МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И РОБОТОТЕХНИКИ

Кафедра “Системы автоматизированного проектирования”

**Пояснительная записка к курсовой работе**

по курсу “Программирование сетевых приложений”

**«**Веб-приложение управления заказами в салоне проката автомобилей**»**

**Исполнитель:** студент гр.10702314 Сухоцкий М.А.

**Руководитель:** Кочуров В.А.

Минск 2016

**СОДЕРЖАНИЕ**

[**1.** **ЦЕЛЬ** 3](#_Toc469407663)

[**2.** **ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ. ЦЕЛЬ.** 3](#_Toc469407664)

[**3.** **ВЫБОР СУБД И СРЕДСТВ ПРОЕКТИРОВАНИЯ ПРИЛОЖЕНИЯ** 4](#_Toc469407665)

[**4.** **ПОРЯДОК РАЗРАБОТКИ ПРОЕКТА** 6](#_Toc469407666)

[**4.1** **Создание базы данных** 6](#_Toc469407667)

[**4.2** **Доступ к данным** 1](#_Toc469407668)1

[**4.3** **Создание интерфейса и приложения** 1](#_Toc469407669)2

[**ВЫВОД** 3](#_Toc469407670)0

# **ЦЕЛЬ**

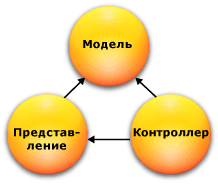
Целью данной курсовой работы является создание web-приложения, представляющего собой веб-приложение управления заказами в салоне проката автомобилей.

Данное веб приложение должно облегчить задачу создания и хранения контрактов на аренду автомобилей и поиска подходящего средства передвижения среди огромного многообразия схожих по характеристикам.

# **ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ. ЦЕЛЬ.**

Шаблон архитектуры Model-View-Controller (MVC) разделяет приложение на три основных компонента: модель, представление и контроллер. Платформа ASP.NET MVC представляет собой альтернативу схеме веб-форм ASP.NET при создании веб-приложений. Платформа ASP.NET MVC является легковесной платформой отображения с широкими возможностями тестирования и, подобно приложениям на основе веб-форм, интегрирована с существующими функциями ASP.NET, например с главными страницами и проверкой подлинности на основе членства. Платформа MVC определяется в сборке System.Web.Mvc.

Шаблон разработки MVC



MVC представляет собой стандартный шаблон разработки, знакомый многим специалистам. Некоторые типы веб-приложений имеют преимущества при создании на платформе MVC. Для других может быть целесообразно использование традиционной схемы приложения ASP.NET, основанной на веб-формах и обратной передаче. В некоторых случаях возможно сочетание двух подходов: применение одной схемы не исключает использования другой.

В состав платформы MVC входят следующие компоненты.

Модели. Объекты моделей являются частями приложения, реализующими логику для домена данных приложения. Объекты моделей часто получают и сохраняют состояние модели в базе данных. Например, объект Product может получать информацию из базы данных, работать с ней, а затем записывать обновленные данные в таблицу Products базы данных SQL Server.

В небольших приложениях эта модель подразумевает концептуальное, а не физическое разделение. Например, если приложение только считывает набор данных и отправляет его в представление, то физический слой модели и связанных классов отсутствует. В этом случае набор данных принимает роль объекта модели.

Представления. Представления служат для отображения пользовательского интерфейса приложения. Пользовательский интерфейс обычно создается на основе данных модели. Примером может служить представление для редактирования таблицы Products, которое содержит текстовые поля, раскрывающиеся списки и флажки, значения которых основаны на текущем состоянии объекта Product.

Контроллеры. Контроллеры осуществляют взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. В приложении MVC представления только отображают данные, а контроллер обрабатывает вводимые данные и отвечает на действия пользователя. Например, контроллер может обрабатывать строковые значения запроса и передавать их в модель, которая может использовать эти значения для отправки запроса в базу данных.

Шаблон MVC позволяет создавать приложения, различные аспекты которых (логика ввода, бизнес-логика и логика интерфейса) разделены, но достаточно тесно взаимодействуют друг с другом. Эта схема указывает расположение каждого вида логики в приложении. Пользовательский интерфейс располагается в представлении. Логика ввода располагается в контроллере. Бизнес-логика находится в модели. Это разделение позволяет работать со сложными структурами при создании приложения, так как обеспечивает одновременную реализацию только одного аспекта. Например, разработчик может сконцентрироваться на создании представления отдельно от бизнес-логики.

Связь между основными компонентами приложения MVC также облегчает параллельную разработку. Например, один разработчик может создавать представление, другой — логику контроллера, а третий — бизнес-логику модели.

# **ВЫБОР СУБД И СРЕДСТВ ПРОЕКТИРОВАНИЯ ПРИЛОЖЕНИЯ**

Выбор системы управления баз данных (СУБД) представляет собой сложную многопараметрическую задачу и является одним из важных этапов при разработке приложений баз данных. Выбранный программный продукт должен удовлетворять как текущим, так и будущим потребностям предприятия, при этом следует учитывать финансовые затраты на приобретение необходимого оборудования, самой системы, разработку необходимого программного обеспечения на ее основе, а также обучение персонала. Кроме того, необходимо убедиться, что новая СУБД способна принести предприятию реальные выгоды

Для крупных web узлов предпочтительно выбирать СУБД, которые хорошо зарекомендовали себя в плане стабильной работы с потоками данных корпоративного уровня. В первое время как нельзя кстати подойдет СУБД Microsoft SQL Server Express 12. Ее преимуществом является то, что в случае большого роста числа клиентов интернет провайдера, все данные будет легко перевести в Microsoft SQL Server, а на данном этапе все ограничения никак не повлияют на процесс.

Само приложение будет разрабатываться при помощи Microsoft ASP.NET MVC 5 Web Application. Это мощная распространённая платформа, обладающая высокой надежностью и удобством разработки web-приложений любой сложности.

# **ПОРЯДОК РАЗРАБОТКИ ПРОЕКТА**

Работа над данным проектом начинается с создания базы данных. Прежде всего, следует определиться с её структурой и создать необходимые таблицы.

Далее разрабатывается непосредственно web-приложение. Для этого была использована Microsoft Visual Studio 2013.

# **Создание базы данных**

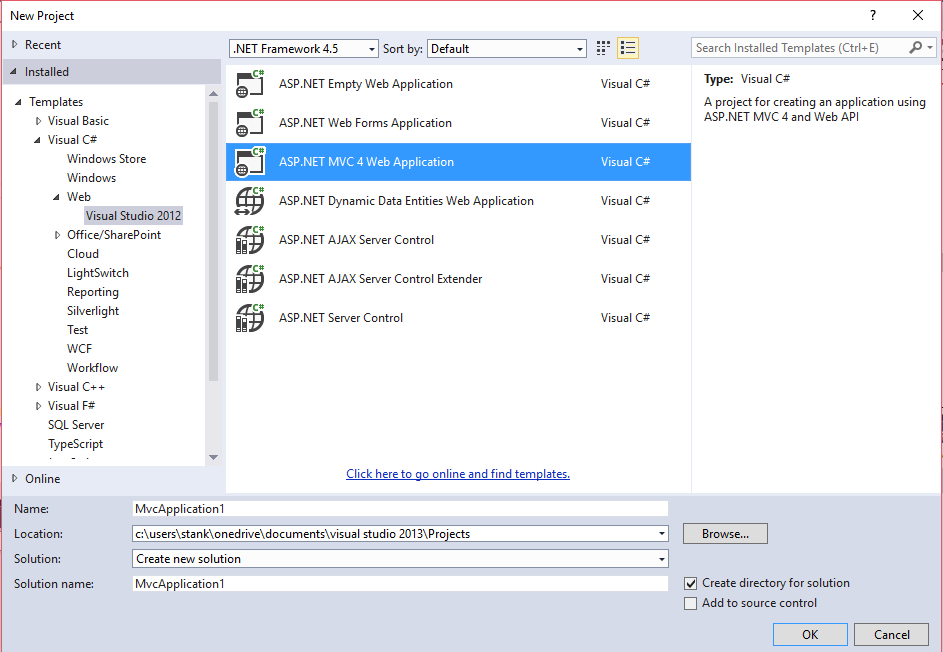
Мы будем использовать локальную базу данных, встроенную в Microsoft Visual Studio 2013. Для это создадим проект выбирается New Project, выбирается ASP.NET, MVC 4, что продемонстрировано на рисунке 

Рисунок 4.1.1 – Создание проекта

Далее в папку App\_Data добавляем базу данных. Для этого нажимаем правой клавишей на папку App\_Data -> Add -> New Item. В всплывшем диалоговом окне выбираем База данных SQL Server и называем DataBase1, что показано на рисунке 4.1.2.

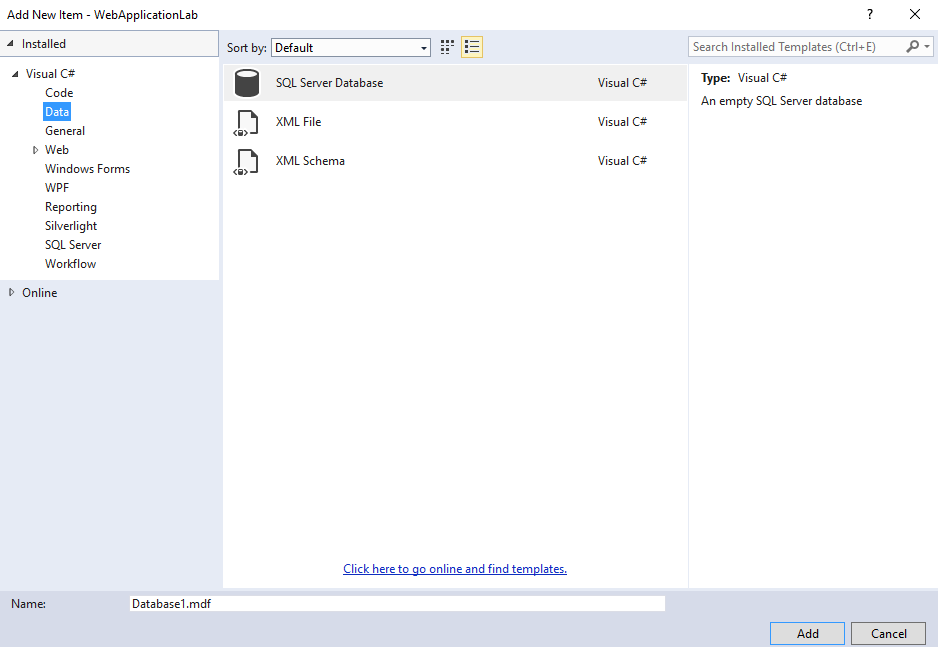


Рисунок 4.1.2 – Добавление базы данных

Теперь можем открыть базу данных в Обозревателе Серверов и, нажав правой клавишей мыши на пункт Таблицы, выбрать в всплывающем меню Add New Table (рис. 4.1.3).

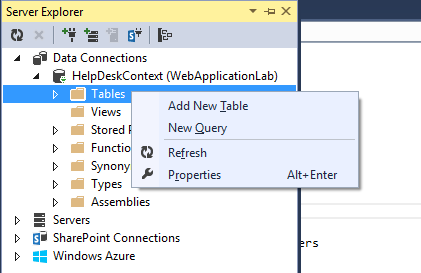


Рисунок 4.1.3 – Добавление новой таблицы базы данных

Таблица Contracts состоит из нескольких полей:

* Поле Id – идентификатор контракта в системе;
* Поле Starts – время подписания контракта;
* Поле Ends – время конца срока действия контракта;
* Поле Price – значение денежной суммы;
* Поле UserId – ID заказчика;
* Поле СarId – ID автомобиля;

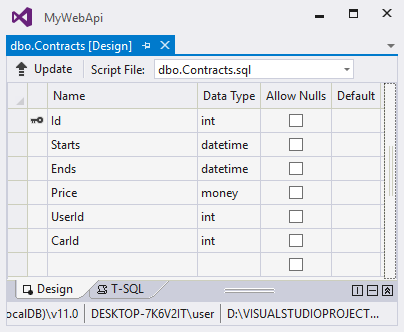


Рисунок 4.1.4 – Таблица Contracts

Таблица Cars состоит из 9 полей полей:

Поле Id – устанавливает ID автомобиля;

* Поле Id – идентификатор автомобиля в системе
* Поле Name – название автомобиля;
* Поле Photo – фото автомобиля;
* Поле Available – доступность автомобиля;
* Поле Color – цвет автомобиля;
* Поле Description – описание автомобиля;
* Поле Range – пробег автомобиля;
* Поле Year – год выпуска автомобиля;
* Поле Features – особенности автомобиля;

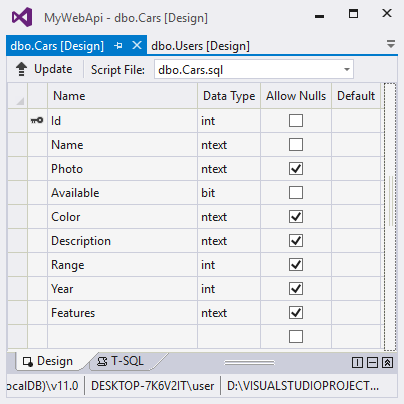


Рисунок 4.1.5 – Таблица Cars

Таблица Users состоит из 5 полей:

* Поле Id – идентификатор пользователя;
* Поле RoleId – идентификатор роли пользователя в системе;
* Поле Name – имя пользователя;
* Поле Phone – телефон пользователя;

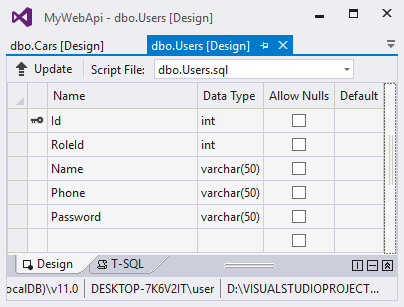


Рисунок 4.1.6 – Таблица Users

Таблица Countries состоит из 3 полей:

* Поле Id – идентификатор роли пользвателя;
* Поле Name – название роли;

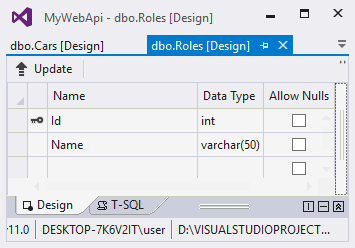


Рисунок 4.1.7 – Таблица Roles

# **Доступ к данным**

После создания таблиц создается модель классов базы данных с помощью дополнения Entity Framework. Пользуясь моделью можно взаимодействовать с самой базой данных, абстрагируясь от SQL запросов.

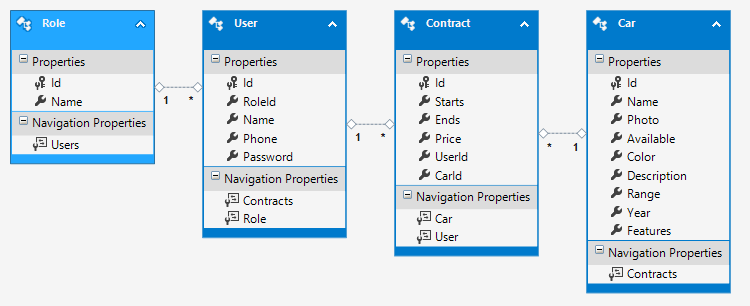
4.2.1. 

Рисунок 4.2.1 – Модель базы данных

# **Создание интерфейса и приложения**

Данное web-приложение использует страницу-шаблон. Она полезна тем, что позволяет избегать повторения, сэкономить время, составить общее оформление для создания последующих страниц.

Для создания мастер-страницы в папку Views/shared добавляем представление и называем его \_layout.cshtml.

MasterPage будет состоять из блоков. Для этого используется тэг <div>.

1. Шапка (class = “container”)

Шапка статична и содержит кнопки для перехода на главную страницу, кнопку регистрации, а также меню, в котором есть ссылки для перехода на другие страницы web-приложения. Шапка показана на рисунке 4.3.1.

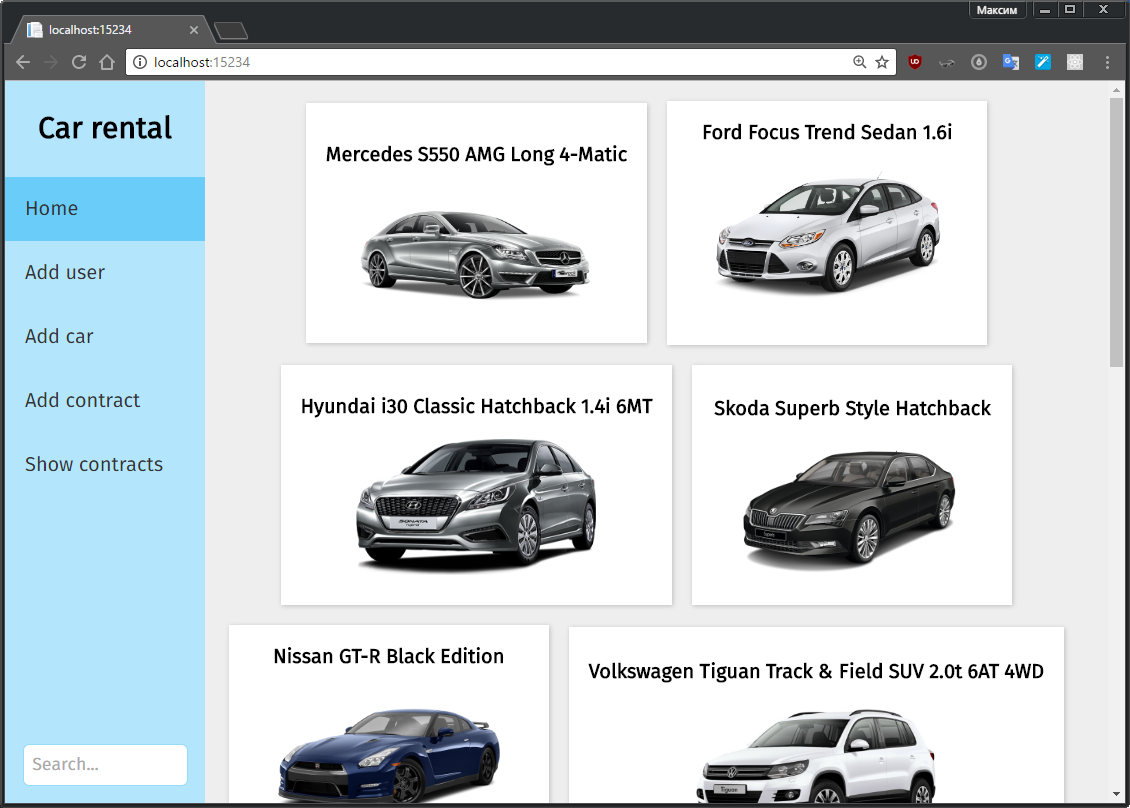


Рисунок 4.3.1 – Шапка

define(function (require) {

const React = require('react');

const Link = require('reactRouter').Link;

const Route = require('reactRouter').Route;

const Router = require('reactRouter').Router;

const browserHistory = require('reactRouter').browserHistory;

const AddCar = require('components/AddCar');

const Home = require('components/Home');

const AddUser = require('components/AddUser');

const AddContract = require('components/AddContract');

const ShowContracts = require('components/ShowContracts');

const CarView = require('components/CarView');

const ContractView = require('components/ContractView');

const Car = require('../api/Car');

class App extends React.Component {

constructor() {

super();

this.timeout = null;

}

search(event) {

event.persist();

clearTimeout(this.timeout);

this.timeout = setTimeout(() => {

var query = event.target.value;

Car.getByFilter(query).then(cars => {

browserHistory.push('/?filter=' + query);

})

}, 200);

}

render() {

return (

<main>

<aside>

<h2>Car rental</h2>

<Link to='/' activeClassName="current">Home</Link>

<Link to='/addUser' activeClassName="current">Add user</Link>

<Link to='/addCar' activeClassName="current">Add car</Link>

<Link to='/addContract' activeClassName="current">Add contract</Link>

<Link to='/showContracts' activeClassName="current">Show contracts</Link>

<span className="separator"></span>

<input type="text" placeholder="Search..." className="search" onChange={this.search.bind(this)}/>

</aside>

{this.props.children}

</main>

)

}

}

return class Routes extends React.Component {

render() {

return (

<Router history={browserHistory}>

<Route component={App}>

<Route path="/" component={Home}></Route>=

<Route path="/addCar" component={AddCar}></Route>

<Route path="/addUser" component={AddUser}></Route>

<Route path="/addContract" component={AddContract}></Route>

<Route path="/showContracts" component={ShowContracts}></Route>

<Route path="/car/:id" component={CarView}></Route>

<Route path="/contract/:id" component={ContractView}></Route>

</Route>

</Router>

)

}

};

});

Листинг 4.3.2 – Код тела страницы

Здесь Routes – класс для вызова тела других страниц.

Также добавим в наш проект таблицу стилей CSS, чтобы украсить оформления web-приложения. Для этого создадим папку styles и в этой папке создадим документ main.sass

Основной функционал web-приложения состоит из общедоступных страниц:

1. Страница, состоящая из карточек доступных авто;
2. Страница, содержащая подробную информацию о каждом авто;
3. Страница регистрации пользователей;
4. Страница добавления автомобилей;
5. Страница контрактов;

Страница, содержащая контракты из таблицы Contracts. При этом в есть ссылка с названием автомобиля, ведущая на страницу об этом авто. Кнопка создания доступна только при входе с аккаунта администратора (рис. 4.3.3). Код данной страницы приведен в Листинге 4.3.4. Здесь Layout = "/components/ShowContracts.jsx" значит, что страница будет использовать мастер страницу. Страница использует перечисление объектов типа Contract из таблицы Contracts.

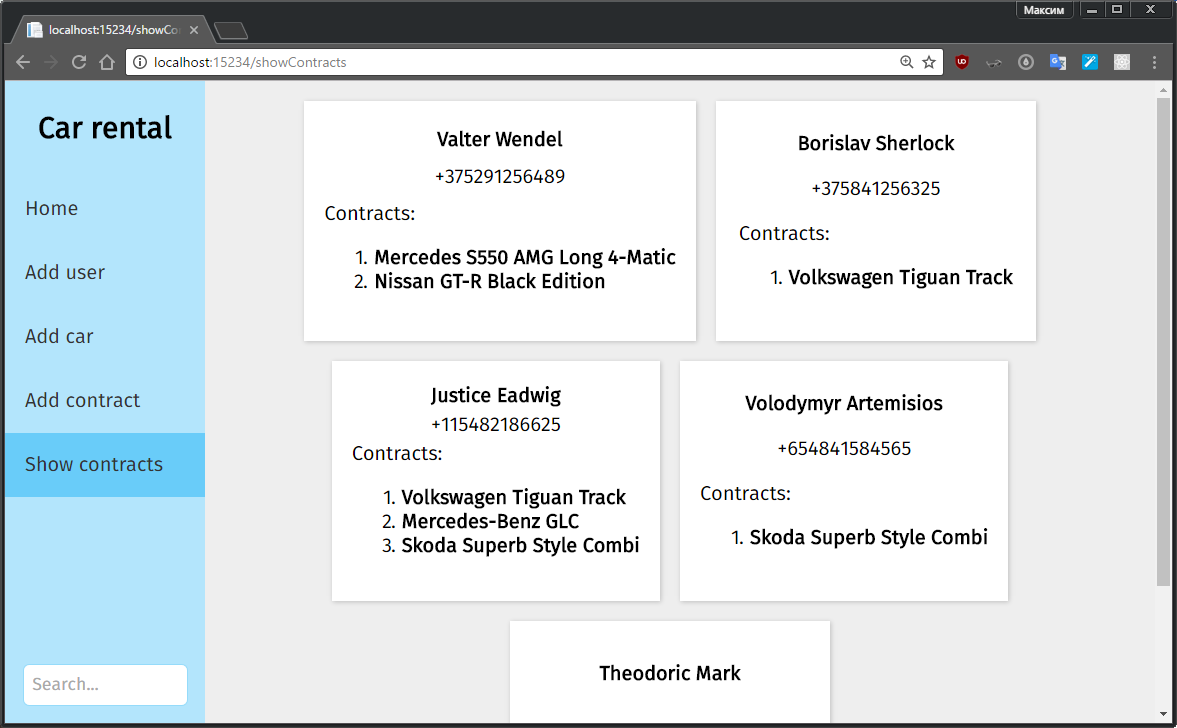


Рисунок 4.3.3 – Страница просмотра контрактов

define(require => {

const React = require('react');

const User = require('../api/User');

const Link = require('reactRouter').Link;

return class ShowUsers extends React.Component {

constructor() {

super();

User.get().then(data => {

this.setState({

users: data

});

})

this.state = {

users: []

};

}

renderContracts(user) {

return user.Contracts.map(contract => {

return <li key={contract.Car.Id}>

<Link to={'/contract/' + contract.Id}>{contract.Car.Name}</Link>

</li>

})

}

render() {

var users = this.state.users;

if (!users.length)

return <p>Loading!</p>;

return (

<content>

{users.map(user => {

return (

<card-component key={user.Id}>

<strong>{user.Name}</strong>

<span>{user.Phone}</span>

<div>

Contracts: {

!user.Contracts.length ? <span>No contracts!</span> : <ol>{this.renderContracts(user)}</ol>

}

</div>

</card-component>

)

})}

</content>

)

}

};

}); Листинг 4.3.4 – Код страницы просмотра контрактов

Рассмотрим подробнее страницу добавления контракта. Страница доступна только администратору. Страница содержит поля для заполнения. Пример такой страницы приведен на рисунке 4.3.5 ниже.

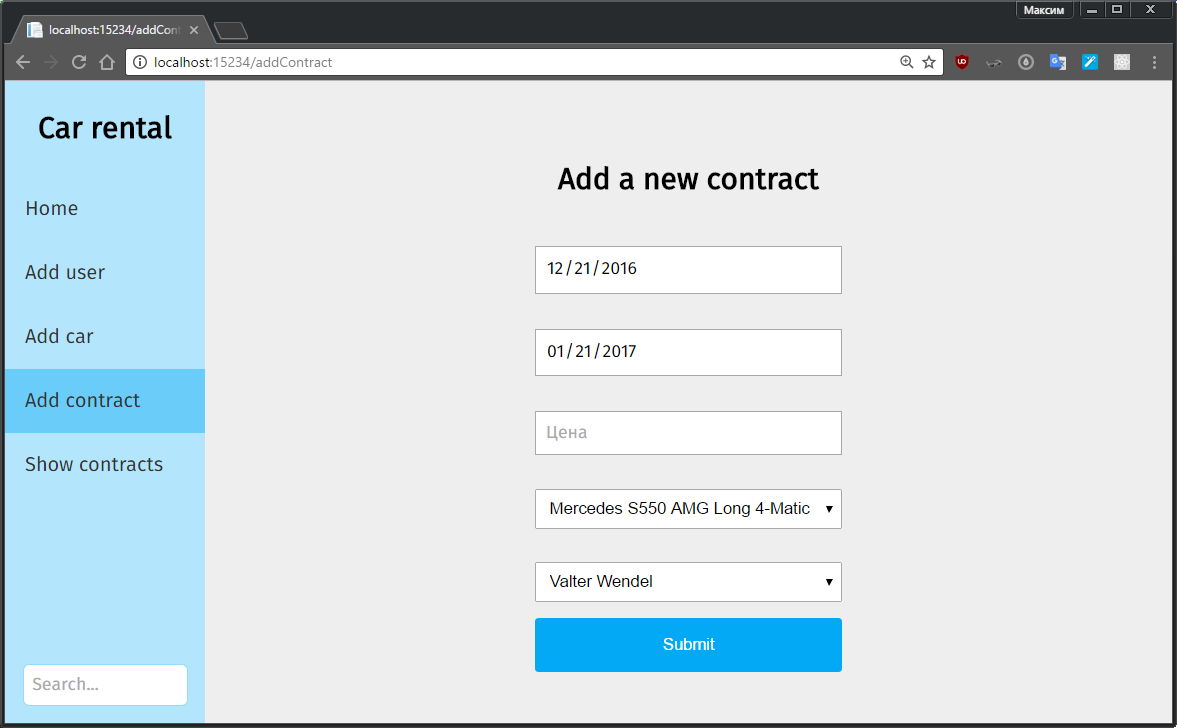


Рисунок 4.3.3 – Страница добавления заказа

define(require => {

const React = require('react');

const Link = require('reactRouter').Link;

const Contract = require('../api/Contract');

const Car = require('../api/Car');

const User = require('../api/User');

var contract = {

starts: '',

ends: '',

price: '',

userId: '',

carId: '',

}

return class AddContract extends React.Component {

constructor() {

super();

var date = new Date();

var futureDate = new Date();

futureDate.setMonth(futureDate.getMonth() + 1);

this.state = {

cars: [],

users: [],

starts: date.toISOString(),

ends: futureDate.toISOString()

}

Car.get().then(cars => this.setState({

cars,

carId: cars[0].Id

}));

User.get().then(users => this.setState({

users,

userId: users[0].Id

}))

}

onSubmit(event) {

const fields = ['starts', 'ends', 'price', 'userId', 'carId'];

var formData = new FormData();

for (let field of fields)

formData.append(field, this.state[field]);

Contract.post(formData).then(response => {

console.log(response);

})

event.preventDefault();

}

onStartsChange(event) {

this.setState({starts: event.length ? event : event.target.value})

}

onEndsChange(event) {

this.setState({ends: event.length ? event : event.target.value})

}

onPriceChange(event) {

this.setState({price: event.target.value})

}

onCarIdChange(event) {

this.setState({carId: event.target.value})

}

onUserIdChange (event) {

this.setState({userId: event.target.value})

}

getCurrentDate() {

var date = new Date();

return date.toLocaleDateString('ru-RU').split('.').reverse().join('-')

}

getFutureDate() {

var date = new Date();

date.setMonth(date.getMonth() + 1);

return date.toLocaleDateString('ru-RU').split('.').reverse().join('-')

}

renderCarSelect(cars) {

return cars.map(car => {

return (

<option key={car.Id} value={car.Id}>

{car.Name}

</option>

)

})

}

renderUsersSelect(users) {

return users.map(user => {

return (

<option key={user.Id} value={user.Id}>

{user.Name}

</option>

)

})

}

render() {

var users = this.state.users;

return (

<content data-flow="vertical">

<h2>Add a new contract</h2>

<form className="contractAdd">

<input

type="date"

placeholder="Start date"

value={this.getCurrentDate()}

onChange={this.onStartsChange.bind(this)}/>

<input

type="date"

placeholder="End time"

value={this.getFutureDate()}

onChange={this.onEndsChange.bind(this)}/>

<input

type="number"

placeholder="Цена"

onChange={this.onPriceChange.bind(this)}/>

<select onChange={this.onCarIdChange.bind(this)}>

{this.renderCarSelect(this.state.cars)}

</select>

<select onChange={this.onUserIdChange.bind(this)}>

{this.renderUsersSelect(this.state.users)}

</select>

<button onClick={this.onSubmit.bind(this)}>Submit</button>

</form>

</content>

)

}

};

});

Листинг 4.3.4 – Код страницы добавления контракта

Рассмотрим подробнее страницу, которое содержит информацию о конкретном заказзе. Пример такой страницы приведен на рисунке 4.3.5 ниже.

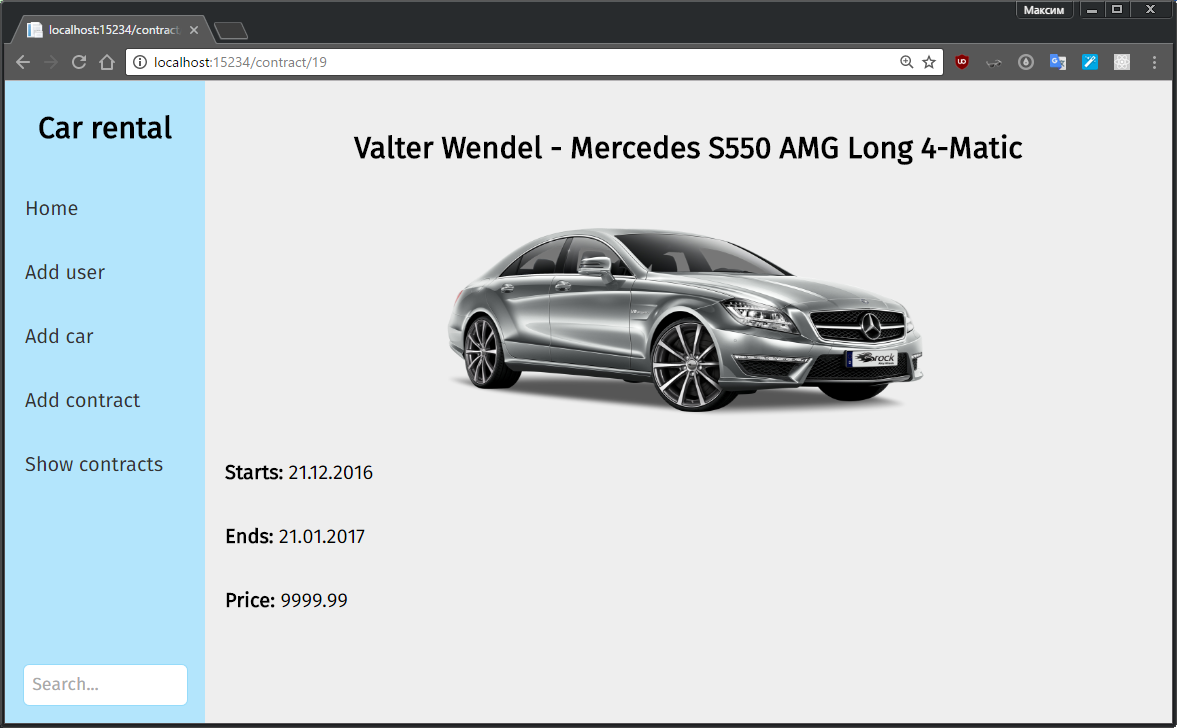


Рисунок 4.3.5 – Страница, содержащая информацию о конкретном контракте (время начала действия, окончания и стоимость)

Код данной страницы приведен в Листинге 4.3.6.

define(require => {

const React = require('react');

const Contract = require('../api/Contract');

return class ContractView extends React.Component {

constructor(props) {

super(props);

this.state = {

contract: {}

}

Contract.get(props.params.id).then(contract => this.setState({contract}))

}

render() {

var contract = this.state.contract;

if (contract.Id) {

return (

<content data-flow="vertical" data-align="left">

<h2>{contract.User.Name} - {contract.Car.Name}</h2>

<img className="photo" src={contract.Car.Photo}/>

<p>

<strong>Starts:</strong> {(new Date(contract.Starts)).toLocaleDateString('ru-RU')}

</p>

<p>

<strong>Ends:</strong> {(new Date(contract.Ends)).toLocaleDateString('ru-RU')}

</p>

<p>

<strong>Price:</strong> {contract.Price}

</p>

</content>

)

} else {

return <p>Loading...</p>

}

}

}

});

Функция принимает в качестве параметра id того заказа, подробную информацию о котором нужно вывести. Кнопки создания доступны только при входе с аккаунта администратора.

Страница, которая даёт информацию о существующих автомобилях представлена на рисунке 4.3.7. Она содержит модели из таблицы Cars базы данных. Кнопки создания доступны только при входе с аккаунта администратора. Код страницы представлен в Листинге 4.3.8.

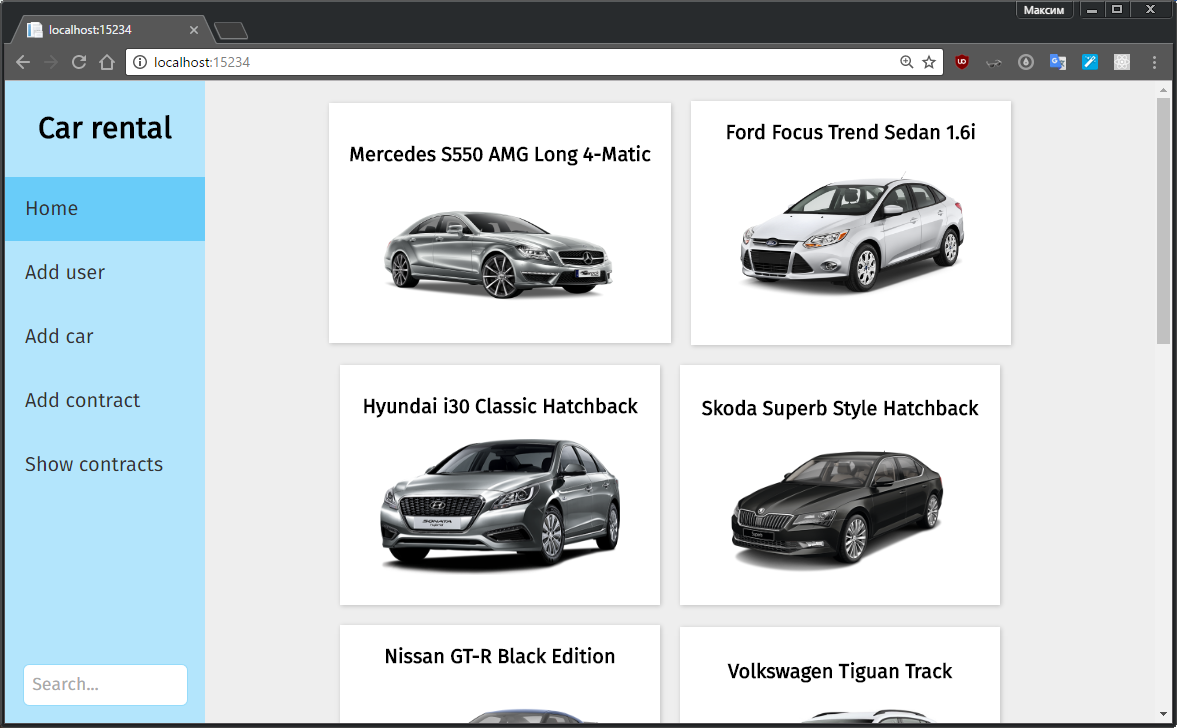


Рисунок 4.3.7 – Страница автомобилей

define(require => {

const React = require('react');

const Link = require('reactRouter').Link;

const Card = require('components/card');

const Car = require('../api/Car');

return class Home extends React.Component {

constructor(props) {

super();

this.state = {

cars: [],

searched: ''

}

if (!props.location.query.filter) {

Car.get().then(cars => this.setState({cars}))

}

}

componentDidUpdate() {

var filter = this.props.location.query.filter;

if (filter && filter != this.state.searched) {

this.setState({

searched: filter

})

Car.getByFilter(filter).then(cars => this.setState({cars}))

}

}

render() {

return (

<content>

{this.state.cars.map(car =>

<Card

key={car.Id}

id={car.Id}

name={car.Name}

photo={car.Photo}/>

)}

</content>

)

}

}

});

Листинг 4.3.8 – Код страницы ресурсов

Также существует определенная страница редактирования, доступная только для администратора. На ней есть возможность вносить коррективы в базу данных, и, вследствие, менять содержимое страниц. Она содержит поля, в которых зафиксированы данные о конкретной модели.

Код данной страницы представлен в Листинге 4.3.10.

define(require => {

const React = require('react');

const Link = require('reactRouter').Link;

const User = require('../api/User');

var user = {

name: '',

isAdmin: '',

password: '',

phone: '',

}

return class AddUser extends React.Component {

constructor() {

super();

this.state = {

users: []

};

}

onNameChange (event) {

event.preventDefault()

user.name = event.target.value;

}

onIsAdminChange (event) {

event.preventDefault()

user.isAdmin = event.target.value;

}

onPasswordChange (event) {

event.preventDefault()

user.password = event.target.value;

}

onPhoneChange (event) {

event.preventDefault()

user.phone = event.target.value;

}

onSubmit(event) {

User.post(object).then(data => {

console.log(data);

})

}

render() {

return (

<div>

<p>Добавление пользователя</p>

<form className="userAdd">

<p><input

type="tel"

placeholder="Телефон"

onChange={this.onPhoneChange}/>

</p>

<p><input

type="password"

placeholder="Пароль"

onChange={this.onPasswordChange}/>

</p>

<p><input

type="text"

placeholder="Имя"

onChange={this.onNameChange}/>

</p>

<p><input

type="text"

placeholder="Администратор?"

onChange={this.onIsAdminChange}/>

</p>

<input

type="button"

value="Добавить"

onClick={this.onSubmit}/>

</form>

</div>

)

}

};

});

Листинг 4.3.10 – Код страницы редактирования ресурсов

Ниже рассмотрена страница добавления автомобиля, доступная только для администратора. Она содержит поля названия авто, описания, возможностей,года выпуска, пробег и цвет (рис. 4.3.11).

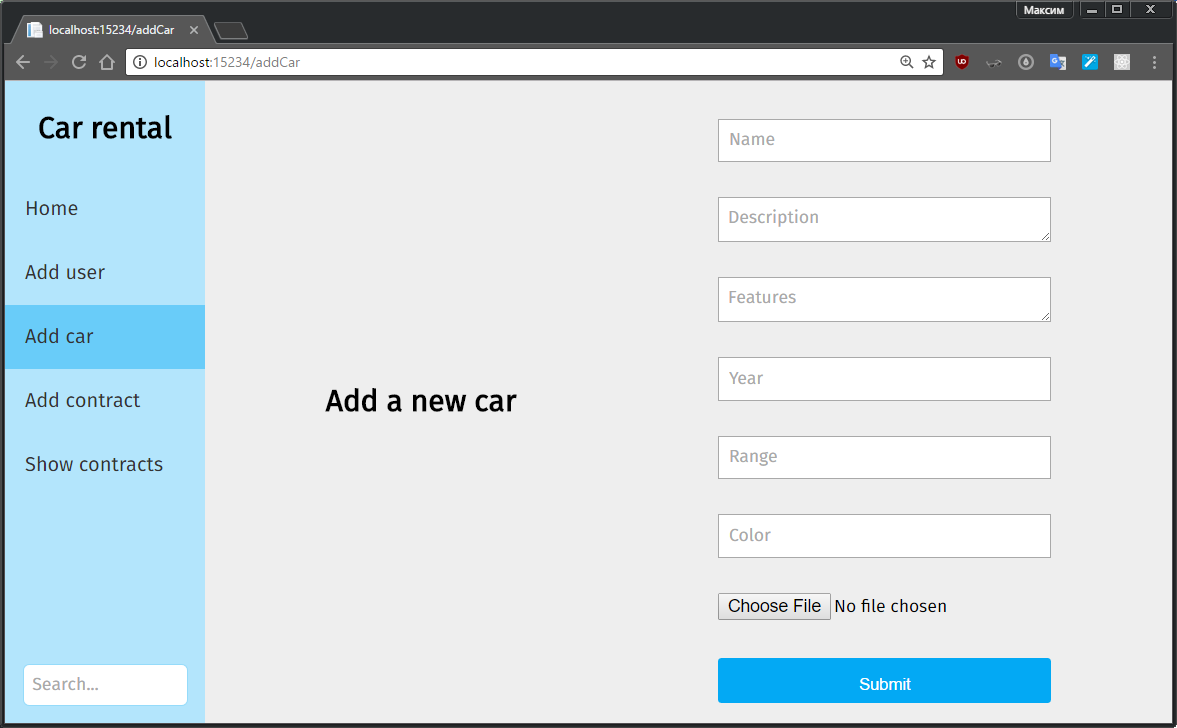


Рисунок 4.3.11 – Страница создания Континента

Код страницы приведен в Листинге 4.3.12 ниже.

define(require => {

const React = require('react');

const Link = require('reactRouter').Link;

const Car = require('../api/Car');

return class AddCar extends React.Component {

constructor() {

super();

this.state = {};

}

imageChosen(event) {

for (let file of event.target.files) {

if (!file.type.match('image.\*'))

continue;

let reader = new FileReader();

reader.addEventListener('load', () => this.setState({photo: reader.result}));

reader.readAsDataURL(file);

}

}

submit(event) {

const fields = ['name', 'description', 'features', 'year', 'range', 'color', 'photo'];

var formData = new FormData();

formData.append('available', true);

for (let field of fields)

formData.append(field, this.state[field]);

Car.post(formData);

event.preventDefault();

}

render() {

return (

<content data-flow="vertical">

<h2>Add a new car</h2>

<form>

<input type="text" placeholder="Name" onChange={e => this.setState({name: e.target.value})}/>

<textarea placeholder="Description" onChange={e => this.setState({description: e.target.value})}></textarea>

<textarea placeholder="Features" onChange={e => this.setState({features: e.target.value})}></textarea>

<input type="number" placeholder="Year" onChange={e => this.setState({year: e.target.value})}/>

<input type="number" placeholder="Range" onChange={e => this.setState({range: e.target.value})}/>

<input type="text" placeholder="Color" onChange={e => this.setState({color: e.target.value})}/>

<input type="file" onChange={this.imageChosen.bind(this)}/>

<img src={this.state.photo} className="preview"/>

<button onClick={this.submit.bind(this)}>Submit</button>

</form>

</content>

)

}

}

});

Листинг 4.3.12 – Код страницы создания автомобиля

После введения данных и нажатия кнопки данные посылаются в контроллер, где он записывает их в базу данных и сохраняет изменения.

define(function (require) {

return {

get() {

return fetch('http://localhost:15234/api/cars/')

.then(response => response.json())

},

getById(id) {

return fetch('http://localhost:15234/api/cars/' + id)

.then(response => response.json())

},

getByFilter(filter) {

return fetch('http://localhost:15234/api/cars/?filter=' + filter)

.then(response => response.json())

},

post(formData) {

return fetch('http://localhost:15234/api/cars', {

method: 'post',

body: formData

}).then(response => response.json())

}

}

})

Страницы для добавления Пользователей и Контрактов сделаны аналогично.

Страница доюбавления нового пользователя. На этой странице есть поля для ввода логина и пароля, данные из которых посылаются на котроллер. Рисунок страницы 4.3.17.

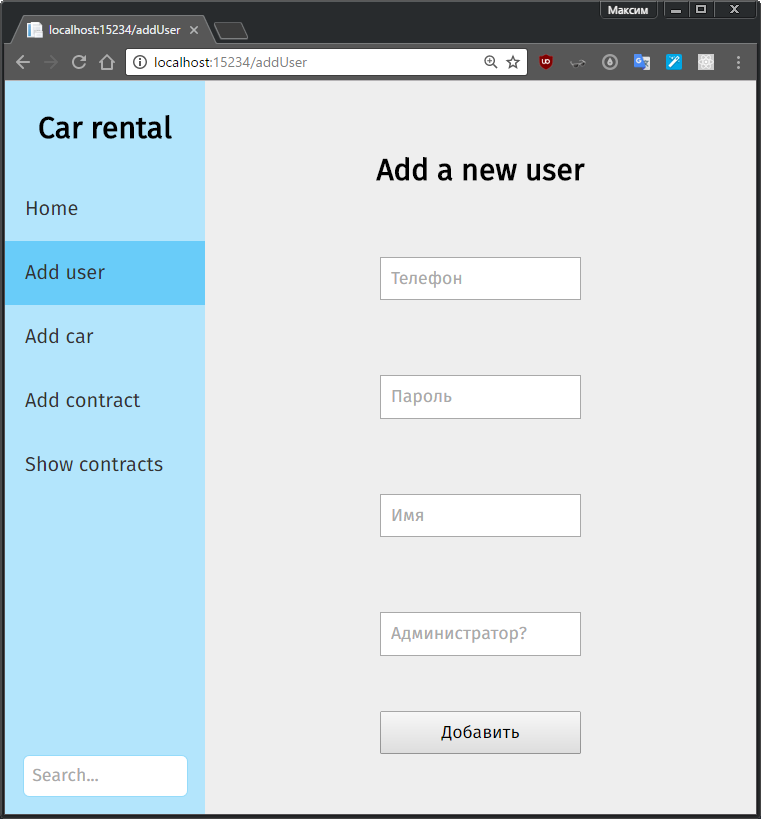


Рисунок 4.3.13 – Страница входа в аккаунт.

Код страницы на Листинге 4.3.14.

define(require => {

const React = require('react');

const Link = require('reactRouter').Link;

const User = require('../api/User');

var user = {

name: '',

isAdmin: '',

password: '',

phone: '',

}

return class AddUser extends React.Component {

constructor() {

super();

this.state = {

users: []

};

}

onNameChange (event) {

event.preventDefault()

user.name = event.target.value;

}

onIsAdminChange (event) {

event.preventDefault()

user.isAdmin = event.target.value;

}

onPasswordChange (event) {

event.preventDefault()

user.password = event.target.value;

}

onPhoneChange (event) {

event.preventDefault()

user.phone = event.target.value;

}

onSubmit(event) {

User.post(object).then(data => {

console.log(data);

})

}

render() {

return (

<content data-flow="vertical">

<h2>Add a new user</h2>

<form className="contractAdd">

<p><input

type="tel"

placeholder="Телефон"

onChange={this.onPhoneChange}/>

</p>

<p><input

type="password"

placeholder="Пароль"

onChange={this.onPasswordChange}/>

</p>

<p><input

type="text"

placeholder="Имя"

onChange={this.onNameChange}/>

</p>

<p><input

type="text"

placeholder="Администратор?"

onChange={this.onIsAdminChange}/>

</p>

<input

type="button"

value="Добавить"

onClick={this.onSubmit}/>

</form>

</content>

)

}

};

});

Листинг 4.3.14 – Код страницы добавления нового пользователя

# **ВЫВОД**

В ходе данного курсового проекта было разработано веб приложение, реализующие следующие задачи:

* Отображение контрактов, пользователей и автомобилей с возможностью их редактирования.

В результате разработки проекта были достигнуты поставленные цели и получен ожидаемый результат.