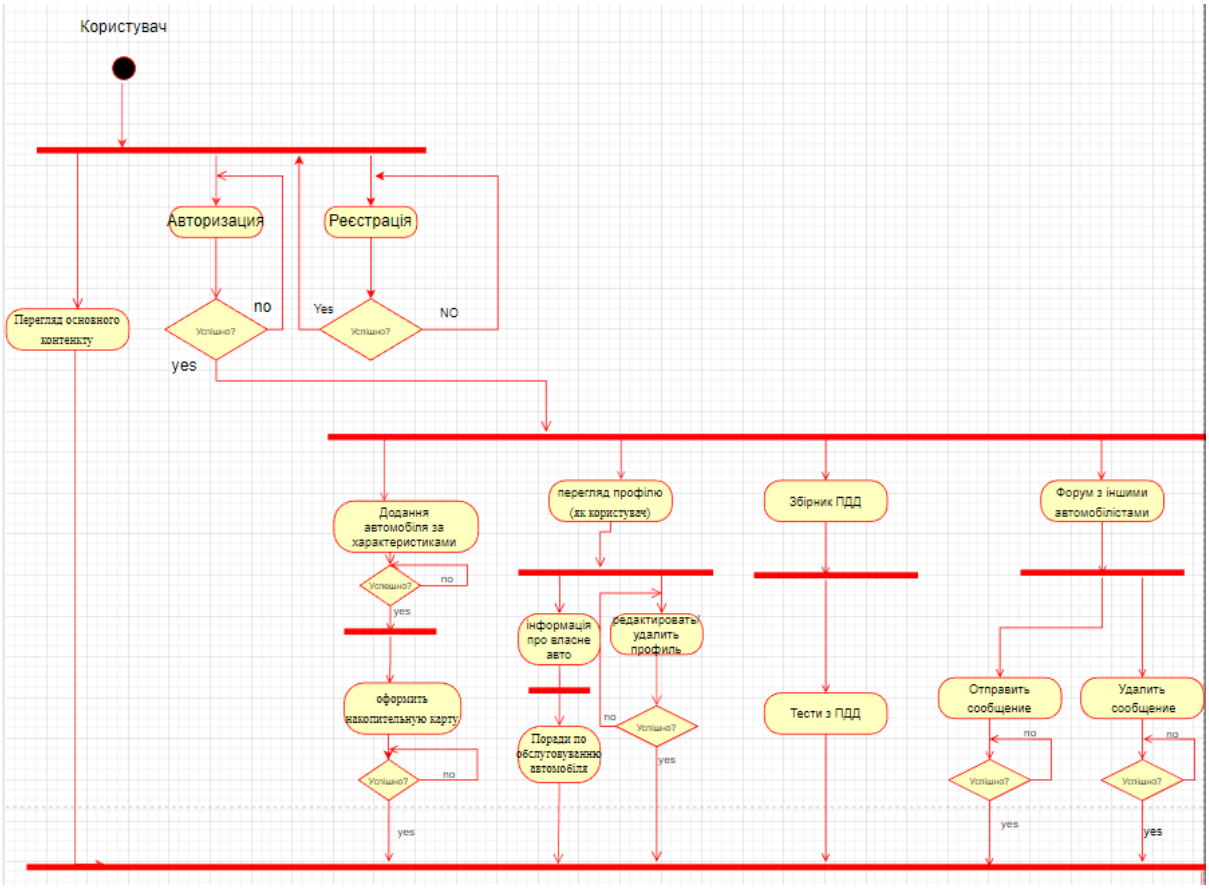

Рис 5 Activity(для авторизованного користувача)

Також була продумана робота кожної функціональності для адміністратора.

Адміністратор визначений у системі автоматично і при авторизації автоматично отримує функціонал, який був розроблений під нього, а саме створення/редагування новин, перегляд профілів користувачів, оновлення інформації в застосунку.

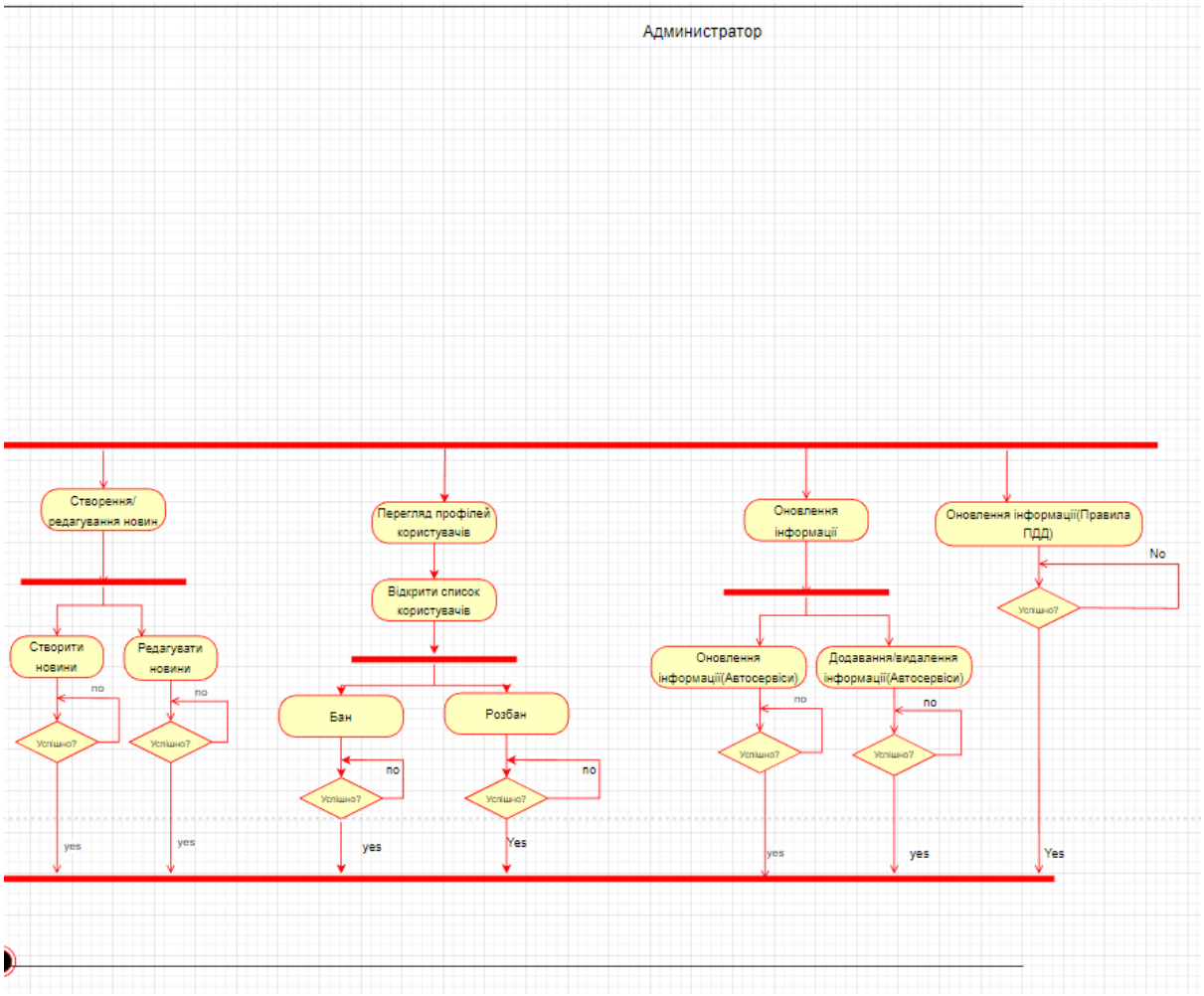

Рис 6 Activity(для адміністратора)

2.3 Проектування макету

Для проектування макету доцільно обрати інструмент спільного проектування інтерфейсу Figma.

Розробляючи макет перше, з чого я почав це навігаційна панель, яка буде присутньою на кожній сторінці застосунку. Головні вимоги, які були поставленні при розробці макету навігаційної панелі, це зручність та зрозумілість. Тому основна навігація мала включати у себе:

- Логотип з лінком на основну сторінку застосунку;
- Лінк на сторінку про застосунок;
- Лінк на сторінку з новинами застосунку;
- Лінк на сторінку з форумом(до нього мають доступ тільки авторизовані користувачі);
- Якщо користувач не авторизований:
    - Лінк на сторінку авторизації;
    - Лінк на сторінку реєстрації;
- Якщо користувач авторизований:
    - Лінк на особистий аккаунт користувача:

Таким чином виглядає навігаційна панель:

Рис 7 навігаційна панель(для неавторизованного користувача)

Після того як була розроблена навігаційна панель, був розроблений макет нижньої частини застосунку, який так само як і навігаційна панель буде доступна всім користувачам. В нижній частині сайту були поміщені лінки на різні соціальні мережі.

Таким чином виглядає нижня частина застосунку:

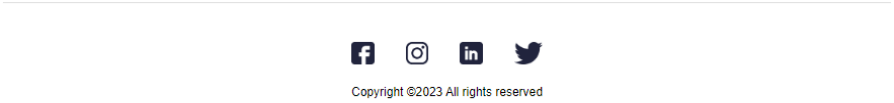

Рис 8 нижня частина застосунку

Після цього було розроблено дві секції основного контенту головної сторінки. Перша включала в себе опис застосунку, а друга новини застосунку та автосвіту загалом.

Таким чином виглядає секція опис застосунку:

Рис 9 секція опис застосунку

Таким чином виглядає секція новини застосунку та новини автосвіту загалом:

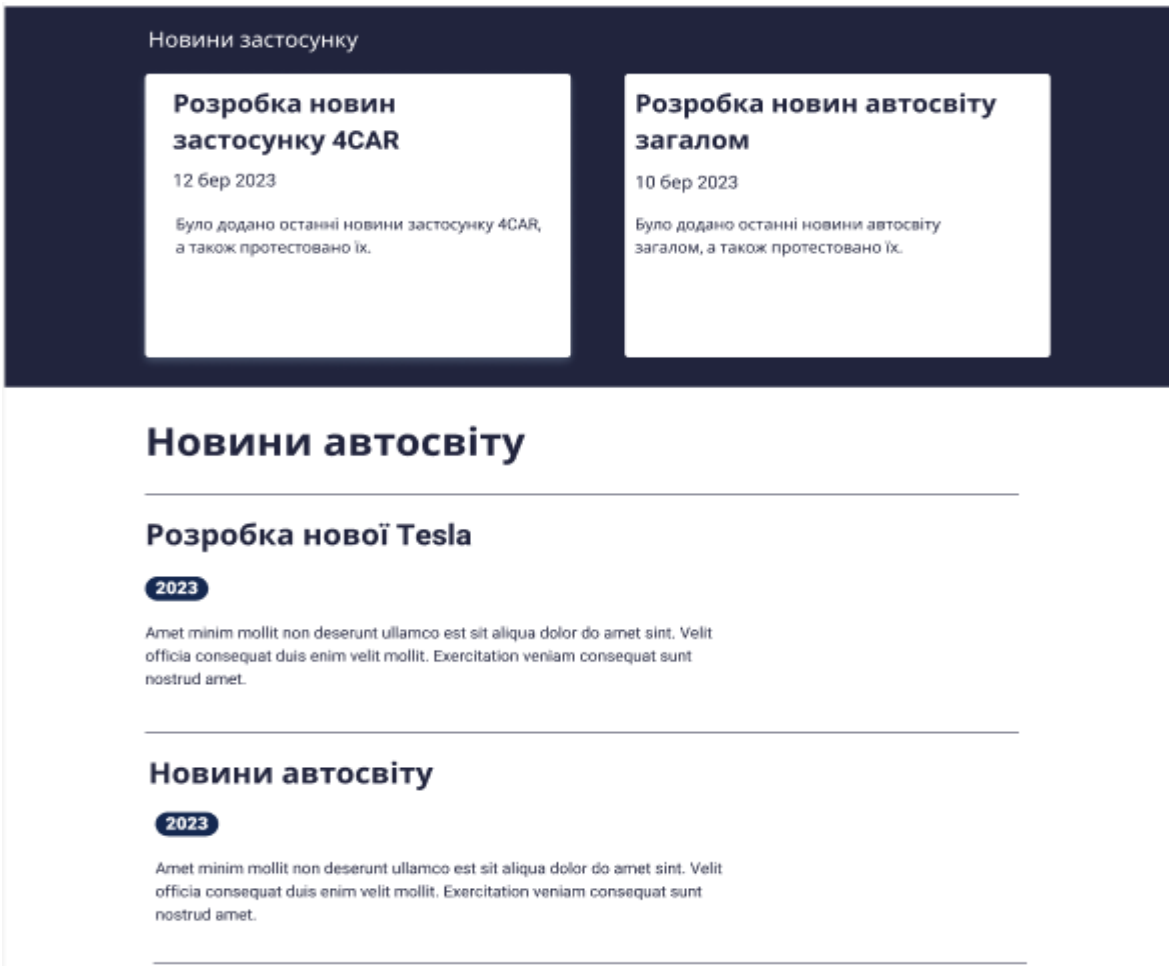

Рис 10 секція новини застосунку та автосвіту загалом

Після того, як всі частини макету були спроектовані, вони були об'єднані в одну секцію(Голован сторінка).

Рис 11 макет головної сторінки застосунку

Після того, як головна сторінка була завершена ми почали розробку макетів для усіх інших сторінок додатку. Розділимо їх на деякі групи:

1. Група форм для входу, реєстрації, додавання міста, додавання новин, додавання автомобіля;
2. Сторінки з новинами;
3. Форум;
4. Тести.

Розглянемо першу групу сторінок, їх дизайн мав бути зручним та зрозумілим, тому ці сторінки мали вигляд полів, які потрібно заповнити та кнопкою(кнопками).

Таким чином виглядає група форм для входу, реєстрації, додавання міста, додавання новин, додавання автомобіля:

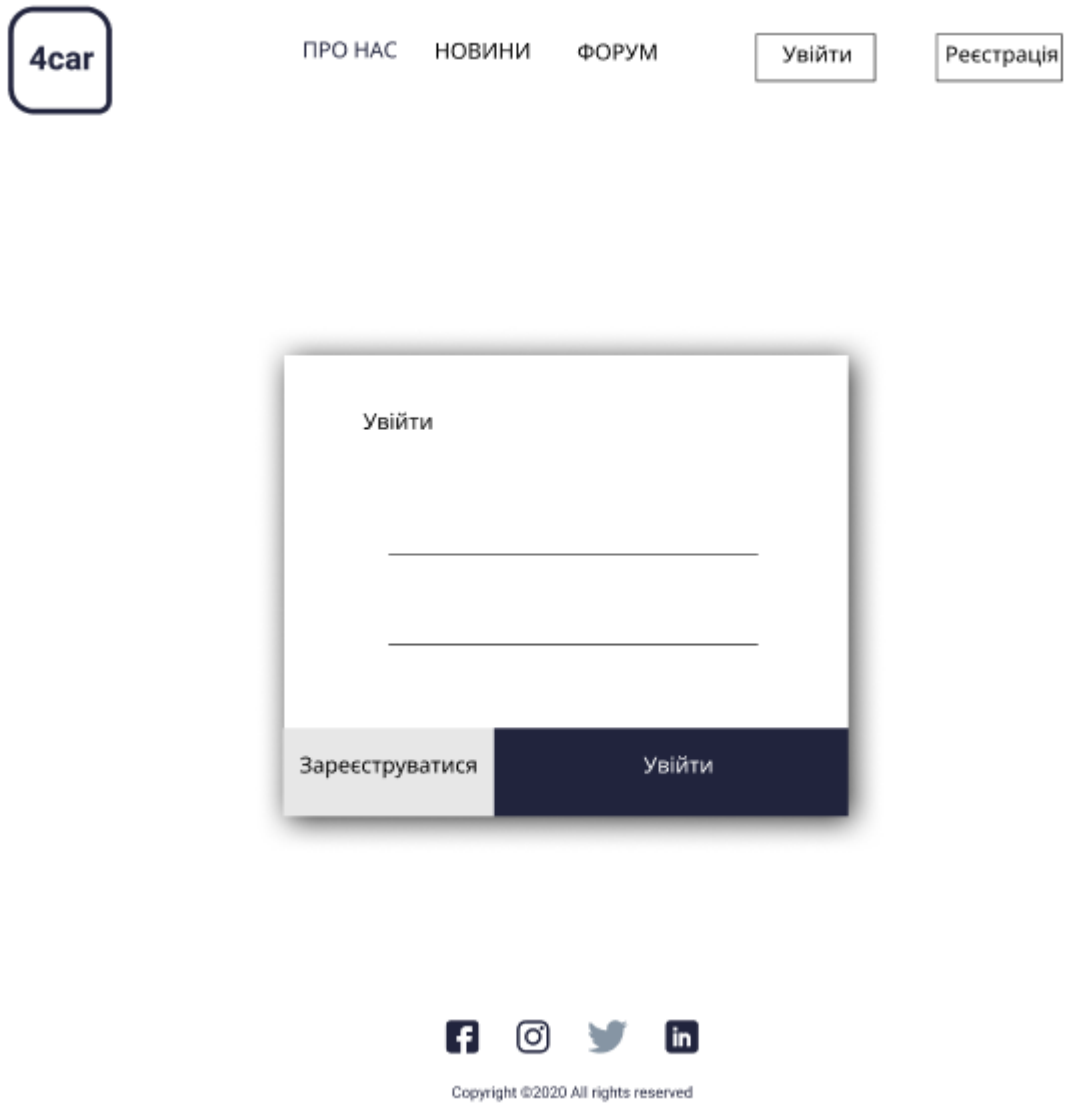

Рис 12 макет форм для застосунку

Всі подальші макети форм були створені на основі цього макету.

Далі було розроблено макет для другого типу сторінок, а саме сторінки з новинами. Основні вимоги до цієї сторінки - це максимально стисло донести інформацію до користувачів застосунку при цьому зазначити дату додання цієї новини. Макет сторінки з новинами має наступний вигляд:

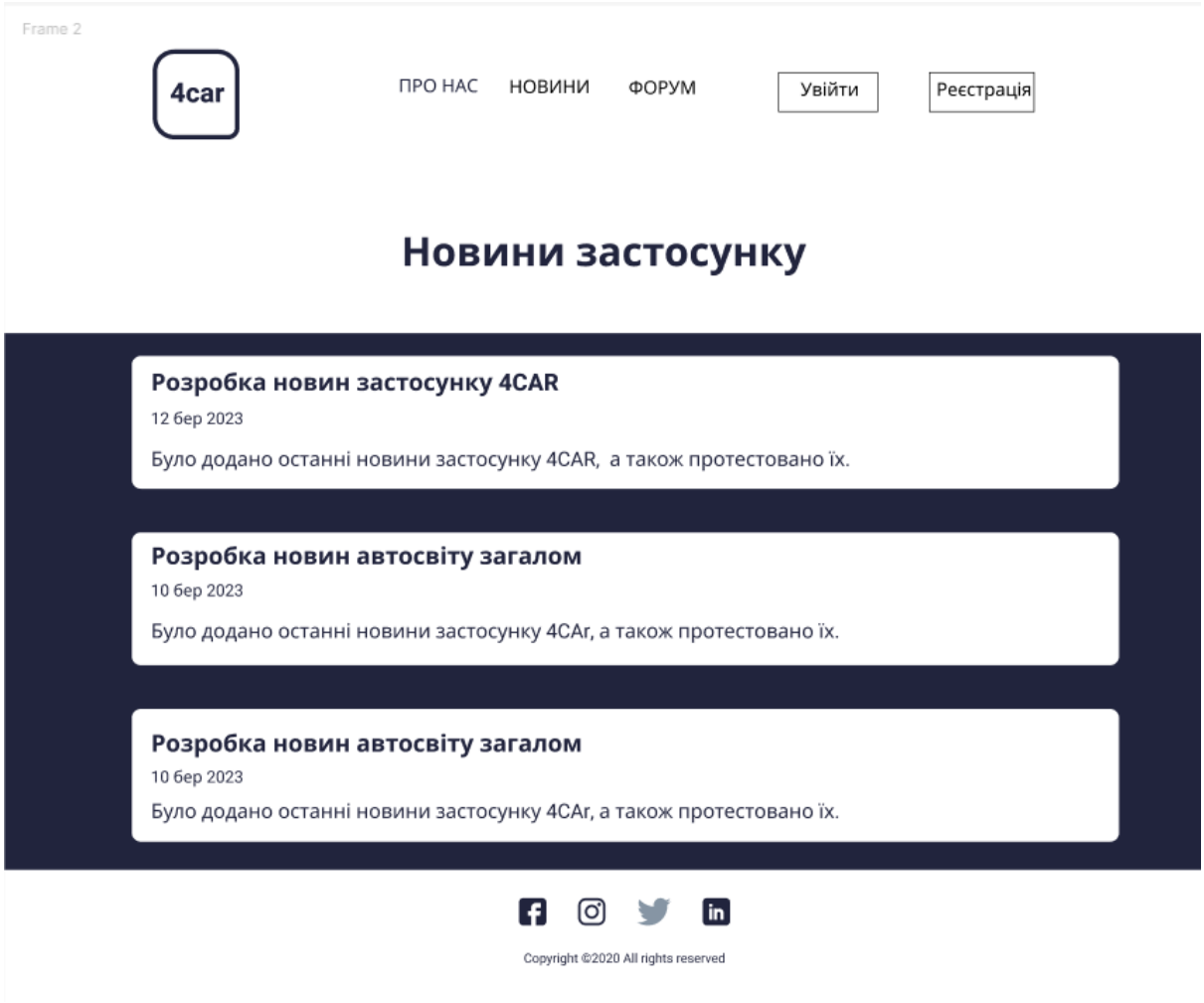

Рис 13 макет сторінки новини застосунку

Після було розроблено макет для третього типу сторінок, а саме форумів.

Ця сторінка мала мати наступні характеристики, бути зручною, але в той самий час надавати максимальну кількість інформації і бути інтуїтивно зрозумілою. Отже в підсумку маємо наступний макет сторінки форуму:

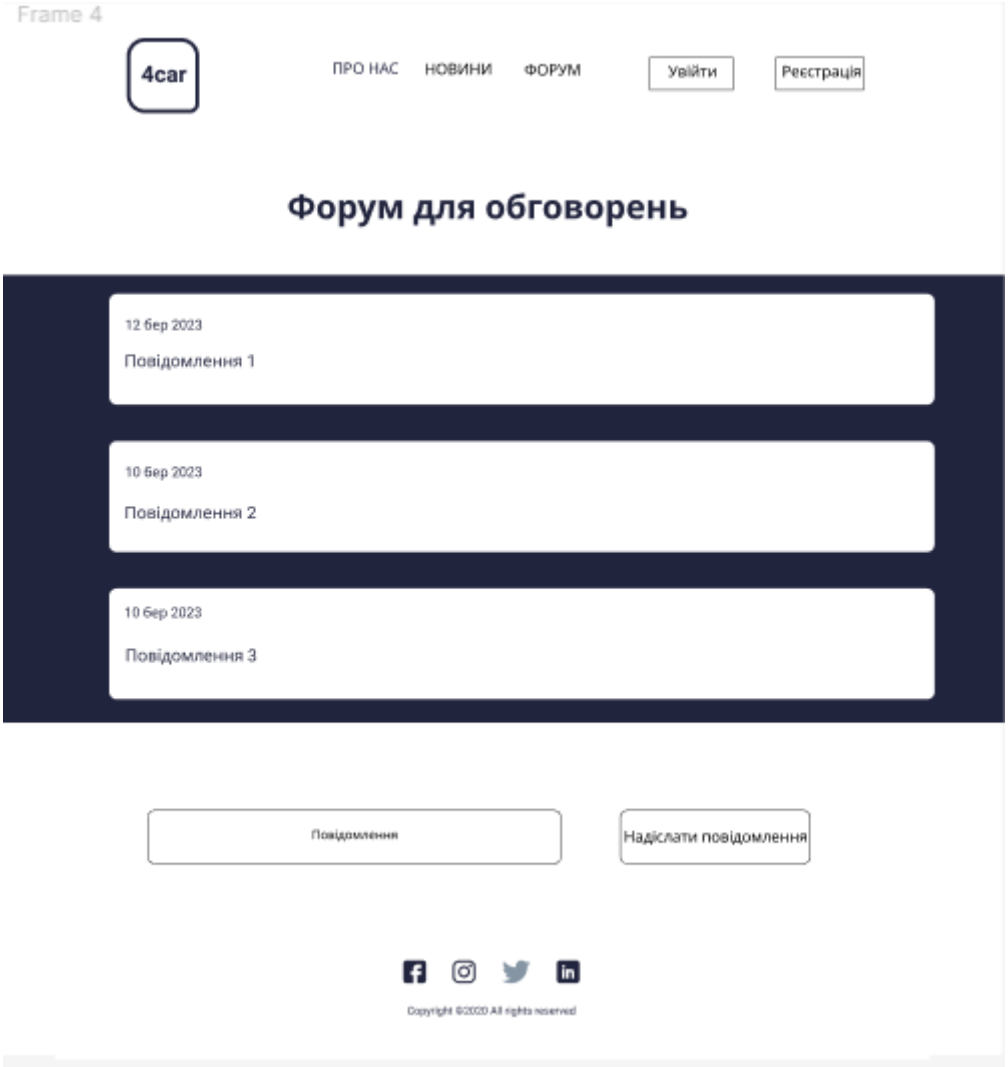

Рис 14 макет сторінки форум

Отже підводячи підсумки було розроблено макет для всіх основних сторінок застосунку, враховуючи потреби користувачів.

2.4 Основні налаштування проекту

При налаштуванні проекту для розробки клієнтської частини застосунку, було проведено наступні налаштування:

- Налаштування доступу (cors) на серверній частині;
- Налаштування proxy та axios на клієнтській частині;
- Завантаження необхідних шрифтів;

Налаштування Cors виконувалось за допомогою springframework анотації, а саме було додано анотацію @CrossOrigin(“*”) у контроллерах серверної частини.

```
@RestController
@CrossOrigin("*")
public class CarController {
```

Рис 15 налаштування cors

Налаштування proxy та axios вже виконувалось на клієнтській частині застосунку. Proxy було додано в Package.json параметром “proxy”:”данні”, а axios був попередньо встановлений у терміналі командою npm install axios. Після цього був створений файл Axios.js в директорії src до якого потім були прописані наступні налаштування:

```
import axios from 'axios';

const instance = axios.create({
    baseURL: "http://localhost:8080"
});

export default instance;
```

Рис 16 налаштування axios

Шрифти в свою чергу були взяті Google Fonts. У всьому нашому застосунку використовувався шрифт Heebo з стилями “Normal” , “Bold”, “Medium”.

2.5 Верстання зовнішньої частини макету

Верстання макету було розпочате з основної сторінки, як і в пункті 3.1 кожна частина макету була прописана в окремих компонентах таких як Navbar, Footer,News, Main. Після цього була імпортована до окремого компоненту Main Page. Структура мала такий вигляд:

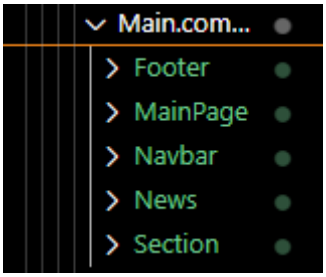

Рис 17 структура головної сторінки

Після цього кожна частина макету була зверстанна за допомогою Html та CSS. Важливо також зазначити, що при роботі з React для написання Html коду ми використовуємо jsx. JSX - це JavaScript XML , вын дає нам змогу писати Html код у React.js. Отже повернемося до верстання, першим був зверстаний Navbar.

```
return(
    <header className="header">
        <div className="container">
            <div className="header-nav">
                <div className="nav-logo">
                    <Link  to='/' classNameName="nav-logo"><img src={logo} alt="logo"/></Link>
                </div>
                    {/* <!-- Navigation --> */}
                <nav className="nav">
                    <ul className="nav-list">
                        <Link to='/about' classNameName="nav-link"><li className="nav-item">ПРО НАС</li></Link>
                        <Link to='/newsPrint' classNameName="nav-link"><li className="nav-item">НОВИНИ</li></Link>
                        <Link to='/forum' classNameName="nav-link"><li className="nav-item">ФОРУМ</li></Link>
                    </ul>
                </nav>
                {accountId == null?
                <div className="sign">

                        <Link to='/login' classNameName="sign-in">
                            <button classNameName="btn">Увійти</button>
                        </Link>
                        <p></p>
                        <Link to='/Registration' classNameName="sign-in">
                            <button classNameName="btn">Реєстрація</button>
                        </Link>

                </div>
                :
                <div className="account">
                    <div >
                        <h3 className="acount-text">
                            Аккаунт:
                        </h3>
                    </div>

                    <div>
                        <Link to={"/account"}>
                            <h3 className="acount-text">{userName}</h3>
                        </Link>
                    </div>
                </div>
                }
            </div>
        </div>
    </header>
```

Рис 18 Html код навігаційної панелі

Після навігаційної панелі за допомогою Html були зверстані всі наступні компоненти головної сторінки. В кінці, всі вони були поміщені до головного компоненту, назва якого була MainPage.jsx. За допомогою import кожен з компонентів був переданий до MainPage де після цього був

експортований.

```
import React from "react";
import Navbar from '../Navbar/Navbar'
import Footer from '../Footer/FooterPage'
import Section from '../Section/Section'
import Sections from '../News/Sections'
import MainContent from "../Section/MainContent/MainContent";


export const MainPage = () => {

    return(
    <div>
        <Navbar/>
        <MainContent/>
        <Sections/>
        <Section/>
        <Footer/>
    </div>
    )
}

export default MainPage;
```

Рис 19 Компонент MainPage

Так само було зверстано всі наступні сторінки застосунку. Важливо також зазначити, що Navbar та Footer присутні на кожній сторінці нашого застосунку. Отже ось такі зверстані компоненти вийшли в підсумку:

```
import Navbar from '../Main.component/Navbar/Navbar'
import Footer from '../Main.component/Footer/FooterPage'
import Registration from './RegistrationScreen'

export const RegistrationPage = () => {

    return(
    <div>
        <Navbar/>

        <Registration/>

        <Footer/>
    </div>
    )
}

export default RegistrationPage;
```

Рис 20 Компонент RegistrationPage

```
import Navbar from '../Main.component/Navbar/Navbar'
import Footer from '../Main.component/Footer/FooterPage'
import Login from './LoginScreen'

export const LoginPage = () => {

    return(
    <div>
        <Navbar/>

        <Login/>

        <Footer/>
    </div>
    )
}

export default LoginPage;
```

Рис 21 Компонент LoginPage

```jsx
import Navbar from '../Main.component/Navbar/Navbar'
import Footer from '../Main.component/Footer/FooterPage'
import AboutUs from './AboutUs'

export const AboutPage = () => {

    return(
    <div>
        <Navbar/>

        <AboutUs/>

        <Footer/>
    </div>
    )
}

export default AboutPage;
```

Рис 22 Компонент AboutPage

```jsx
import Navbar from '../Main.component/Navbar/Navbar'
import Footer from '../Main.component/Footer/FooterPage'
import Forum from './Forum'

export const ForumPage = () => {

    return(
    <div>
        <Navbar/>

        <Forum/>

        <Footer/>
    </div>
    )
}

export default ForumPage;
```

Рис 23 Компонент ForumPage

```
1   import Navbar from '../Main.component/Navbar/Navbar'
2   import FooterPage from '../Main.component/Footer/FooterPage'
3   import NewsPrint from './NewsPrint'
4
5   export const NewsPage = () => {
6
7       return(
8       <div>
9           <Navbar/>
10
11          <NewsPrint/>
12
13          <FooterPage/>
14      </div>
15      )
16  }
17
18  export default NewsPage;
```

Рис 24 Компонент ForumPage

У підсумку у нас були зверстані основні сторінки нашого застосунку. Що давало нам змогу перейти до іншого етапу розробки застосунку.

2.6 Реалізація роутингу між компонентами

Після того, як компоненти застосунку були зверстанні, потрібно було налаштувати їх роутинг. Це було зроблено за допомогою react-router-dom. Для роботи з ним попередньо ми маємо використовуючи термінал, за допомогою команди npm install react-router-dom, встановити його. Після цього ми отримали можливість імпортувати роутери до нашого проекту та зробили налаштування роутингу між компонентами. Ось такий вигляд він має:

```jsx
import React, { useContext, useState } from "react";
import {BrowserRouter , Routes, Route} from 'react-router-dom'
import LoginScreen from './screens/Authorization/NavbarPlusLogin';
import RegistrationScreen from './screens/Authorization/NavparPlusRegitration';
import Cities from './screens/Cities/AddCitites';
import MainPage from "./screens/Main.component/MainPage/MainPage";
import AboutPage from "./screens/AboutUs/AboutPage";
import NewsAdd from "./screens/News.component/NewsAdd";
import NewsPrintPlusNavbar from "../components/screens/News.component/NewsPrintPlusNavbar"
import CarAdd from "./screens/Car.component/CarAdd";
import CarPrint from "./screens/Car.component/CarPrint";
import ForumPlusNavbar from '../components/screens/Forum/ForumPlusNavbar'
import { Context } from "../Context";
import Account from "./screens/Account/Acount";

const Router = ()=>{
    const [accountId, setAccountId] = useState(null)

    return(
        <BrowserRouter>
            <Context.Provider value={{accountId, setAccountId}}>
                <Routes>

                    <Route element={<MainPage/>} path="/"/>
                    <Route element={<Account/>} path="/account"/>
                    <Route element={<LoginScreen/>} path="/login"/>
                    <Route element={<Cities/>} path="/addcity"/>
                    <Route element={<RegistrationScreen/>} path="/Registration"/>
                    <Route element={<AboutPage/>} path="/about"/>
                    <Route element={<NewsAdd/>} path="/newsAdd"/>
                    <Route element={<NewsPrintPlusNavbar/>} path="/newsPrint"/>
                    <Route element={<CarAdd/>} path="/carAdd"/>
                    <Route element={<CarPrint/>} path="/carPrint"/>
                    <Route element={<ForumPlusNavbar/>} path="/forum"/>
                    <Route path="*" element={<div>Page not found</div>}/>

                </Routes>
            </Context.Provider>
        </BrowserRouter>
    )
}

export default Router;
```

Рис 25 Роутинг між компонентами

Після налаштування роутингу ми отримали можливість переходу з одного компоненту застосунку на інший. Для налаштування переходу було використано спеціальні теги jsx розмітки <Link to= ‘url компоненту’></Link>. В коді це мало наступний вигляд:

```
<Link to='/login' classNameName="sign-in">
    <button classNameName="btn">Увійти</button>
</Link>
<p></p>
<Link to='/Registration' classNameName="sign-in">
    <button classNameName="btn">Реєстрація</button>
</Link>
```

Рис 26 Перехід між компонентами

2.7 Взаємодія серверної та клієнтської частини

Після реалізації всього вищеперерахованого, залишалося зробити взаємодію серверної та клієнтської частини застосунку. Для початку була реалізована взаємодія домашньої основної сторінки з серверною частиною. Перша вимога була додання аутентифікації користувача, тобто реєстрація та вхід. Для реєстрації та авторизації був використаний метод Post через інтерфейс fetch. Для реєстрації були використані хуки(hooks) useState для збереження даних, які користувач передає через форму, а також axios.get для пошуку міста за назвою.

Код взаємодії сторінки реєстрації з серверною частиною застосунку виглядає наступним чином:

```
export const RegistrationScreen = () => {
  const [fullName, setFullname] = useState('');
  const [phoneNumber, setPhoneNumber ]= useState('');
  const [email, setEmail] = useState('');
  const [login, setLogin] = useState('');
  const [password, setPassword] = useState('');
  const [citys, setCity] = useState('');
  const [cities, setCities] = useState('');

  const cityId = cities

  var city = { id : cityId}

  const citySubmit = async (e) => {
    e.preventDefault();
    const city_url = "/cityName/" + citys
    axios.get(city_url).then((res) => res.data).then((data) => {
      const cityId = data.cityId;
      setCities(cityId)
    });
    let regobj ={fullName,phoneNumber,email,password,login,city};
    const json = JSON.stringify(regobj)


    fetch(REGISTER_URL, {method: "POST",headers: { 'content-type': 'application/json' } , body: JSON.stringify(regobj)}).then((res) => {console.log(res)})

    console.log(regobj)
  }
```

Рис 27 взаємодія реєстрації з серверною частиною застосунку

Ось таким чином буде виглядати сторінка реєстрації в браузері:

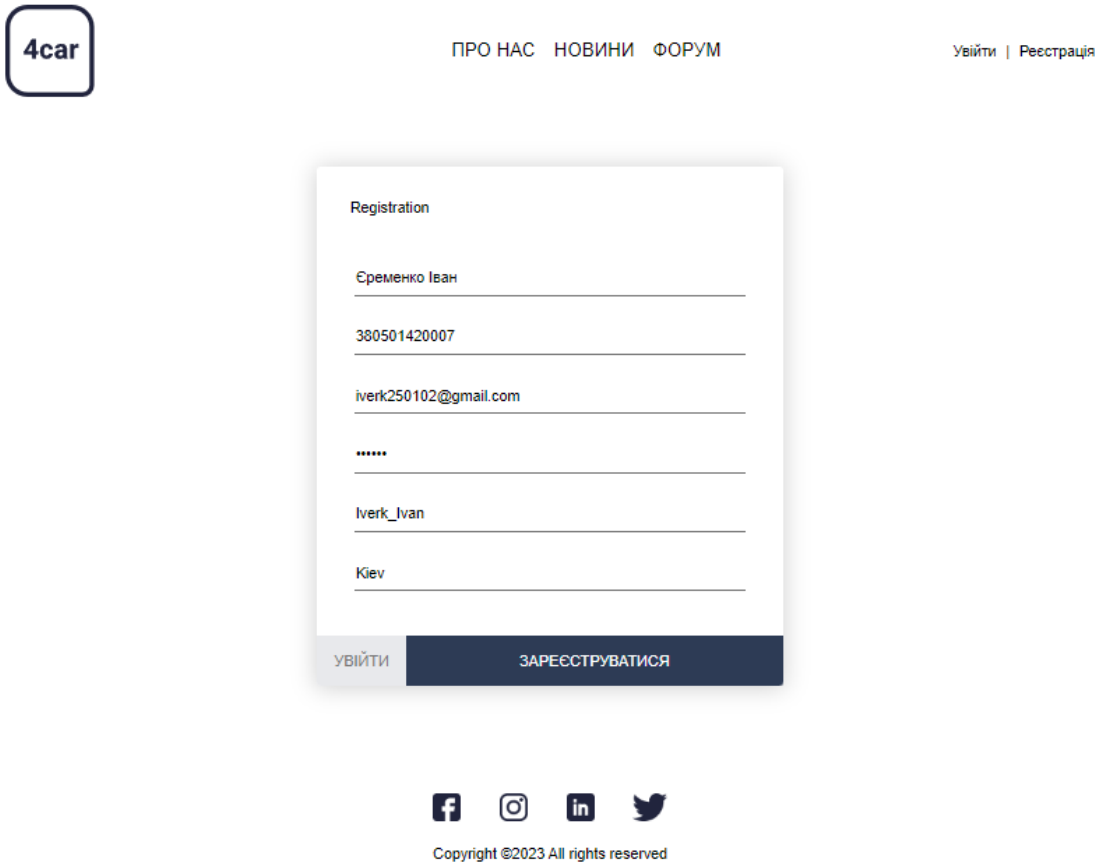

Рис 28 взаємодія реєстрації з серверною частиною застосунку

І при натисканні клавіші “Зареєструватися” буде зроблена запис до бази даних застосунку:

10	158	iverk250102@gmail.com	Єременко Іван	Iverk_Ivan	250102	380501420007	3

Рис 29 зареєстрований аккаунт у базі даних

Після реалізації реєстрації, було реалізовано взаємодію сторінки входу з серверною частиною. Для входу був доданий такий хук(hooks), як useContext для зберігання унікального поля аккаунту, даний хук дає змогу зберігати глобально певні дані, якими в нашому випадку було поле accountId. Тобто основна логіка полягала в тому, що ми заповнюємо форму , а саме вводимо логін та пароль, після чого робимо метод Post в

тілі якого отримуємо дані про аккаунт і записуємо в useContext поле accountId. В коді це має наступний вигляд:

```
export const AuthorizationScreen = () => {
  const [login, setLogin] = useState('');
  const [password, setPass] = useState('');
  const {accountId, setAccountId} = useContext(Context);



  const handleSubmit = (e) => {
    e.preventDefault();
    let regobj = {password,login};
    console.log(regobj)
    const LOGIN__SRC = LOGIN_SRC + login + "/password/" + password
    console.log(LOGIN__SRC)
    fetch(LOGIN__SRC, {
        method: "POST",
        headers: { 'content-type': 'application/json' },
        body: JSON.stringify(regobj)
    }).then((res) => {
      if(res.status === 200){
          return res.json()
          .then((data) => {
            setAccountId(data.accountId)
          })
      }
      else console.log(res.error)
    })
  }
```

Рис 30 взаємодія входу з серверною частиною

Сторінка входу має наступний вигляд у браузері:

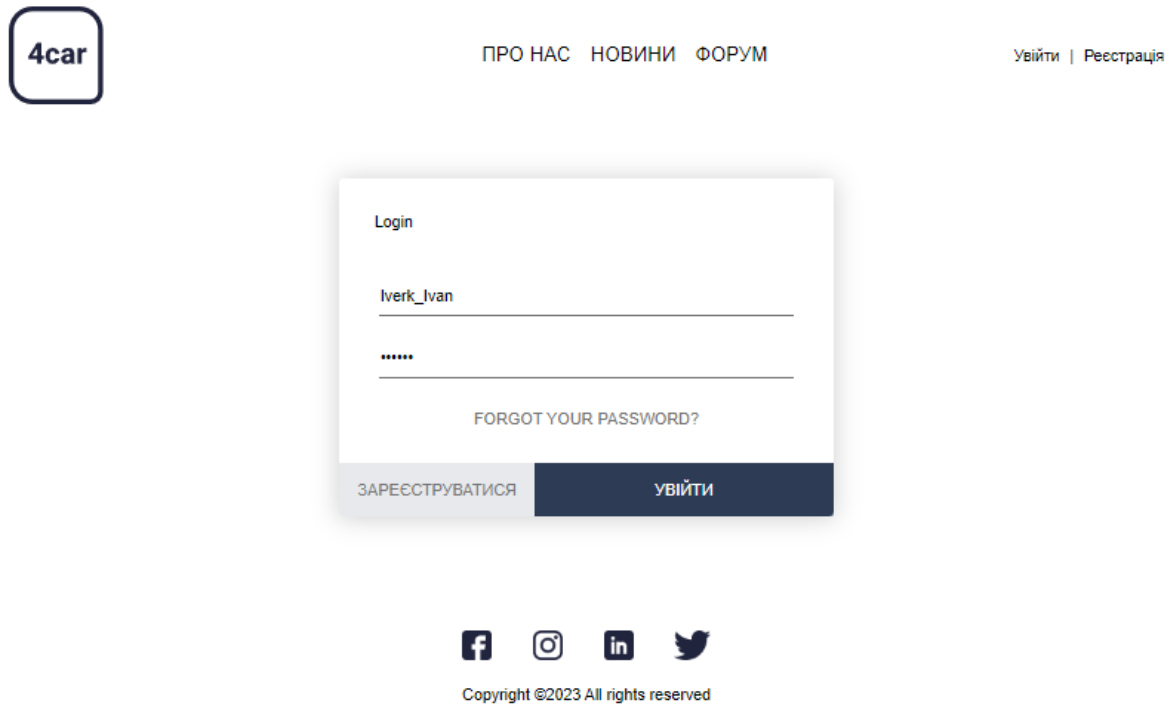

Рис 31 Сторінка входу у браузері

Після натискання кнопки “увійти” було зроблено перенаправлення на головну сторінку, але при цьому ми мали відкривати користувачу нові можливості перегляду аккаунту і тд. Оскільки користувач вже увійшов до системи, навігаційна панель мала змінюватися і замість кнопок входу та реєстрація мала з'явитися навігація до особистого аккаунту. Для цього за топомогою тернарних операторів додалась наступна умова:

```
{accountId == null?
<div className="sign">

    <Link to='/login' classNameName="sign-in">
        <button classNameName="btn">Увійти</button>
    </Link>
    <p></p>
    <Link to='/Registration' classNameName="sign-in">
        <button classNameName="btn">Реєстрація</button>
    </Link>

</div>
:
<div className="account">
    <div >
        <h3 className="acount-text">
            Аккаунт:
        </h3>
    </div>

    <div>
        <Link to={"/account"}>
            <h3 className="acount-text">{userName}</h3>
        </Link>
```

Рис 32 Тернарний оператор для перевірки чи здійснений вхід до акаунту

Скрін коли користувач не увійшов до акаунту ми вже подивилися, ось такий вигляд матиме головна сторінка, якщо користувач увійшов до акаунту:

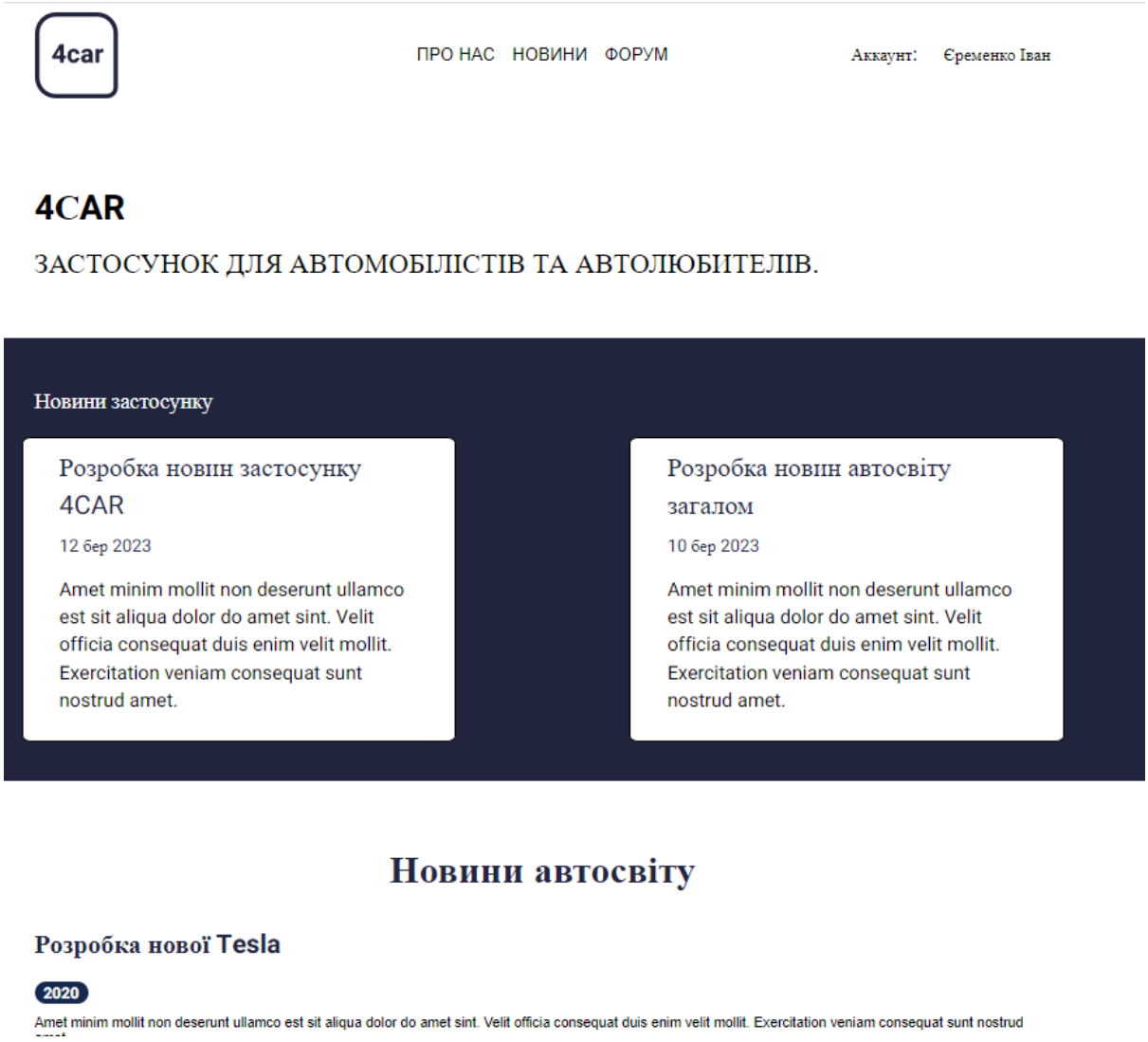

Рис 33 Головна сторінка після входу до акаунту

Тобто у нас з'явився лінк до нашого особистого аккаунту. На цьому зв'язок аутентифікації та серверної частини було закінчено.

Після того, як була зроблена аутентифікація, була розроблена сторінка особистого акаунту користувача. Використовуючи useContext, де ми попередньо зберегли accountId, ми можемо виводити всі дані про акаунт використовуючи axios.get. Сторінка особистого аккаунту мала включати в себе дані про акаунт, а також лінки на заповнення автомобіля по характеристикам, а також перегляд інформації про автомобіль. Сторінка персональної інформації має наступний вигляд:

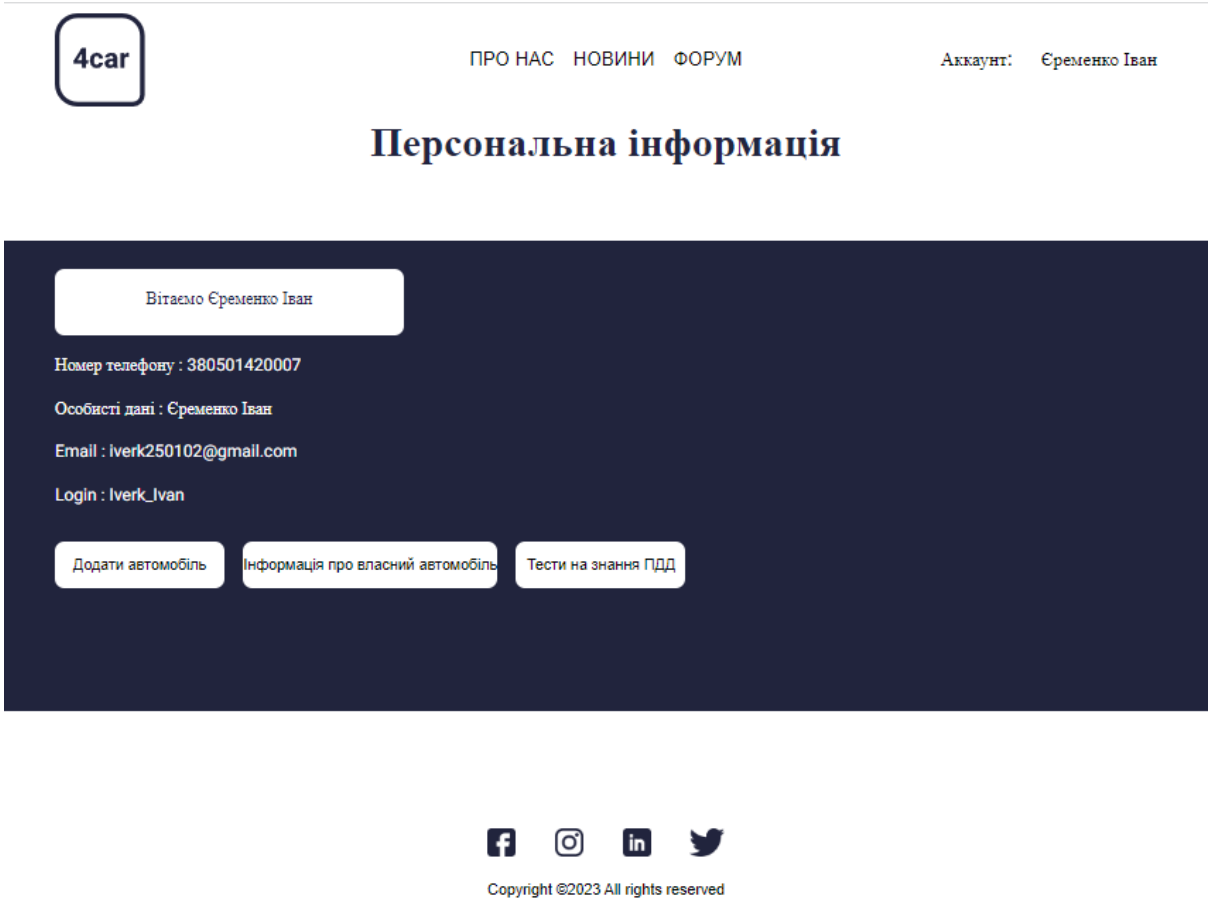

Рис 34 Сторінка акаунту(персональних даних)

При натисканні на кнопку “Додати автомобіль” відкривається форма для заповнення даних про авто, подібні до форми реєстрації, а при натисканні на кнопку “Інформація про автомобіль” виводиться вся інформація про ваш автомобіль подібна до сторінки персональної інформації, а при натисканні на кнопку “Тести на знання ПДР” відкривається сторінка, де користувач може пройти тестування. Тестування зроблено таким чином:

через axios.get ми отримуємо питання та правильний варіант відповіді, після цього ці дані передаються до Html коду, де все виводиться задаються відповіді, а також встановлюється флаг на правильну відповідь, після чого сумується кількість правильних відповідей і виводиться в результат тестування.

Після того як сторінка акаунту (особистої їнформації) була поєднана з серверною частиною застосунку. Був розроблений форум для спілкування користувачів. Для цього був використаний useState для зберігання інформації(повідомлень), useEffect для оновлення при додаванні нового повідомлення, а також useContext для того, щоб надіслати повідомлення.

Вивід повідомлень був зроблений за допомогою axios.get. А написання за допомогою fetch.post, який параметрами забирав дані з форми. Ось такий має вигляд код для форуму:

```
const[message, setMessage] = useState([]);
const {accountId, setAccountId}=useContext(Context)


const[forum, setForum] = useState([]);

useEffect(()=>{
    axios.get(MESSAGE_SRC).then(response => (
        response.data
    )).then(data =>{
        setForum(data)
    });
})

function padTo2Digits(num) {
    return num.toString().padStart(2, '0');
  }

function formatDate(date) {
    return [
      date.getFullYear(),
      padTo2Digits(date.getMonth() + 1),
      padTo2Digits(date.getDate()),
    ].join('-');
  }

  const date = formatDate(new Date())


  console.log(date);

const handleSubmit = (e) => {
    e.preventDefault();
    let regobj = {date , message , accountId};
    const body = JSON.stringify(regobj)
    console.log(body)
    fetch(POST_SRC, {
        method: "POST",
        headers: { 'content-type': 'application/json' },
        body: body
    }).then((res) => {
      return res.json().then((data) => {
          console.log(data)
      })
```

Рис 35 Код для коректної роботи форуму

На сторінці у браузері, це має наступний вигляд:

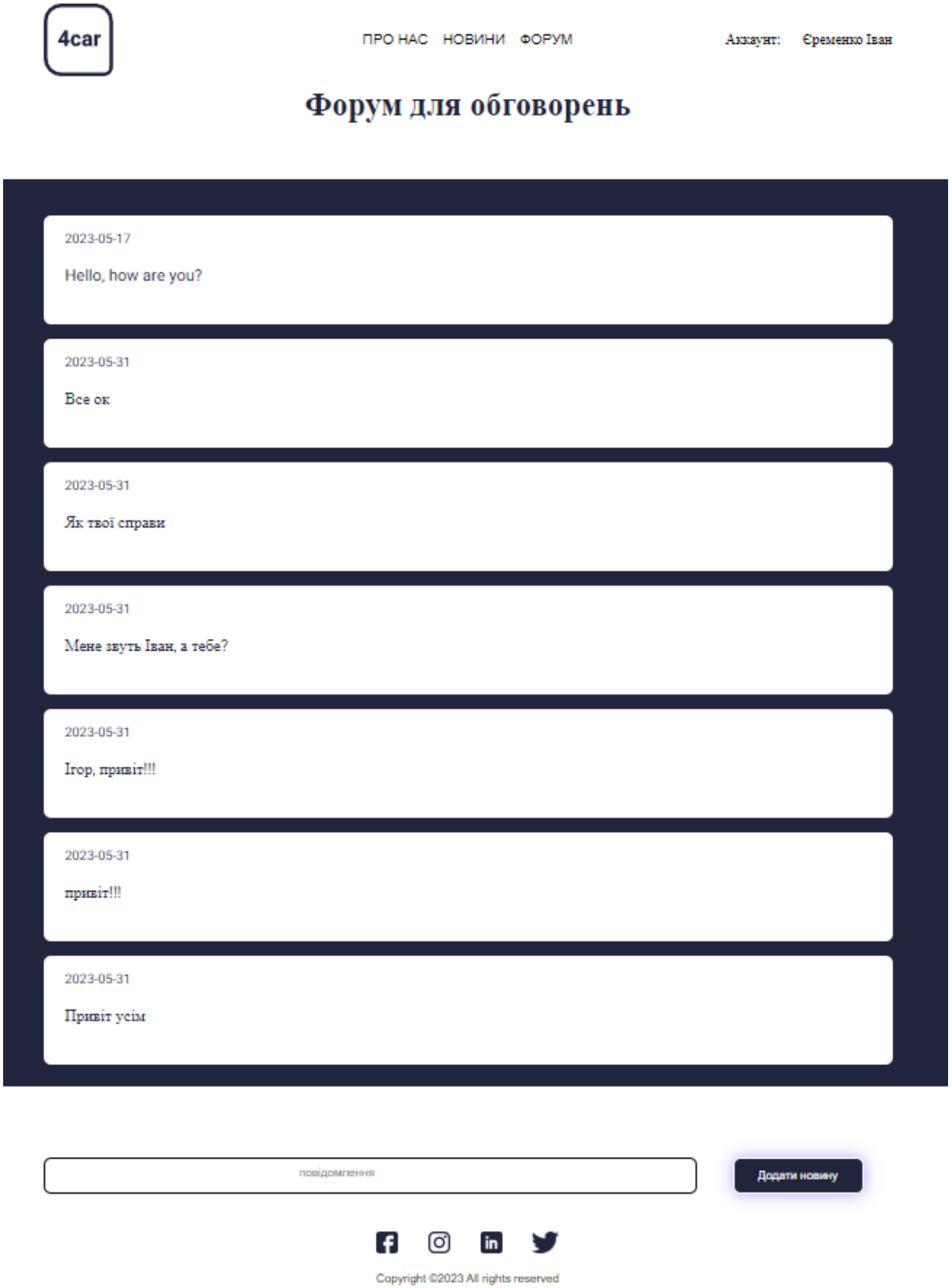

Рис 36 Сторінка форуму у браузері

В полі повідомлення користувач може ввести повідомлення, яке при натисканні кнопки “Додати новину” автоматично запишеться до бази даних і відтвориться на екрані застосунку.

На цьому основні функції застосунку для користувача були зроблені. Після цього додавалась частина можливостей для адміністратора. При авторизації, адміністратор має наступні функції:

- Додати місто/автосервіс/новину;
- Переглянути профіль;

Для додавання різних даних до застосунку використовувався матод fetch.post, в тіло якого передавалися всі необхідні дані і зберігалися в базі даних. Перегляд профілю реалізований так само як і для користувача. На цьому було завершена розробка застосунку.

ВИСНОВКИ

Результатом моєї дипломної роботи стало розроблення застосунку для автомобілістів. Були встановлені задачі, пояснена актуальність даної теми, проведений аналіз конкурентів, також була зроблена аналітика перспектив проекту у майбутньому. Після цього було описано функціональність системи та її покрокову роботу. Також було проведено проектування макету та основні налаштування. Після цього було розроблено клієнтську частину застосунку, а саме зверстано макет за допомогою Html, CSS коду, налаштовано роутинг між сторінками застосунку а також була зроблена взаємодія клієнтської та серверної частини застосунку.

Клієнтська частини застосунку була реалізована мовою програмування JavaScript, за допомогою фреймворку ReactJS. Також було покроково пояснено взаємодію клієнтської та серверної чатин застосунку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Основні застосунки для автомобілістів: веб-сайт: https://automarket.co.ua/blog/interesting/247566/top_prilozhenij_dlya_voditelya.html (дата звернення 12.03.23)
2. Документація React LS: https://legacy.reactjs.org/docs/getting-started.html (дата звернення 22.03.23)
3. Axios введення: https://axios-http.com/ru/docs/intro(дата звернення 02.04.23)
4. Fetch використання :https://www.atlassian.com/ru/git/tutorials/syncing/git-fetch(дата звернення 15.04.23)
5. Шрифти для проектування та стилізування застосунку: https://fonts.google.com/ (дата звернення 20.04.23)