Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету(відділення))
Кафедра інформаційних систем і комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(рівень вищої освіти)

на тему: «Розроблення сервісу "Car sharing" засобами фреймворку svelte»

Виконав студент 2 курсу, групи ІСТС-21
спеціальності: 126
„Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)
Бурко Денис Олегович
(прізвище, ініціали)
Керівник: ст.викл. Бекас Б.О.,
доц. Борецька І.Б.
(прізвище, ініціали)
Рецензент: Думанський О. Я.
(прізвище, ініціали)

Львів-2023

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 " Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри ІСКМ
Сторожук О.Л.
"21" 11 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бурко Денис Олегович
(прізвище, ім'я, по батькові)

1. Тема роботи *«Розроблення сервісу "Car sharing" засобами фреймворку svelte»*
керівники роботи. *Бекас Б.О., Борецька І.Б., к.т.н., доц.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"21"11 2022 року № С-521*
2. Термін подання студентом роботи *10.06.2023р.*
3. Вихідні дані до роботи: *Розробити програмне забезпечення для взяття автомобіля в оренду. Для розробки використати фреймворк Svelte.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1) Стан проблемної області;
2) Інформаційне та математичне забезпечення;
3) Програмне та технічне забезпечення;
4)Висновки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді.

6. Консультанти розділів проекту (роботи)

_____

7. Дата видачі завдання 23 листопада 2022р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	23.11-20.12	Викон.
2.	Постановка задачі і її формалізація	20.12-13.01	Викон.
3.	Виконання вхідного етапу технології	13.01-14.04	Викон.
4.	Реалізація головних класів проекту	14.04-20.05	Викон.
5.	Виконання етапу відлагодження проєкту	20.05.-25.05.	Викон.
6.	Виконання етапу впровадження та випуску бета-версії.	25.05.-02.06.	Викон.
7.	Оформлення записки до дипломного проекту.	02.06.-10.06.	Викон.

Студент _____ Бурко Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи

_____ ст. викл. Бекас Б.О.,

_____ доц. Борецька І.Б.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить 42 сторінки пояснювальної записки, 28 рисунків, 1 додаток та 15 використаних джерел.

Дипломна робота присвячена розробці програмного забезпечення для «кар шерінгу». Для створення застосунку використати фреймворк Svelte. Також потрібно реалізувати сторінку авторизації та реєстрації. Передбачити сторінку автомобілів яких ви можете взяти в оренду та сторінку кліянта, який може бачити історію та статус прокату. Розробити зручний та простий інтерфейс, який дозволить скористатись автомобілем у кілька кліків.

Ключові слова: Svelte, фреймворк, кар шерінг, програмне забезпечення.

ABSTRACT

The thesis contains 42 pages of explanatory notes, 28 figures, 1 appendix and 15 references.

The thesis is devoted to the development of software for "car sharing". To create an application, use the Svelte framework. You also need to implement the authorization and registration page. Provide a page for cars that you can rent and a page for the client who can see the history and status of the rental. Develop a convenient and simple interface that will allow you to use the car in a few clicks.

Keywords: Svelte, framework, car sharing, software.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити програмне забезпечення для кар шерінгу. Для створення застосунку використати фреймворк Svelte. Розробити сторінку авторизації та реєстрації. Також передбачити сторінку автомобілів яких ви можете взяти в оренду та сторінку кліянта, який може бачити історію та статус прокату. Розробити зручний та простий інтерфейс, який дозволить скористатись автомобілем у кілька кліків.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AST - абстрактне синтаксичне дерево (Abstract Syntax Tree).

DOM - об’єктна модель документа (Document Object Model).

API - програмний інтерфейс додатка (Application Programming Interface).

Svelte - JavaScript-фреймворк для створення вебдодатків.

SPA - односторінковий вебдодаток (Single Page Application).

ВСТУП

Каршерінг - це вид оренди автомобілів. Що відрізняє його від традиційної оренди автомобілів (Hertz, Enterprise тощо), так це те, що він розроблений таким чином, щоб бути зручним для людей, які хочуть орендувати автомобілі на короткі періоди часу (кілька годин) і платити лише за їх використання (рахунок виставляється залежно від тривалості вашого автомобіля та пройденої відстані).

Ще одна відмінність від традиційної оренди автомобілів, яка робить каршерінг більш практичним для людей, які не володіють автомобілем, полягає в тому, що він дозволяє отримати доступ до автомобіля в будь-який час, а не тільки в робочий час. А оскільки автомобілі розкидані по місту на зарезервованій парковці, швидше за все, є одна така парковка недалеко від місця, де ви живете, що дозволяє легко дійти до неї.

Об’єктом дослідження використання Svelte для створення сервісу для оренди автомобіля.

Метою роботи створення активного сервісу для легкого і швидкого способу пересування по місту чи сільській місцевості.

Предметом дослідження є використання фреймворку Svelte у застосунках сфери прокату авто.

Практичне значення пришвидшити процес взяття в оренду автомобіля.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області

Хоча Каршерінг стає популярним серед багатьох людей, які люблять випробовувати нові автомобілі і не хочуть ризикувати їх покупкою. Для новачків у спільному використанні автомобілів це відносно нова тенденція способу життя, яка дозволяє сісти за кермо без усіх клопотів із володінням автомобілем.

Що таке каршерінг?

Каршерінг швидко зростає в популярності, і хоча ви, можливо, чули цей термін раніше, ви, можливо, не знаєте точно, що він означає. У багатьох людей складається враження, що каршерінг призначений лише для тих, хто ще не володіє автомобілем. Це не зовсім так.

Якщо вам подобається ідея володіння автомобілем або навіть якщо ви любите свій автомобіль, чому б не спробувати щось нове? Каршерінг - це інноваційний і зростаючий спосіб пересування містом. Це відмінний варіант для звичайних громадян, але він також може бути економічно вигідним і екологічно чистим варіантом для компаній.

Каршерінг проти прокату автомобілів: в чому різниця?

Незалежно від того, робите ви це для навколишнього середовища або добираєтеся з точки А в точку Б, каршерінг є чудовою альтернативою традиційній оренді. Послуга каршерінга - це форма оренди автомобіля, яка дозволяє людині орендувати транспортний засіб на погодинній основі.

На відміну від традиційних компаній з прокату автомобілів, які виставляють рахунки користувачам на основі тривалості часу, протягом якого вони мають машину, або відстані, яку вони проїжджають, каршерінгам зазвичай

виставляються рахунки погодинно на основі використання. Гнучкість, яку пропонує служба спільного використання автомобілів, в якій учасники можуть орендувати автомобілі в будь-який час, робить цю опцію більш привабливою для людей, у яких немає автомобіля.

Як працює каршерінг?

Процес каршерінга відносно простий. В якості першого кроку вам потрібно з'ясувати, чи є у вашому місті послуги каршерінга і, якщо так, то з якими операторами вони пов'язані. Швидкий пошук в Інтернеті може бути всім, що потрібно.

Щоб знайти варіанти спільного використання автомобілів у вашому регіоні, виконайте пошук в Інтернеті за запитом "каршерінг" плюс назва вашого міста чи селища. Іншим ресурсом для пошуку програм каршерінгу є список операторів каршерінгу у Вікіпедії за країнами.

Найкращі Програми Для Оренди Автомобілів

При приєднанні до послуги каршерінга вас можуть попросити підписати договір, в якому викладені правила членства. Підписатися на каршерінг легко. Більшість служб мають онлайн-форми реєстрації, щоб зареєструватися та вибрати тарифний план, який найкраще підходить для вас.

Після того, як вас приймуть до програми, вам потрібно буде знати, як забронювати автомобілі та де їх знайти.

1.2. Переваги та недоліки каршерінга

Переваги каршерінга

Незалежно від того, чи намагаєтеся ви уникнути високих тарифів на паркування, хочете заощадити гроші або не хочете мати справу з клопотами, пов'язаними з наявністю автомобіля, є багато причин, чому каршерінг - це майбутнє. Хоча програми каршерінгу ще не повсюдно поширені в кожному

місті, вони зростають у популярності та доступності. Отже, давайте розглянемо переваги приєднання до послуги каршерінга.

1. Економічно вигідний

Володіти автомобілем дорого. Від газу до паркування, до витрат на технічне обслуговування та страхування, володіння автомобілем коштує тисячі доларів на рік. Додайте це до багатьох інших супутніх витрат, і легко зрозуміти, чому каршерінг вигідний.

Каршерінг дає можливість орендувати багато видів транспортних засобів погодинно або вдень і платити тільки за час, який ви використовуєте. Рівень членства низький, а також немає витрат на технічне обслуговування або утримання, що дозволяє легко сісти за кермо кількома способами.

2. Екологічно чистий

Більшість людей прагнуть зменшити свій вуглецевий слід. Каршерінг може допомогти зменшити кількість транспортних засобів на дорозі, кількість пройдених кілометрів та загальне володіння автомобілем.

Чим менше транспортних засобів на дорозі, тим менше заторів і викидів вуглекислого газу. Це також зменшує потребу в додатковій парковці. Це дозволяє створити більше зелених насаджень.

3. Зниження стресу

Для людей, які живуть у міських районах, володіння автомобілем може бути податком, а обмежений простір для зберігання може бути незручним.

Основною перевагою каршерінга є його зручність; Учасники можуть швидко захопити транспортний засіб, щоб задовольнити свої потреби в будь-який

момент. Завдяки великому вибору транспортних засобів, каршерінг ідеально підходить для швидких поїздок по місту або поїздок на вихідні. Ви можете орендувати позашляховик або автомобіль більшого розміру для важливих доручень, тоді як ви можете сісти на компактний автомобіль для швидкої поїздки до ресторану.

Деякі люди мають лише один автомобіль у своєму домогосподарстві; У таких випадках володіння автомобілем може відчувати себе тягарем. Каршерінг дає можливість взяти в користування транспортний засіб на період від декількох годин до декількох днів, не турбуючись про технічне обслуговування або зберігання.

4. Розширений доступ

Каршерінг робить його доступнішим і простішим для пересування. Це зменшує затори та забруднення в місті. Це чудова можливість для людей, які думають про придбання автомобіля, але не впевнені, як вони можуть собі це дозволити. Ті, хто має поганий кредитний рейтинг, також можуть скористатися послугами спільного використання автомобілів.

Програми каршерінга збільшують кількість людей, які мають доступ до автомобілів, оскільки позбавляють від необхідності купувати автомобіль.

Недоліки каршерінга

В даний час все більше людей вибирають послуги каршерінга. Однак, хоча ідея спільного використання автомобілів чудова, є кілька недоліків, про які ви повинні знати.

1. Підвищення цін у години пік

Однією з основних проблем спільного використання автомобілів є його різке зростання цін у години пік. Оскільки люди, як правило, використовують свої автомобілі в однаковий час - наприклад, годину пік, шкільні пробіжки та свята - витрати вищі, коли попит вищий.

2. Недоступність транспортних засобів

Знайти автомобіль, коли він вам потрібен, може бути важко, тому що багато людей хочуть використовувати автомобілі одночасно або тому, що в певному регіоні мало компаній з каршерінгу.

3. Ліміти пробігу

Ще одна причина, чому каршерінг поганий, полягає в тому, що він може обмежити частину вашої особистої свободи. Програми каршерінга накладають обмеження на пробіг та інші обмеження, які можуть зробити їх дорогими і непрактичними для використання в тривалих поїздках.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Svelte

Svelte - це інструмент для створення веб-додатків, подібний до JavaScript фреймворків, таких як React та Vue, з серцем для полегшення створення інтерактивних інтерфейсів користувача.

Але є одна ключова відмінність: Svelte перетворює вашу програму в ідеальну програму JavaScript на етапі збирання/компіляції, а не інтерпретує код програми на етапі запуску. Це означає, що ви не платите за продуктивність, споживану фреймворком, і немає додаткових втрат при першому завантаженні програми.

Ви можете використовувати Svelte для створення всієї вашої програми, або ви можете поступово змішувати її з існуючим кодом. Ви також можете поставляти компоненти як автономні пакети в будь-якому місці і без додаткових накладних витрат традиційних фреймворків.

Svelte здатна компілювати код у невеликий, незалежний від фреймворку простий js код, що робить програму швидкою як для запуску, так і для запуску.

Краща продуктивність

Багато студентів, які навчаються, реагують або vue, можливо, чули коментарі на кшталт "віртуальний будинок - це швидко", тому, коли вони бачать це, вони дивуються, чому svelte швидше без віртуального дому?

Це насправді непорозуміння, react і vue та інші фреймворки для досягнення основної мети virtual dom - це не продуктивність, а охоплення базових операцій dom, щоб користувачі могли створювати наші додатки за допомогою декларативного, керованого державою способу UI для поліпшення ремонтопридатності коду.

Крім того, що react або vue мають на увазі під хорошою продуктивністю віртуального дому, це те, що фреймворк все ще може забезпечити хороші гарантії продуктивності без будь-яких спеціальних оптимізацій сторінки. Наприклад, в наступному сценарії ми повторно виводимо список кожен раз, коли отримуємо дані з сервера, і якщо ми не будемо робити спеціальні оптимізації за допомогою звичайних операцій dom, ми будемо кожен раз заново відтворювати всі елементи списку, і споживання продуктивності відносно високе. Фреймворки, такі як react, позначають елементи списку ключем і лише повторно відображають елементи списку, які змінилися, таким чином продуктивність покращується.

Думаючи про вищеописаний сценарій, якщо ми також відзначаємо елементи списку при маніпулюванні реальним dom, і тільки заново вимальовуємо елементи списку, які змінилися, усуваючи необхідність у virtual dom diff, то продуктивність навіть вище, ніж віртуальний dom.

Потім Svelte реалізує цю оптимізацію, зіставляючи дані з реальним DOM, обчислюючи та зберігаючи їх під час компіляції через AST, а також оновлюючи DOM безпосередньо при зміні даних, що відбувається дуже швидко при ініціалізації та оновленні, оскільки він не покладається на віртуальний DOM.

Реальна реактивність

Svelte додає чуйність до самого JavaScript без необхідності створення складної бібліотеки управління станом. Адаптивна реалізація svelte пояснюється далі в розділі інтерпретації вихідного коду.

Тенденції розвитку

Svelte is Rich Harris (автор роллапу), а svelte розпочав відкритий сорсинг у 2016 році та почав привертати більш широку увагу у 2019 році. Svelte на Github зараз має 49.9 тисяч зірок. Щотижневі завантаження svelte на Npm становлять близько 15 Вт.

Хоча він все ще значно відстає від React, Vue та Angular з точки зору кількості зірок та завантажень, це зрозуміло, враховуючи його відносно пізній дебют. Більше того, згідно з дослідженням фреймворка, його задоволеність та інтерес користувачів є найвищими за останні два роки, а його використання та популярність швидко зростають.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розроблення інтерфейсу програми

Щоб розгорнути Svelte в робочій області, вам спочатку потрібно встановити необхідні залежності та налаштувати проект. Ось кроки, які потрібно виконати:

Крок 1: Встановлення Node.js. Перш ніж почати, переконайтеся, що ви встановили Node.js на своєму комп'ютері. Ви можете завантажити його з https://nodejs.org і встановити згідно з інструкціями для вашої операційної системи.

Крок 2: Створення нового проекту Svelte. Відкрийте командний рядок або термінал та перейдіть до робочого каталогу, де ви хочете створити свій проект. Потім введіть наступну команду для створення нового проекту Svelte:

```
npx degit sveltejs/template svelte-app
```

Ця команда створить нову папку з назвою "svelte-app" із початковим шаблоном Svelte.

Крок 3: Встановлення залежностей. Перейдіть до створеної папки "svelte-app" за допомогою команди:

```
cd svelte-app
```

Потім виконайте наступну команду, щоб встановити залежності проекту:

```
npm install
```

Ця команда завантажить всі необхідні пакети, включаючи Svelte.

Крок 4: Запуск розробки. Після встановлення залежностей ви можете запустити розробку вашого проекту Svelte, виконавши наступну команду:

```
npm run dev
```

Ця команда запустить локальний сервер розробки і відкриє ваш проект у браузері. Ви зможете бачити зміни в реальному часі, коли ви редагуєте свій код Svelte.

Тепер перейдемо до самої розробки і почнемо з головного файлу App.svelte. В ньому використовуються різні компоненти, налаштовуються маршрути та задаються стилі для основного контейнера програми. Почнемо з компонентів які будуть використовуватись у проекті:

- Router – використовується для налаштування маршрутизації;
- Tailwindcss, – містить налаштування та стилі;
- "components/Footer.svelte" – підвал сайту;
- "components/Header.svelte" – шапка сайту;
- "routes/Home.svelte" – головна сторнка;
- "routes/Login.svelte" – сторінка входу;
- "routes/Register.svelte" – сторінка реєстрації;
- "routes/mod&Admin/AdminPanel.svelte" –панель адміністратора;
- "routes/CarList.svelte" – сторінка зі списком автомобілів;
- "routes/CarInfo.svelte" – сторінка з інформацією про конкретний автомобіль;
- "routes/NotFound.svelte" – це компонент, який відображається, якщо маршрут не знайдено;
- "routes/mod&Admin/AllReservations.svelte" – сторінка з усіма резерваціями для адміністраторів;
- "routes/MyReservations.svelte" – сторінка з резерваціями користувача;
- "routes/ReservInfo.svelte" – сторінка з інформацією про конкретну резервацію;

За допомогою Router буде присвоюватись шлях кожному компоненту який ми оголосили. Наприклад, шлях "/" відповідає компоненту Home, шлях "/login" - Login, тощо.

У секції <style> задається стиль для main, який визначає фоновий колір, ширину, висоту та вирівнювання вмісту.

Дана логіка [див. рис. Файл App.svelte] встановлює маршрутизацію та обгортку сторінки, включаючи хедер, вміст та футер.

```
<script>
    import Router from "svelte-spa-router";
    import Tailwindcss from "./Tailwindcss.svelte";
    import Footer from "./components/Footer.svelte";
    import Header from "./components/Header.svelte";
    import Home from "./routes/Home.svelte";
    import Login from "./routes/Login.svelte";
    import Register from "./routes/Register.svelte";
    import AdminPanel from "./routes/mod&Admin/AdminPanel.svelte";
    import CarList from "./routes/CarList.svelte";
    import CarInfo from "./routes/CarInfo.svelte";
    import NotFound from "./routes/NotFound.svelte";
    import AllReservations from "./routes/mod&Admin/AllReservations.svelte";
    import MyReservations from "./routes/MyReservations.svelte";
    import ReservInfo from "./routes/ReservInfo.svelte";
    import TimeSimulation from "./routes/mod&Admin/TimeSimulation.svelte";
    import TermsConditions from "./routes/TermsConditions.svelte";
</script>
<Tailwindcss />
<main id="main">
    <Header />
    <Router
        routes={{
            "/": Home,
            "/login": Login,
            "/register": Register,
            "/adminPanel": AdminPanel,
            "/carList": CarList,
            "/carInfo/:id": CarInfo,
            "/allReservations": AllReservations,
            "/myReservations": MyReservations,
            "/reservInfo/:id": ReservInfo,
            "/timeSimulation": TimeSimulation,
            "/termsConditions": TermsConditions,
            "*": NotFound,
        }}
    />
    <Footer />
</main>
```

Рис. 3.1. Файл App.svelte

Тепер потрібно розробити кожен з компонентів які ми оголосили, почнемо з реєстрації [див. рис. Файл Register.svelte].

Імпортується компонент Loader з файлу "./components/Loader.svelte", який буде відображатися під час обробки даних.

Визначаються змінні: canRegister використовується для перевірки можливості реєстрації, а login, password, email - для збереження даних, введених користувачем.

Функція onRegister, яка обробляє подію натискання на кнопку реєстрації. Вона перевіряє, чи заповнені обов'язкові поля (логін, пароль, електронна пошта), створює об'єкт FormData зі значеннями полів і відправляє його на сервер за допомогою fetch. Після успішного виконання запиту результат обробляється: якщо реєстрація відбулася успішно (data.opperation == true), поля форми очищаються, в іншому випадку виводиться повідомлення про помилку.

Якщо користувач не залогінений (status.logged == false), canRegister встановлюється на true, що дозволяє реєстрацію. В іншому випадку користувач перенаправляється на головну сторінку за допомогою location.href.

```
<script>
    import Loader from "./components/Loader.svelte";
    let canRegister = false;
    let login, password, email;
    function onRegister(e) {
        e.preventDefault();
        if (login && password && email) {
            let form = new FormData();
            form.append("login", login);
            form.append("password", password);
            form.append("email", email);

            fetch("./backend/Register.php", {
                method: "POST",
                body: form,
                mode: "no-cors",
            })
                .then((res) => res.json())
                .then((data) => {
                    if (data.locReload) {
                        location.href = "./#";
                    }
                    APP.displayMessage(
                        !data.opperation,
                        data.info,
                        document.getElementById("header")
                    );
                    if (data.opperation == true) {
                        [login, password, email] = new Array(3).fill("");
                    }
                });
        } else {
            APP.displayMessage(
                true, "Wrong data", document.getElementById("header")
            );
        }
    }
    APP.checkStatus().then((status) => {
        if (status.logged == false) {
            canRegister = true;
        } else {
            location.href = "./#";
        }
    });
```

Рис. 3.2. Файл Register.svelte

Компонент "Login.svelte" містить логіку для авторизації [див. рис. Файл Login.svelte].

```
import { displayMessage, checkStatus } from "../js/app.js";
import { store_STATUS } from "../stores/storeone";
import Loader from "./components/Loader.svelte";
let login, password;
let canLogin = false;
function onLogin(e) {
    e.preventDefault();
    if (login && password) {
        let form = new FormData();
        form.append("login", login);
        form.append("password", password);

        fetch("./backend/Login.php", {
            method: "POST",
            body: form,
            mode: "no-cors",
        })
            .then((res) => res.json())
            .then((data) => {
                if (data.locReload || data.opperation) {
                    location.href = "./#";
                    let stat = checkStatus();
                    store_STATUS.update((v) => stat);
                } else {
                    displayMessage(
                        !data.opperation,
                        data.info,
                        document.getElementById("header")
                    );
                }
            });
    } else {
        displayMessage(
            true, "Wrong data", document.getElementById("header")
        );
    }
}
checkStatus().then((status) => {
    if (status.logged == false) {
        canLogin = true;
    } else {
        location.href = "./#";
    }
});
```

Рис. 3.3. Файл Login.svelte

Логіка з даного рисунка наступна:

Функції displayMessage і checkStatus імпортуються з файлу "../js/app.js" для відображення повідомлень та перевірки статусу авторизації.

Змінна store_STATUS імпортується з файлу "../stores/storeone" для отримання статусу користувача.

Компонент Loader імпортується з файлу "./components/Loader.svelte" і використовується для відображення процесу обробки даних або взаємодії з сервером.

Змінні login і password використовуються для збереження даних, введених користувачем, під час авторизації.

Функція onLogin визначається для обробки події натискання на кнопку авторизації. Вона перевіряє, чи заповнені обов'язкові поля (логін, пароль), створює об'єкт FormData і відправляє його на сервер за допомогою fetch.

При успішному запиті результат обробляється, і перенаправє на головну сторінку, а також оновлюється статус авторизації в збереженому стані store_STATUS. У випадку невдалої авторизації виводиться повідомлення про помилку.

Файл "CarList.svelte" відповідає за відображення списку автомобілів.

```
import { carListTypes, mySort } from "../js/app";
import ChevronCircleUp from "svelte-icons/fa/FaChevronCircleUp.svelte";
import ChevronCircleDown from "svelte-icons/fa/FaChevronCircleDown.svelte";
import CarListRow from "./components/carListRow.svelte";
import Loader from "./components/Loader.svelte";

let canRender = false;

let carList = getCars();
let tempList = [];
let archList = [];

let sortedBy;
let sortDirect = "UP";

let models = [];
let brands = [];
let years = [];
let prices = [];

let filtrModel = "all";
let filtrBrand = "all";
let filtrYear = "all";
let filtrPrice = "all";
```

Рис. 3.4. Файл CarList.svelte частина 1

Функції carListTypes і mySort з файлу "../js/app.js" для функціональності, пов'язаної зі списком автомобілів та сортуванням.

Компоненти ChevronCircleUp та ChevronCircleDown з пакету "svelte-icons/fa". Ці компоненти використовуються для відображення значків стрілок, які показують напрямок сортування.

CarListRow з файлу "./components/carListRow.svelte". Це компонент для відображення окремого рядка списку автомобілів.

Змінна canRender, яка початково має значення false. Вона використовується для управління відображенням списку автомобілів.

Функція carList отримує дані про автомобілі за допомогою асинхронного запиту до сервера. tempList і archList використовуються для зберігання тимчасових і архівних списків автомобілів відповідно.

Зміна sortedBy вказує, за яким полем відсортовано список автомобілів, а sortDirect вказує напрям сортування ("UP" або "DOWN").

Змінні models, brands, years і prices, які використовуються для фільтрації списку автомобілів за моделлю, маркою, роком випуску і ціною.

Визначаються змінні filtrModel, filtrBrand, filtrYear і filtrPrice, які зберігають вибрані параметри фільтрації.

```
async function getCars() {
    const URL = "./backend/CarList.php";
    let res = await fetch(URL, { method: "GET" });
    res = await res.json();
    tempList = res.cars;
    archList = res.cars;
    models = res.cars
        .map((e) => e.model)
        .filter((e, i, a) => a.indexOf(e) == i);
    fixFilters(res.cars);
    canRender = true;
    return res;
}
function fixFilters(tab) {
    brands = tab.map((e) => e.brand).filter((e, i, a) => a.indexOf(e) == i);
    years = tab.map((e) => e.year).filter((e, i, a) => a.indexOf(e) == i);
    prices = tab.map((e) => e.price).filter((e, i, a) => a.indexOf(e) == i);

    if (filtrBrand != "all") {
        if (brands.indexOf(filtrBrand) == -1) {
            filtrBrand = "all";
        }
    }
    if (filtrYear != "all") {
        if (years.indexOf(filtrYear) == -1) {
            filtrYear = "all";
        }
    }
    if (filtrPrice != "all") {
        if (prices.indexOf(filtrPrice) == -1) {
            filtrPrice = "all";
        }
    }
}
function carDetail(id) {
    location.href = `./#/carInfo/${id}`;
}
```

Рис. 3.5. Файл CarList.svelte частина 2

Асинхронна функція getCars(), яка виконує запит до сервера для отримання списку автомобілів. Результат запиту зберігається в tempList і archList, а також ініціалізуються змінні models, brands, years і prices. Змінна canRender встановлюється в true, що дозволяє відображати список автомобілів.

Функція fixFilters(tab) виконує попередню обробку фільтрів. Вона оновлює значення змінних brands, years і prices, а також перевіряє, чи вибрані фільтри є валідними і, у разі потреби, скидає їх до значень за замовчуванням.

Функція carDetail(id) перенаправляє користувача на сторінку з детальною інформацією про вибраний автомобіль.

```
function filtrCars(type, select) {
    sortedBy = undefined;
    sortDirect = "UP";
    if (type == "model") {
        filtrModel = select.value;
        filtrPrice = "all";
        filtrBrand = "all";
        filtrYear = "all";
    } else if (type == "brand") {
        filtrBrand = select.value;
    } else if (type == "year") {
        filtrYear = select.value;
    } else if (type == "price") {
        filtrPrice = select.value;
    }

    if (carList) {
        if (filtrModel == "all") {
            tempList = Array.from(archList);
        } else {
            tempList = archList.filter((e) => e.model == filtrModel);
            fixFilters(tempList);
        }
        if (filtrBrand != "all") {
            tempList = tempList.filter((e) => e.brand == filtrBrand);
            fixFilters(tempList);
        }
        if (filtrYear != "all") {
            tempList = tempList.filter((e) => e.year == filtrYear);
            fixFilters(tempList);
        }
        if (filtrPrice != "all") {
            tempList = tempList.filter((e) => e.price == filtrPrice);
            fixFilters(tempList);
        }

        carList = Promise.resolve({
            cars: tempList,
        });
    }
}
```

Рис. 3.6. Файл CarList.svelte частина 3

Функція filtrCars(type, select) виконує фільтрацію списку автомобілів відповідно до вибраних фільтрів. Вона оновлює змінні filtrModel, filtrBrand, filtrYear і filtrPrice відповідно до вибраних параметрів. Потім вона застосовує фільтри до tempList, оновлює значення tempList і встановлює carList в нове значення, яке включає оновлений список автомобілів.

```
function setSort(type) {
    if (sortedBy != type) {
        sortedBy = type;
        sortDirect = "UP";
    } else {
        sortDirect = sortDirect == "UP" ? "DOWN" : "UP";
    }
    if (carList) {
        carList = Promise.resolve({
            cars: mySort(tempList, sortDirect, sortedBy),
        });
    }
}
```

Рис. 3.7. Файл CarList.svelte частина 4

Функція setSort(type) встановлює параметри сортування списку автомобілів. Якщо type відповідає вже вибраному полю сортування, то змінюється напрям сортування (sortDirect). У залежності від sortedBy і sortDirect оновлюється carList.

Перейдемо «CarInfo.svelte» який відповідає за інформацію по автомобіль.

```
export let params;
import { displayMessage } from "../js/app";
import Loader from "./components/Loader.svelte";
import MyIcon from "./components/MyIcon.svelte";

let today = new Date(Date.now());

let startDate, endDate, logged;

let resDays = 0;
let pricePerDay = 0;

let carInfo = getCarInfo(params.id);
```

Рис. 3.8. Файл CarInfo.svelte частина 1

Експортується змінна params, яка використовується для отримання параметрів URL.

Імпортується функція displayMessage з модуля ../js/app.js.

Імпортується компонент Loader з файлу ./components/Loader.svelte.

Імпортується компонент MyIcon з файлу ./components/MyIcon.svelte.

Ініціалізуються змінні today, startDate, endDate, logged, resDays та pricePerDay.

Викликається функція getCarInfo(params.id), яка отримує інформацію про автомобіль за допомогою запиту до сервера.

```
async function getCarInfo(id) {
    let form = new FormData();
    form.append("id", id);
    const URL = "./backend/CarInfo.php";
    let res = await fetch(URL, {
        method: "POST",
        body: form,
        mode: "no-cors",
    });
    res = await res.json();
    pricePerDay = parseInt(res.data.price);
    logged = res.logged;
    return res;
}

function setDate(type, input) {
    if (type == "start") {
        startDate = input.target.value;
    } else {
        endDate = input.target.value;
    }
    if (startDate && endDate) {
        let sTime = new Date(startDate).getTime();
        let eTime = new Date(endDate).getTime();
        let sub = eTime - sTime;
        resDays = Math.round(sub / 86400000);
    }
}
```

Рис. 3.9. Файл CarInfo.svelte частина 2

Функція getCarInfo(id) виконує запит до сервера для отримання інформації про автомобіль з використанням переданого ідентифікатора id.

Функція setDate(type, input) встановлює дати початку та кінця бронювання та обчислює кількість днів бронювання resDays.

```
async function reservCar(queueStatus = false) {
    if (queueStatus) {
        location.href = "./#/myReservations";
    } else {
        if (resDays > 0 && logged == true) {
            let form = new FormData();
            form.append("offerID", params.id);
            form.append("endPrice", resDays * pricePerDay);
            form.append("resStart", startDate);
            form.append("resTime", resDays);
            const URL = "./backend/MakeReserv.php";
            let res = await fetch(URL, {
                method: "POST",
                body: form,
                mode: "no-cors",
            });
            res = await res.json();
            if (res.status == true) {
                carInfo = getCarInfo(params.id);
            }
            displayMessage(
                !res.status,
                res.info,
                document.getElementById("header")
            );
        } else {
            displayMessage(
                true,
                logged == false
                    ? "You must be logged in"
                    : "Wrong reservation date",
                document.getElementById("header")
            );
        }
    }
}
```

Рис. 3.10. Файл CarInfo.svelte частина 3

Функція reservCar(queueStatus) виконує бронювання автомобіля за допомогою запиту до сервера. Умовна конструкція перевіряє, чи виконані умови для здійснення бронювання і чи користувач має ввійти в систему перед бронюванням.

Функція displayMessage використовується для відображення повідомлень про стан бронювання.

Тепер напишемо логіку для роботи з резервованими автомобілями для клієнта [див. рис. Файл MyReservations.svelte].

```
import { checkStatus, displayMessage } from "../js/app";
import Loader from "./components/Loader.svelte";
import MyReservationsTable from "./components/MyReservationsTable.svelte";

let canRender = false;
let reservations;

async function getMyReservations() {
    const URL = "./backend/MyReservations.php";
    let res = await fetch(URL, { method: "GET" });
    res = await res.json();
    return res;
}
async function changeVal(action, id, rentID = false) {
    let form = new FormData();
    form.append("action", action);
    form.append("id", id);
    if (action == "returnCar") {
        form.append("rentID", rentID);
    }
    fetch("./backend/MyReservations.php", {
        method: "POST",
        body: form,
        mode: "no-cors",
    })
        .then((res) => res.json())
        .then((data) => {
            reservations = getMyReservations();
            displayMessage(
                !data.action,
                data.info,
                document.getElementById("header")
            );
        });
}
checkStatus().then((status) => {
    if (status.logged == true) {
        canRender = true;
        reservations = getMyReservations();
    } else {
        location.href = "./#";
    }
});
```

Рис. 3.11. Файл MyReservations.svelte

Функція getMyReservations() виконує запит до сервера для отримання списку бронювань користувача.

Функція changeVal(action, id, rentID = false) виконує зміни у списку бронювань користувача на основі вказаної дії. Умовна конструкція перевіряє, чи користувач увійшов в систему. Якщо так, то відображається список бронювань користувача та можливість внесення змін. Якщо користувач не увійшов в систему, відбувається перенаправлення на головну сторінку.

Потрібно розробити компонент який відповідає за отримання інформації про конкретне бронювання і перевірку доступу до цієї інформації. Якщо доступ дозволений, відображається компонент, який показує деталі бронювання, або здійснюється перенаправлення на головну сторінку, якщо доступ заборонений [див. рис. Файл ReservInfo.svelte].

```
<script>
    export let params;
    import { calculateDate } from "../js/app";
    import Loader from "./components/Loader.svelte";

    let rentInfo = getRentInfo(params.id);
    let canRender = false;

    async function getRentInfo(id) {
        let form = new FormData();
        form.append("id", id);
        const URL = "./backend/reservInfo.php";
        let res = await fetch(URL, {
            method: "POST",
            body: form,
            mode: "no-cors",
        });
        res = await res.json();
        if (res.access == false) {
            location.href = "./#";
        } else {
            canRender = true;
        }
        return res;
    }
</script>
```

Рис. 3.12. Файл ReservInfo.svelte

Тепер перейдемо до логіки адміністратора, спочатку створимо адміністрування над користувачами, де зможемо роздавати права доступу до системи, або взагалі заблокувати користувача [див. рис. Файл AdminPanel.svelte].

```
<script>
    import { checkStatus } from "../../js/app";
    import AdminPanelRow from "../components/AdminPanelRow.svelte";
    import Loader from "../components/Loader.svelte";

    let canAdmin = false;
    let users;

    async function getAllUsers() {
        const URL = "./backend/AdminPanel.php";
        let res = await fetch(URL, { method: "GET" });
        res = await res.json();
        return res;
    }

    async function changeVal(id, type) {
        let form = new FormData();
        form.append("action", type);
        form.append("id", id);

        fetch("./backend/AdminPanel.php", {
            method: "POST",
            body: form,
            mode: "no-cors",
        })
            .then((res) => res.json())
            .then((data) => {
                users = getAllUsers();
            });
    }

    checkStatus().then((status) => {
        if (status.logged == true && status.permiss == "admin") {
            canAdmin = true;
            users = getAllUsers();
        } else {
            location.href = "./#/notFound";
        }
    });
</script>
```

Рис. 3.13. Файл AdminPanel.svelte

Також для адміністратора потрібно написати компонент який буде відповідати за обробку даних бронювань та керування відображенням цих даних на сторінці. Розробити функціонал для сортування бронювань за різними стовпцями та виконання різних дій з бронюваннями, такими як зміна статусу, видалення або оновлення дати.

Спочатку ми імпортуємо необхідні модулі та компоненти, які будуть використовуватися для цього файлу [див. рис. Файл AllReservations.svelte частина 1].

У нас є кілька змінних, які використовуються для зберігання даних та керування відображенням на сторінці:

- canRender: вказує, чи може компонент бути відображений на сторінці. Він стає true, якщо користувач ввійшов у систему і має дозвіл для перегляду цієї сторінки.
- accepted: масив, який зберігає список прийнятих бронювань.
- reservations: зберігає дані про всі бронювання.
- tempOb і archOb: тимчасові змінні для зберігання оригінальних даних про бронювання та архівних бронювань.
- sortBy і sortDir: масиви, що використовуються для зберігання інформації про тип сортування та напрямок сортування для кожного стовпця в таблиці.

```
import * as APP from "../../js/app";
import ChevronCircleUp from "svelte-icons/fa/FaChevronCircleUp.svelte";
import ChevronCircleDown from "svelte-icons/fa/FaChevronCircleDown.svelte";
import Loader from "../components/Loader.svelte";
import AllReservationsTable from "../components/AllReservationsTable.svelte";

let canRender = false;
let accepted = [];
let reservations;
let tempOb = null;
let archOb = null;

let sortBy = [undefined, undefined, undefined, undefined];
let sortDir = ["UP", "UP", "UP", "UP"];
```

Рис. 3.14. Файл AllReservations.svelte частина 1

У нас є декілька асинхронних функцій, які використовуються для отримання даних з сервера та виконання дій з бронюваннями:

getAllReservations(): відправляє запит до сервера для отримання всіх бронювань. Отримані дані обробляються, а прийняті бронювання мітяться з обробленими датами [див. рис. Файл AllReservations.svelte частина 2].

```
async function getAllReservations() {
    const URL = "./backend/AllReservations.php";
    let res = await fetch(URL, { method: "GET" });
    res = await res.json();
    accepted = res.accepted;
    accepted.map(
        (e) => (e.resTime = APP.calculateDate(e.resStart, e.resTime))
    );
    tempOb = archOb = res;
    sortBy.forEach((e, i) => {
        if (e != undefined) {
            let where =
                i == 0
                    ? "waiting"
                    : i == 1
                    ? "accepted"
                    : i == 2
                    ? "declined"
                    : "archival";
            sortDir[i] = sortDir[i] == "UP" ? "DOWN" : "UP";
            setSort(i, where, e);
        }
    });
    return res;
}
```

Рис. 3.15. Файл AllReservations.svelte частина 2

changeVal(action, id, type = null, array = null): виконує різні дії з бронюваннями в залежності від переданої дії (action). Наприклад, змінює статус бронювання, видаляє бронювання або оновлює дату бронювання [див. рис. Файл AllReservations.svelte частина 3].

setSort(index, where, type): сортує бронювання відповідно до вказаного типу сортування та напрямку [див. рис. Файл AllReservations.svelte частина 4].

Ми також використовуємо функцію APP.checkStatus(), яка перевіряє статус користувача, щоб дозволити доступ до панелі бронювань. Якщо

користувач увійшов у систему та має відповідний дозвіл (адміністратор або модератор), canRender встановлюється на true, і ми отримуємо дані про бронювання з сервера [див. рис. Файл AllReservations.svelte частина 5].

```
async function changeVal(action, id, type = null, array = null) {
    let form = new FormData();
    form.append("action", action);
    form.append("id", id);
    if (action == "changeStatus") {
        form.append("status", type);
    } else if (action == "deleteReserv") {
        form.append("rentID", type);
    } else if (action == "updateReserv") {
        for (let i = 0; i < array.length; i++) {
            if (array[i].id == id) {
                form.append("resStart", array[i].resStart);
                form.append(
                    "resTime",
                    APP.calculateDays(array[i].resStart, array[i].resTime)
                );
                break;
            }
        }
    }

    fetch("./backend/AllReservations.php", {
        method: "POST",
        body: form,
        mode: "no-cors",
    })
        .then((res) => res.json())
        .then((data) => {
            reservations = getAllReservations();
            APP.displayMessage(
                !data.action,
                data.info,
                document.getElementById("header")
            );
        });
}
```

Рис. 3.16. Файл AllReservations.svelte частина 3

```
async function setSort(index, where, type) {
    if (sortBy[index] != type) {
        sortBy[index] = type;
        sortDir[index] = "UP";
    } else {
        sortDir[index] = sortDir[index] == "UP" ? "DOWN" : "UP";
    }

    if (reservations) {
        reservations[where] = Promise.resolve(
            APP.mySort(tempOb[where], sortDir[index], sortBy[index])
        );
        if (where == "accepted") {
            accepted = APP.mySort(accepted, sortDir[index], sortBy[index]);
        }
    }
}
```

Рис. 3.17. Файл AllReservations.svelte частина 4

```
APP.checkStatus().then((status) => {
    if (
        status.logged == true &&
        (status.permiss == "admin" || status.permiss == "mod")
    ) {
        canRender = true;
        reservations = getAllReservations();
    } else {
        location.href = "./#/notFound";
    }
});
```

Рис. 3.18. Файл AllReservations.svelte частина 5

3.2. Тестування проєкту

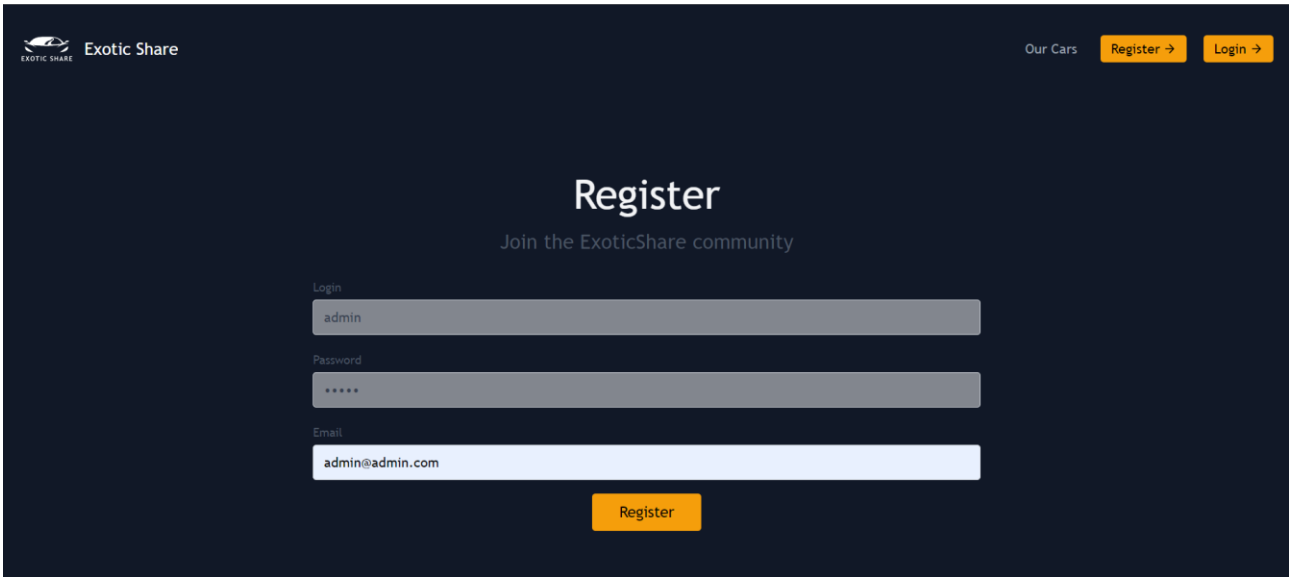

Рис. 3.19. Сторінка Реєстрації

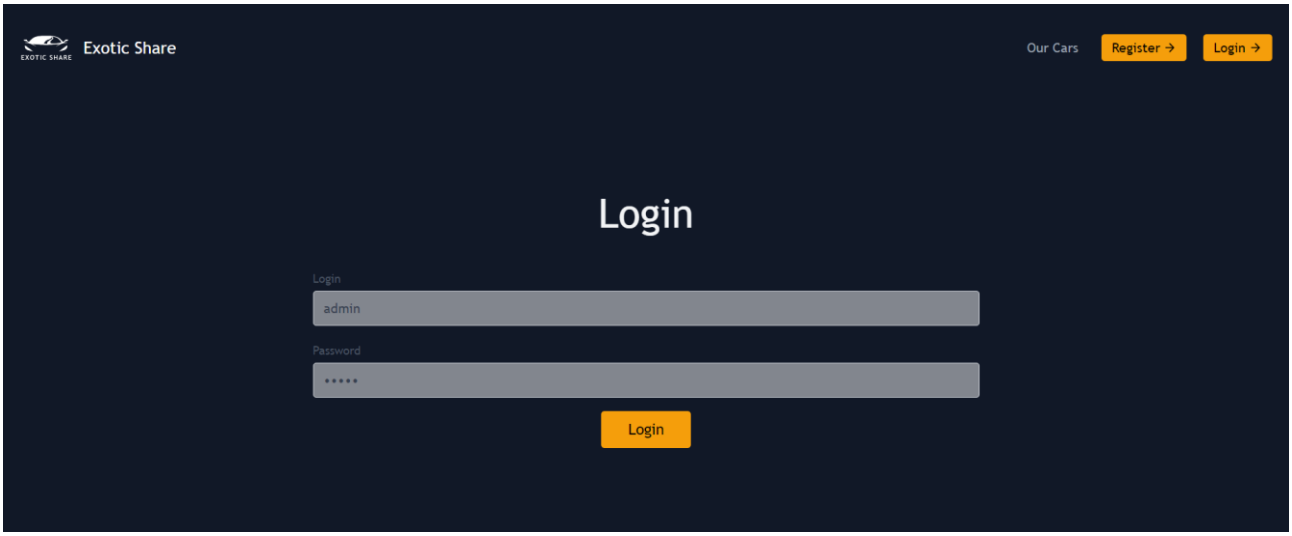

Рис. 3.20. Сторінка Авторизації

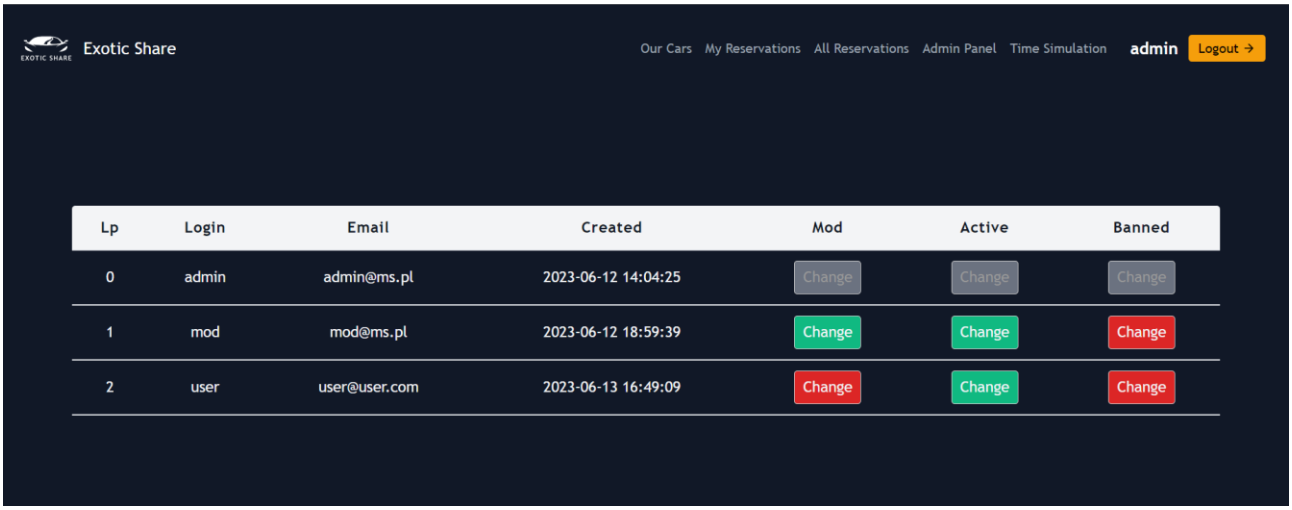

Рис. 3.21. Сторінка Керування користувачами

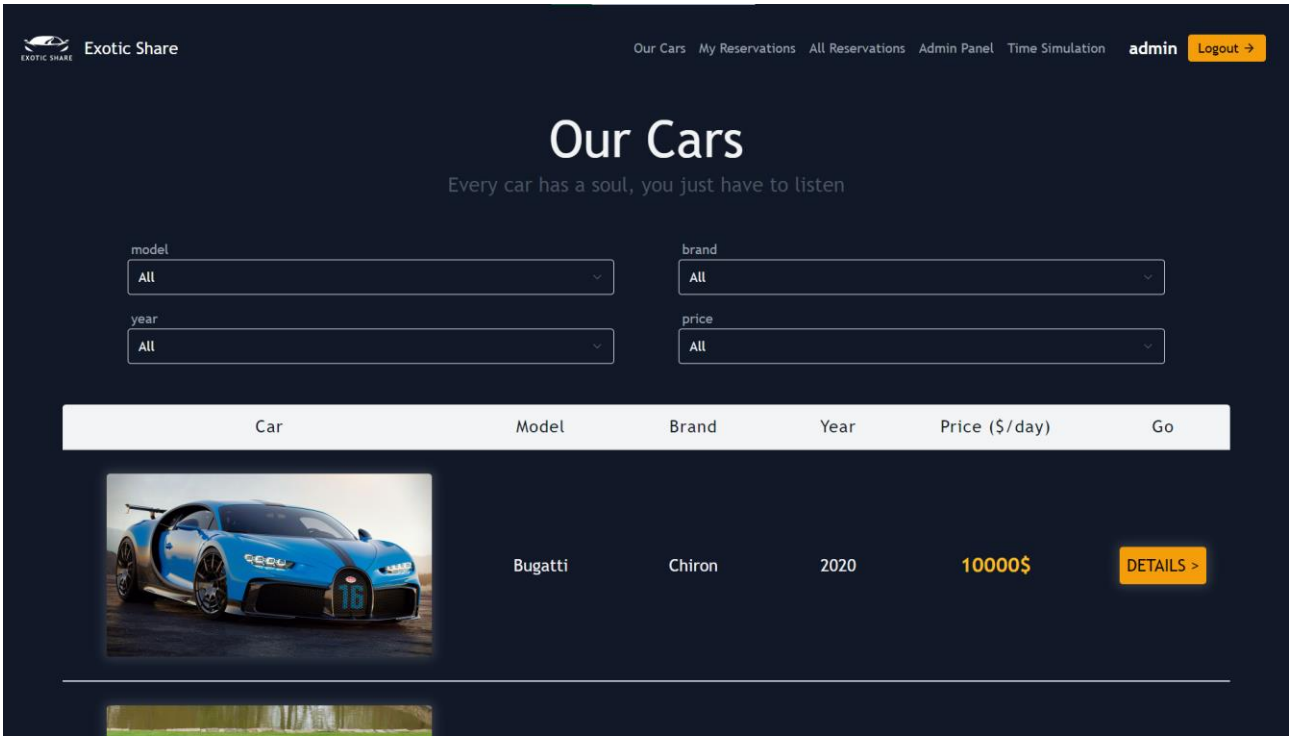

Рис. 3.22. Сторінка Всі автомобілі

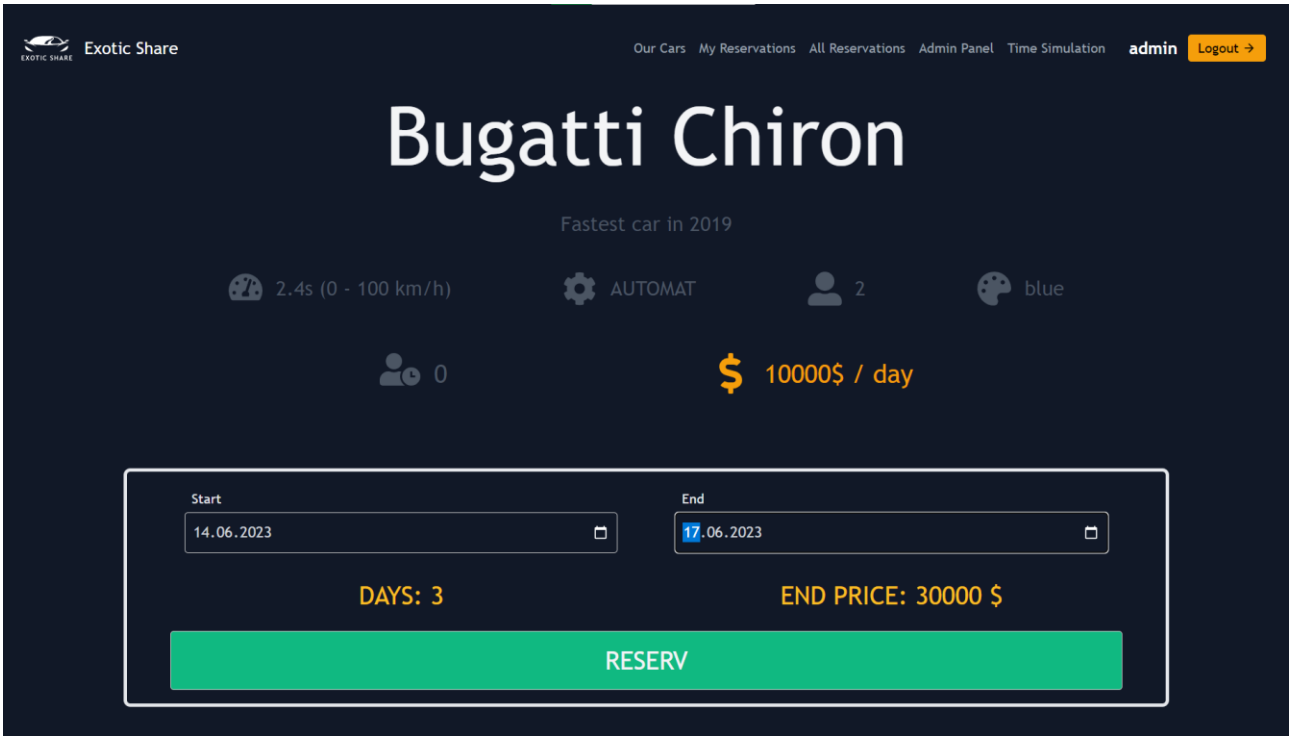

Рис. 3.23. Сторінка Оформлення автомобіля

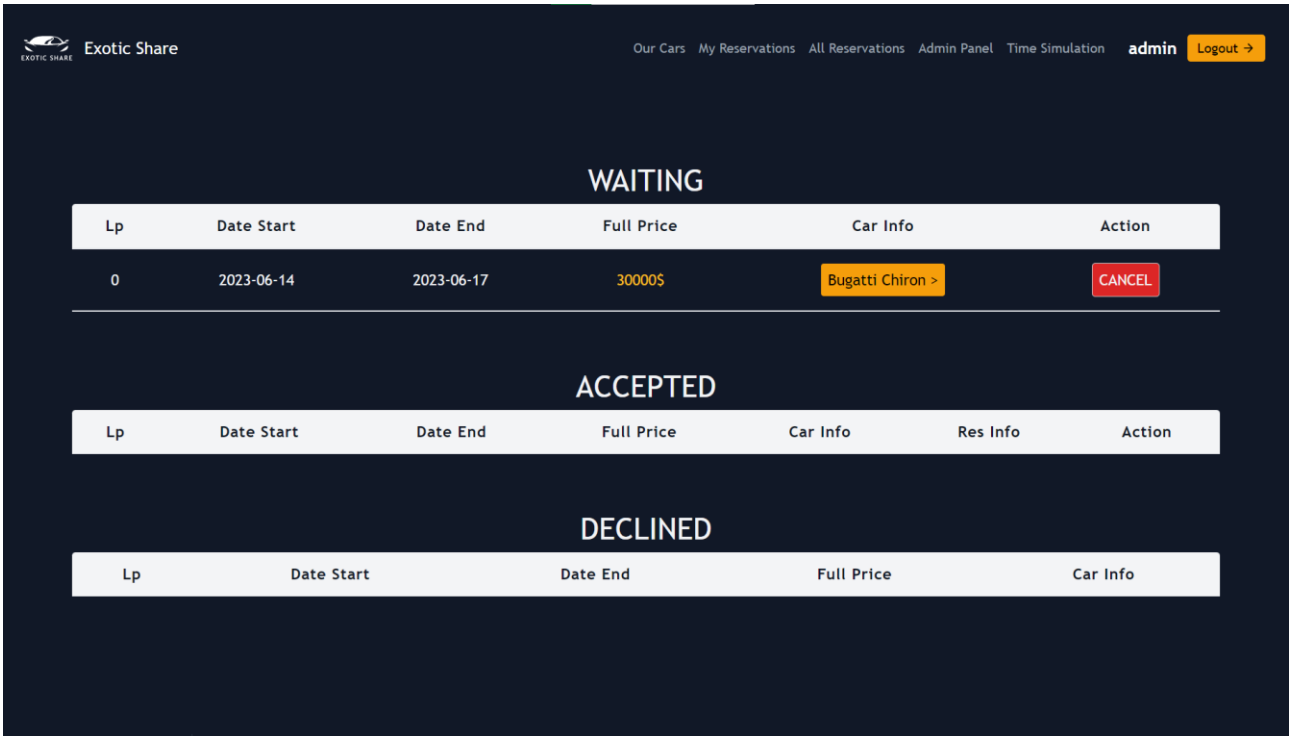

Рис. 3.24. Сторінка Мої резервації (В очікувані)

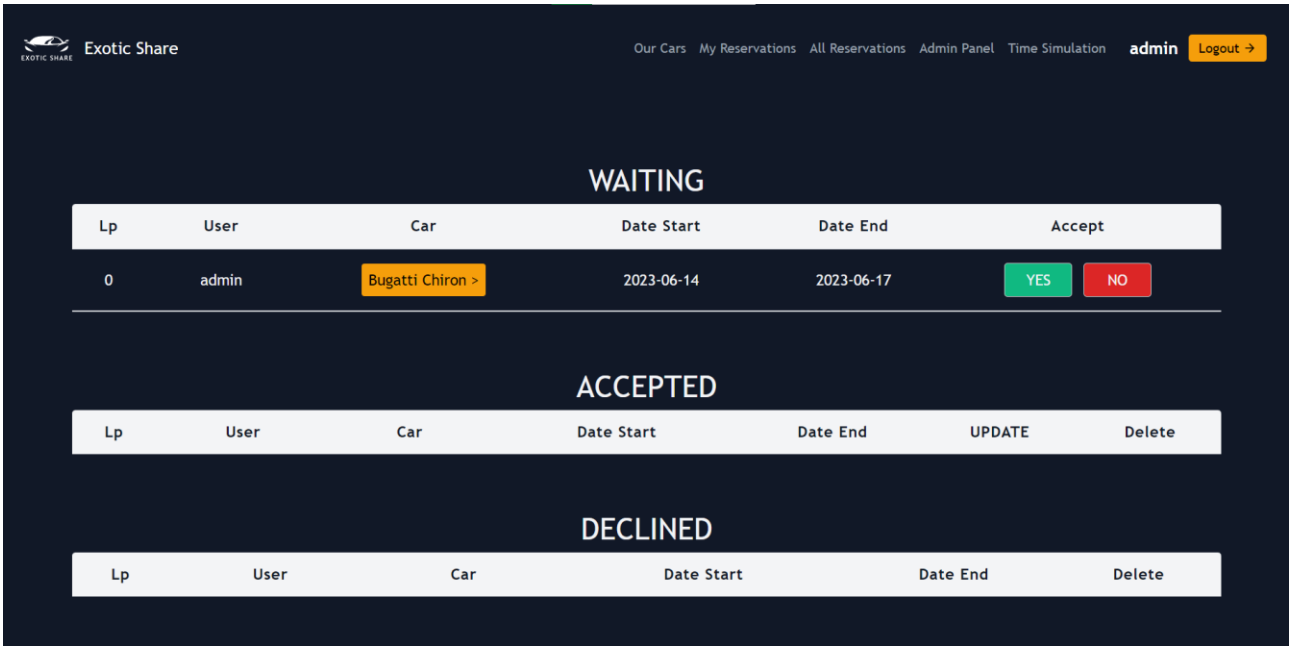

Рис. 3.25. Сторінка Керування резерваціями для адміністратора

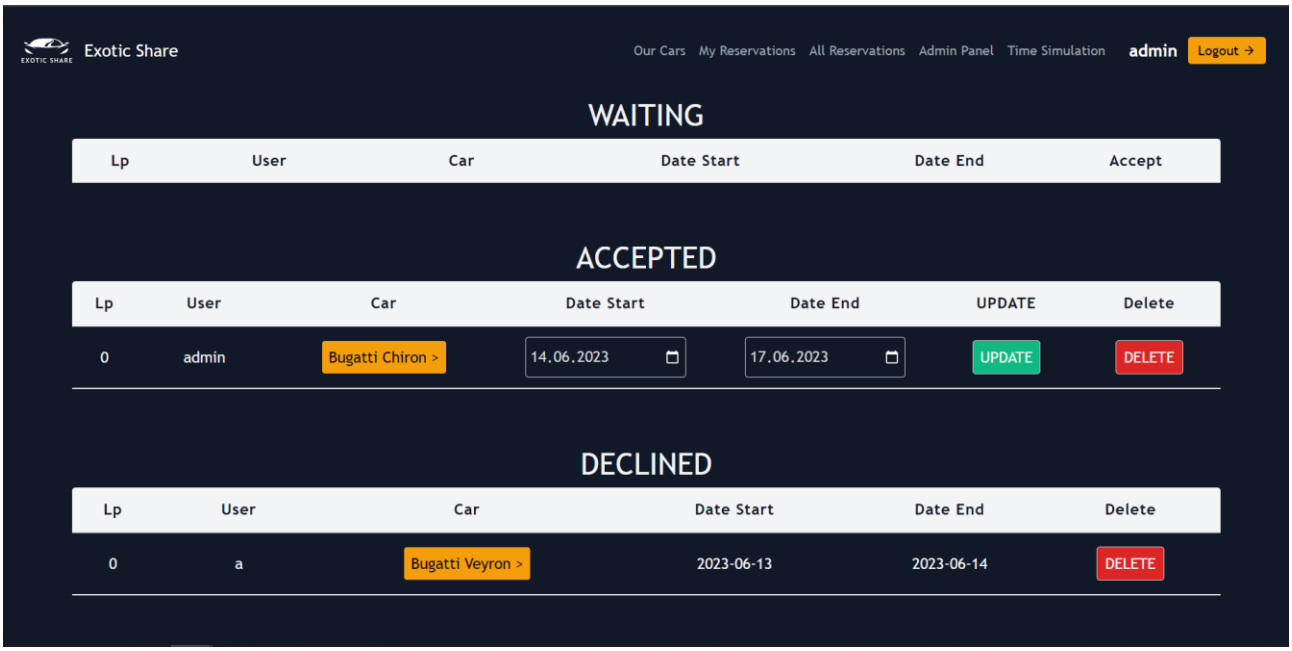

Рис. 3.26. Приклад підтвердження та відмови резервації

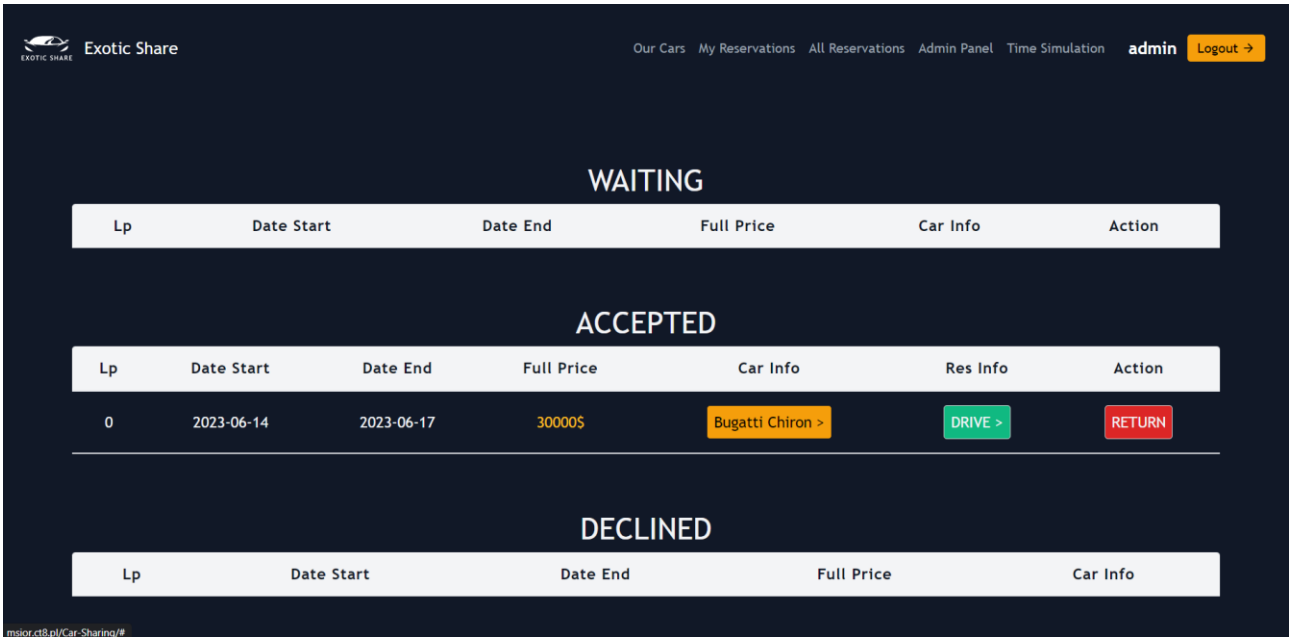

рис. 3.27 Приклад підтвердження резервації для клієнта

рис. 3.28 Сторінка інформації про резерв

ВИСНОВКИ

Розроблено програмне забезпечення для «кар шерінгу». Для створення застосунку використано фреймворк Svelte. Розроблено сторінку авторизації та реєстрації. Також передбачино сторінку автомобілів яких ви можете взяти в оренду та сторінку кліянта, який може бачити історію та статус прокату. Розроблено зручний та простий інтерфейс, який дозволить скористатись автомобілем у кілька кліків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mike Cantelon, Marc Harter, TJ Holowaychuk, Nathan Rajlich: Node.js in Action — NY, December 3, 2013.
2. Alessandro Segala: Svelte 3 Up and Running: A fast-paced introductory guide to building high-performance web applications with SvelteJS – Packt Publishing Ltd, 2020.
3. Robin Wieruch: The Road to Firebase: Your journey to master Firebase in JavaScript – Independently published (June 7, 2021).
4. Svelte. Official documentation. [Електронний ресурс] – Режим доступу: https://svelte.dev/docs Node.js. (дата звернення: 05.06.2023)
5. Node.js Documentation. [Електронний ресурс] – Режим доступу: https://nodejs.org (дата звернення: 05.06.2023)
6. React Documentation. [Електронний ресурс] – Режим доступу: https://react.dev (дата звернення: 05.06.2023)
7. Vue.js Guide. [Електронний ресурс] – Режим доступу: https://vuejs.org (дата звернення: 05.06.2023)
8. Shaheen S., Cohen A. Shared Mobility Policies for the City of the Future. – 2020. – 25 ст.
9. Becker H., Ciari F., Axhausen K. Comparing car-sharing schemes in Europe. – Transportation Research. – 2019. – №52. – P. 94–108.
10. Deloitte. Future of Mobility: Car Sharing and Urban Transport Trends. – 2021.
11. McKinsey & Company. The Future of Car Sharing: Growth and Opportunities. – 2021.
12. Libby A. Practical Svelte: Create Performant Applications with the Svelte Component Framework. – Apress, 2021. – 317 ст.
13. Holthausen S. Svelte: A Beginner’s Guide. – SitePoint, 2022. – 70 ст.
14. Volmann M. Svelte and Sapper in Action. – Manning Publications, 2020.
15. Porcello E., Banks A. Learning Svelte 3. – O’Reilly Media, 2020. – 240 ст.

ДОДАТКИ

Компонент Register.svelte

```
<script>
    import * as APP from "../js/app.js";
    import Loader from "./components/Loader.svelte";
    let canRegister = false;
    let login, password, email;

    function onRegister(e) {
        e.preventDefault();
        if (login && password && email) {
            let form = new FormData();
            form.append("login", login);
            form.append("password", password);
            form.append("email", email);

            fetch("./backend/Register.php", {
                method: "POST",
                body: form,
                mode: "no-cors",
            })
                .then((res) => res.json())
                .then((data) => {
                    if (data.locReload) {
                        location.href = "./#";
                    }
                    APP.displayMessage(
                        !data.opperation,
                        data.info,
                        document.getElementById("header")
                    );
                    if (data.opperation == true) {
                        [login, password, email] = new Array(3).fill("");
                    }
                });
        } else {
            APP.displayMessage(
                true,
                "Wrong data",
                document.getElementById("header")
            );
        }
    }

    APP.checkStatus().then((status) => {
        if (status.logged == false) {
            canRegister = true;
        } else {
            location.href = "./#";
        }
    });
</script>
```

Компонент Login.svelte

```
<script>
    import { displayMessage, checkStatus } from "../js/app.js";
    import { store_STATUS } from "../stores/storeone";
    import Loader from "./components/Loader.svelte";

    let login, password;
    let canLogin = false;

    function onLogin(e) {
        e.preventDefault();
        if (login && password) {
            let form = new FormData();
            form.append("login", login);
            form.append("password", password);

            fetch("./backend/Login.php", {
                method: "POST",
                body: form,
                mode: "no-cors",
            })
                .then((res) => res.json())
                .then((data) => {
                    if (data.locReload || data.opperation) {
                        location.href = "./#";
                        let stat = checkStatus();
                        store_STATUS.update((v) => stat);
                    } else {
                        displayMessage(
                            !data.opperation,
                            data.info,
                            document.getElementById("header")
                        );        }        });
        } else {
            displayMessage(
                true,
                "Wrong data",
                document.getElementById("header")
            );        }    }

    checkStatus().then((status) => {
        if (status.logged == false) {
            canLogin = true;
        } else {
            location.href = "./#";
        }    });
</script>
```

Компонент CarList.svelte

```
<script>
    import { carListTypes, mySort } from "../js/app";
    import ChevronCircleUp from "svelte-icons/fa/FaChevronCircleUp.svelte";
    import ChevronCircleDown from "svelte-icons/fa/FaChevronCircleDown.svelte";
    import CarListRow from "./components/carListRow.svelte";
    import Loader from "./components/Loader.svelte";

    let canRender = false;
```

```
let carList = getCars();
let tempList = [];
let archList = [];

let sortedBy;
let sortDirect = "UP";

let models = [];
let brands = [];
let years = [];
let prices = [];

let filtrModel = "all";
let filtrBrand = "all";
let filtrYear = "all";
let filtrPrice = "all";

async function getCars() {
    const URL = "./backend/CarList.php";
    let res = await fetch(URL, { method: "GET" });
    res = await res.json();
    tempList = res.cars;
    archList = res.cars;
    models = res.cars
        .map((e) => e.model)
        .filter((e, i, a) => a.indexOf(e) == i);
    fixFilters(res.cars);
    canRender = true;
    return res;
}

function fixFilters(tab) {
    brands = tab.map((e) => e.brand).filter((e, i, a) => a.indexOf(e) == i);
    years = tab.map((e) => e.year).filter((e, i, a) => a.indexOf(e) == i);
    prices = tab.map((e) => e.price).filter((e, i, a) => a.indexOf(e) == i);

    if (filtrBrand != "all") {
        if (brands.indexOf(filtrBrand) == -1) {
            filtrBrand = "all";
        }
    }
    if (filtrYear != "all") {
        if (years.indexOf(filtrYear) == -1) {
            filtrYear = "all";
        }
    }
    if (filtrPrice != "all") {
        if (prices.indexOf(filtrPrice) == -1) {
            filtrPrice = "all";
        }        }    }
function carDetail(id) {
    location.href = `./#/carInfo/${id}`;
}

function filtrCars(type, select) {
    sortedBy = undefined;
    sortDirect = "UP";
```

```
        if (type == "model") {
            filtrModel = select.value;
            filtrPrice = "all";
            filtrBrand = "all";
            filtrYear = "all";
        } else if (type == "brand") {
            filtrBrand = select.value;
        } else if (type == "year") {
            filtrYear = select.value;
        } else if (type == "price") {
            filtrPrice = select.value;
        }

        if (carList) {
            if (filtrModel == "all") {
                tempList = Array.from(archList);
            } else {
                tempList = archList.filter((e) => e.model == filtrModel);
                fixFilters(tempList);
            }
            if (filtrBrand != "all") {
                tempList = tempList.filter((e) => e.brand == filtrBrand);
                fixFilters(tempList);
            }
            if (filtrYear != "all") {
                tempList = tempList.filter((e) => e.year == filtrYear);
                fixFilters(tempList);
            }
            if (filtrPrice != "all") {
                tempList = tempList.filter((e) => e.price == filtrPrice);
                fixFilters(tempList);
            }

            carList = Promise.resolve({
                cars: tempList,
            });        }    }

    function setSort(type) {
        if (sortedBy != type) {
            sortedBy = type;
            sortDirect = "UP";
        } else {
            sortDirect = sortDirect == "UP" ? "DOWN" : "UP";
        }
        if (carList) {
            carList = Promise.resolve({
                cars: mySort(tempList, sortDirect, sortedBy),
            });        }    }
</script>
```

Компонент CarInfo.svelte

```
<script>
    export let params;
    import { displayMessage } from "../js/app";
    import Loader from "./components/Loader.svelte";
    import MyIcon from "./components/MyIcon.svelte";
```

```
let today = new Date(Date.now());

let startDate, endDate, logged;

let resDays = 0;
let pricePerDay = 0;

let carInfo = getCarInfo(params.id);

async function getCarInfo(id) {
    let form = new FormData();
    form.append("id", id);
    const URL = "./backend/CarInfo.php";
    let res = await fetch(URL, {
        method: "POST",
        body: form,
        mode: "no-cors",
    });
    res = await res.json();
    pricePerDay = parseInt(res.data.price);
    logged = res.logged;
    return res;
}

function setDate(type, input) {
    if (type == "start") {
        startDate = input.target.value;
    } else {
        endDate = input.target.value;
    }
    if (startDate && endDate) {
        let sTime = new Date(startDate).getTime();
        let eTime = new Date(endDate).getTime();
        let sub = eTime - sTime;
        resDays = Math.round(sub / 86400000);
    }
}

async function reservCar(queueStatus = false) {
    if (queueStatus) {
        location.href = "./#/myReservations";
    } else {
        if (resDays > 0 && logged == true) {
            let form = new FormData();
            form.append("offerID", params.id);
            form.append("endPrice", resDays * pricePerDay);
            form.append("resStart", startDate);
            form.append("resTime", resDays);
            const URL = "./backend/MakeReserv.php";
            let res = await fetch(URL, {
                method: "POST",
                body: form,
                mode: "no-cors",
            });
            res = await res.json();
            if (res.status == true) {
                carInfo = getCarInfo(params.id);
```

```
                }
                displayMessage(
                    !res.status,
                    res.info,
                    document.getElementById("header")
                );
            } else {
                displayMessage(
                    true,
                    logged == false
                        ? "You must be logged in"
                        : "Wrong reservation date",
                    document.getElementById("header")
                );     }    }    }
</script>
```

Компонент MyReservations.svelte

```
<script>
    import { checkStatus, displayMessage } from "../js/app";
    import Loader from "./components/Loader.svelte";
    import MyReservationsTable from "./components/MyReservationsTable.svelte";

    let canRender = false;
    let reservations;

    async function getMyReservations() {
        const URL = "./backend/MyReservations.php";
        let res = await fetch(URL, { method: "GET" });
        res = await res.json();
        return res;
    }

    async function changeVal(action, id, rentID = false) {
        let form = new FormData();
        form.append("action", action);
        form.append("id", id);
        if (action == "returnCar") {
            form.append("rentID", rentID);
        }

        fetch("./backend/MyReservations.php", {
            method: "POST",
            body: form,
            mode: "no-cors",
        })
            .then((res) => res.json())
            .then((data) => {
                reservations = getMyReservations();
                displayMessage(
                    !data.action,
                    data.info,
                    document.getElementById("header")
                );   });    }
    checkStatus().then((status) => {
        if (status.logged == true) {
            canRender = true;
```

```
            reservations = getMyReservations();
        } else {
            location.href = "./#";
        }    });
</script>
```

Компонент AdminPanel.svelte

```
<script>
    import { checkStatus } from "../../js/app";
    import AdminPanelRow from "../components/AdminPanelRow.svelte";
    import Loader from "../components/Loader.svelte";

    let canAdmin = false;
    let users;

    async function getAllUsers() {
        const URL = "./backend/AdminPanel.php";
        let res = await fetch(URL, { method: "GET" });
        res = await res.json();
        return res;
    }

    async function changeVal(id, type) {
        let form = new FormData();
        form.append("action", type);
        form.append("id", id);

        fetch("./backend/AdminPanel.php", {
            method: "POST",
            body: form,
            mode: "no-cors",
        })
            .then((res) => res.json())
            .then((data) => {
                users = getAllUsers();
            });    }

    checkStatus().then((status) => {
        if (status.logged == true && status.permiss == "admin") {
            canAdmin = true;
            users = getAllUsers();
        } else {
            location.href = "./#/notFound";
        }    });
</script>
```

Компонент AllReservations.svelte

```
<script>
    import * as APP from "../../js/app";
    import ChevronCircleUp from "svelte-icons/fa/FaChevronCircleUp.svelte";
    import ChevronCircleDown from "svelte-icons/fa/FaChevronCircleDown.svelte";
    import Loader from "../components/Loader.svelte";
```

```
import AllReservationsTable from "../components/AllReservationsTable.svelte";

let canRender = false;
let accepted = [];
let reservations;
let tempOb = null;
let archOb = null;

let sortBy = [undefined, undefined, undefined, undefined];
let sortDir = ["UP", "UP", "UP", "UP"];

async function getAllReservations() {
    const URL = "./backend/AllReservations.php";
    let res = await fetch(URL, { method: "GET" });
    res = await res.json();
    accepted = res.accepted;
    accepted.map(
        (e) => (e.resTime = APP.calculateDate(e.resStart, e.resTime))
    );
    tempOb = archOb = res;
    sortBy.forEach((e, i) => {
        if (e != undefined) {
            let where =
                i == 0
                    ? "waiting"
                    : i == 1
                    ? "accepted"
                    : i == 2
                    ? "declined"
                    : "archival";
            sortDir[i] = sortDir[i] == "UP" ? "DOWN" : "UP";
            setSort(i, where, e);
        }
    });
    return res;
}

async function changeVal(action, id, type = null, array = null) {
    let form = new FormData();
    form.append("action", action);
    form.append("id", id);
    if (action == "changeStatus") {
        form.append("status", type);
    } else if (action == "deleteReserv") {
        form.append("rentID", type);
    } else if (action == "updateReserv") {
        for (let i = 0; i < array.length; i++) {
            if (array[i].id == id) {
                form.append("resStart", array[i].resStart);
                form.append(
                    "resTime",
                    APP.calculateDays(array[i].resStart, array[i].resTime)
                );
                break;
            }     }      }

    fetch("./backend/AllReservations.php", {
        method: "POST",
```

```
            body: form,
            mode: "no-cors",
        })
            .then((res) => res.json())
            .then((data) => {
                reservations = getAllReservations();
                APP.displayMessage(
                    !data.action,
                    data.info,
                    document.getElementById("header")
                );
            });    }

    async function setSort(index, where, type) {
        if (sortBy[index] != type) {
            sortBy[index] = type;
            sortDir[index] = "UP";
        } else {
            sortDir[index] = sortDir[index] == "UP" ? "DOWN" : "UP";
        }

        if (reservations) {
            reservations[where] = Promise.resolve(
                APP.mySort(tempOb[where], sortDir[index], sortBy[index])
            );
            if (where == "accepted") {
                accepted = APP.mySort(accepted, sortDir[index], sortBy[index]);
            }        }    }

    APP.checkStatus().then((status) => {
        if (
            status.logged == true &&
            (status.permiss == "admin" || status.permiss == "mod")
        ) {
            canRender = true;
            reservations = getAllReservations();
        } else {
            location.href = "./#/notFound";
        }
    });
</script>
```