

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий  
Кафедра Информационные системы и технологии  
Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

**«Web-приложение «Аренда помещений»»**

Выполнил студент Дмитрук Илья Игоревич  
(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна  
(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Евгения Александровна  
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой \_\_\_\_\_

Минск 2024

## Содержание

|  |    |
|--|----|
| Введение .....   | 4  |
| 1 Постановка задачи .....  | 5  |
| 1.2 Аналитический обзор аналогов .....                                   | 5  |
| 1.1.1 Airbnb .....   | 5  |
| 1.1.2 Turo .....   | 6  |
| 1.1.3 Kufar .....  | 7  |
| 1.1.4 Вывод .....  | 8  |
| 1.2 Разработка функциональных требований и вариантов использования ..... | 8  |
| 1.3 Выводы по разделу .....  | 9  |
| 2 Проектирование web-приложения .....                                    | 10 |
| 2.1 Обобщенная структура управления приложением .....                    | 10 |
| 2.2 Диаграммы UML, взаимосвязь всех компонентов .....                    | 10 |
| 2.4 Выводы по разделу .....  | 11 |
| 3 Разработка web-приложения .....  | 12 |
| 3.1 Разработка клиентской части web-приложения .....                     | 12 |
| 3.2 Разработка серверной части web-приложения .....                      | 19 |
| 3.3 Выводы по разделу .....  | 22 |
| 4 Тестирование web-приложения .....                                      | 23 |
| 5 Руководство пользователя .....   | 27 |
| 5.1 Руководство пользователя .....                                       | 27 |
| 5.2 Установка приложения .....   | 33 |
| 5.3 Выводы по разделу .....  | 33 |
| Заключение .....   | 34 |
| Список используемых источников .....                                     | 35 |
| Приложение А .....   | 36 |

## Введение

Цель данной работы заключается в создании веб-приложения для аренды помещений, которое позволит пользователям выставлять в аренду свои помещения и арендовать их.

Для разработки серверной части данного веб-приложения был выбран фреймворк Express, который позволяет разрабатывать быстрое и многофункционально веб-приложение; а для разработки клиентской части была выбрана библиотека ReactJS. Для работы с БД будет использоваться модуль Prisma, который позволит создать абстракцию над объектами базы данных MSSQL.

Для гарантированной безопасности пользователей приложения в курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении картинок.

Основные требования к приложению:

- обеспечивать возможность регистрации и авторизации;
- поддерживать роли администратора, владельца, арендатора;
- позволять выставлять на аренду помещения;
- позволять просматривать выставленные на аренду помещения;
- позволять отправлять заявки на аренду помещений;
- позволять принимать заявки на аренду помещений;
- позволять просматривать арендуемые помещения;
- давать возможность фильтровать выставленные на аренду помещения.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, тестирование веб-приложения, а также инструкции по использованию приложения.

## 1 Постановка задачи

### 1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

#### 1.1.1 Airbnb

Airbnb представляет многофункциональный сервис для аренды жилых помещений различных типов. Он позволяет выбирать помещения для разных целей, в самых разнообразных местах по всему миру. Присутствует удобный фильтр, позволяющий отобрать помещения по цене, типу, наличию различных удобств. Информация о помещениях представлена довольно подробно и позволяет пользователем досконально понять, конкретно что они хотят снять. Однако, особенно на основной странице, много места занимают изображения. Довольно важной функцией данного сервиса является возможность просмотра местоположения жилья на карте.

На данном веб-сайте присутствует страница объявлений, благодаря которой пользователи могут узнать о различных мероприятиях, событиях, новостях, идеях для путешествия, и, самое главное, как для этого всего снять себе хорошее жильё.

Так же хочется выделить хороший дизайн и удобный интерфейс данного веб-приложения, а также возможность обычному пользователю выставлять свои помещения на аренду.

Интерфейс приложения представлен на рисунке 1.1.

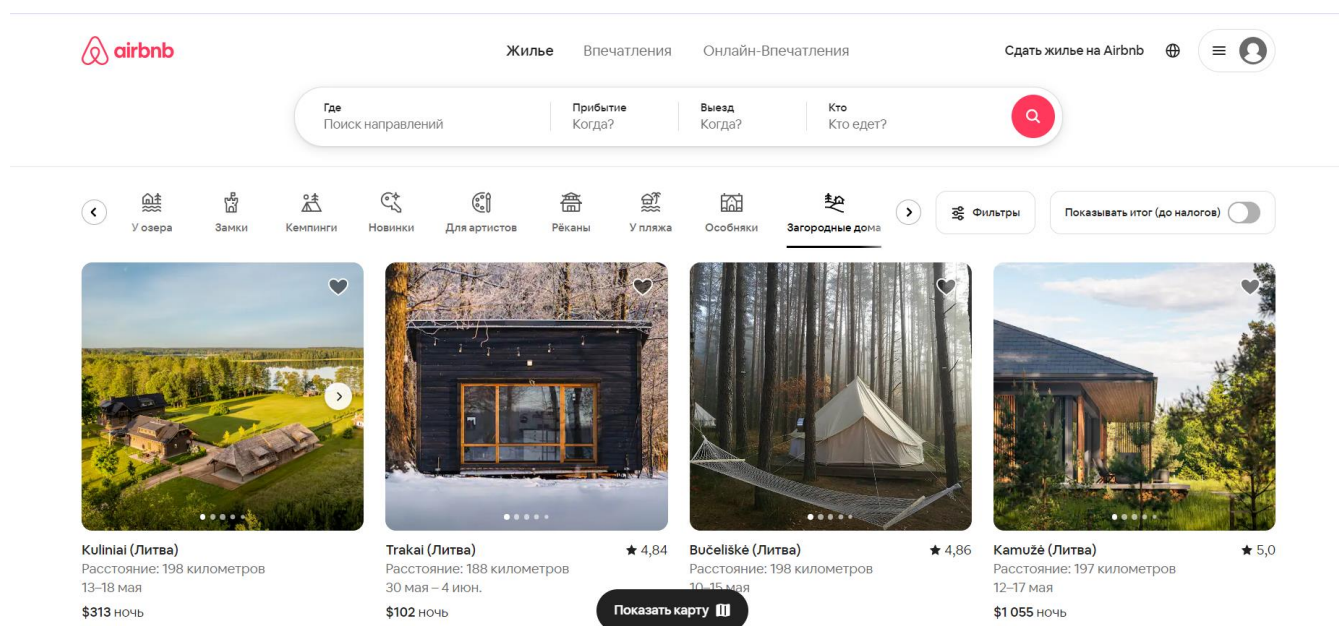


Рисунок 1.1 – Приложение «Airbnb»

Следуя из вышеописанного, можно отметить следующие преимущества и недостатки.

Преимущества:

- удобная фильтрация;
- система объявлений;
- возможность аренды помещений в разных местах мира;
- возможность выставления своего помещения на аренду;
- подробная информация о помещениях.
- удобный интерфейс.

Недостатки:

- фотографии помещений загромождают много места.

### 1.1.2 Turo

Turo – это веб-приложение для аренды автомобилей. Данный сервис позволяет выбрать нужный вам автомобиль в определённом городе, и арендовать его на определённое время. Информация о автомобилях изложена довольно кратко, но для понимания автомобиля вполне достаточно. Важной особенностью является возможность просмотра советов о аренде автомобилей в разных городах. Так же можно посмотреть информацию о владельцах автомобилей и отзыв на их.

Важным недостатком данного веб-сайта является то, что при переходе на главную страницу, мы в первую очередь встречаем подробную информацию о сайте, а не о самих автомобилях, выставленных на аренду. Для получения страницы с автомобилями, нам надо пролистать вниз и выбрать нужный нам город. Таким образом, данный веб-сайт не предоставляет пользователю свой основной функционал сразу. Так же выбор автомобилей не очень велик и отсутствует фильтр по характеристикам. Дизайн довольно приятный, но без особенностей. Интерфейс приложения представлен на рисунке 1.2.

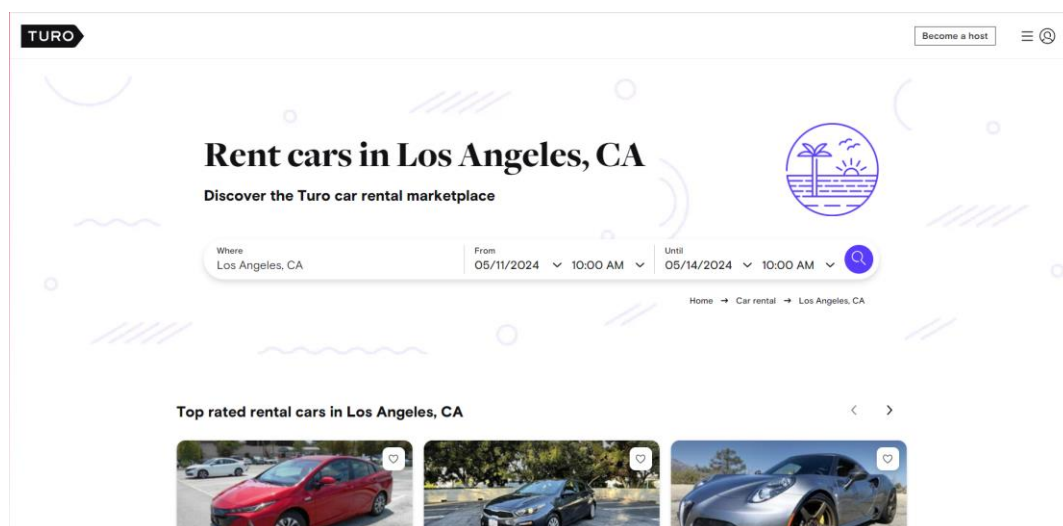


Рисунок 1.2 – Приложение «Turo»

Следуя из вышеописанного, можно отметить следующие преимущества и недостатки.

Преимущества:

- возможность просмотра советов для аренды автомобилей в разных городах;
- интерфейс без лишних элементов;
- возможность просмотра информации о владельцах автомобилей;
- возможность просмотра отзывов на владельцев автомобилей;

Недостатки:

- основной функционал веб-сайта предоставляется пользователю не сразу;
- отсутствию фильтрации по характеристикам.

### 1.1.3 Kufar

Kufar представляет с собой большую торговую площадку. На ней пользователи могут выставлять объявления о продаже различных вещей, а также совершать покупки этих вещей. Приложение также позволяет выставлять эти вещи на аренду и арендовать их. На аренду могут выставляться в том числе и помещения. Аренда происходит путём личного договора с владельцем. В данном веб-приложении отсутствует сохранение информации о том, что уже арендовано или куплено.

На сайте можно отфильтровать вещи по типам и характеристикам. Причём, характеристики довольно гибкие, то есть можно самостоятельно описать различный критерий предмета.

Дизайн веб-сайта простой, но очень много лишних элементов интерфейса. Приложение представлено на рисунке 1.3.

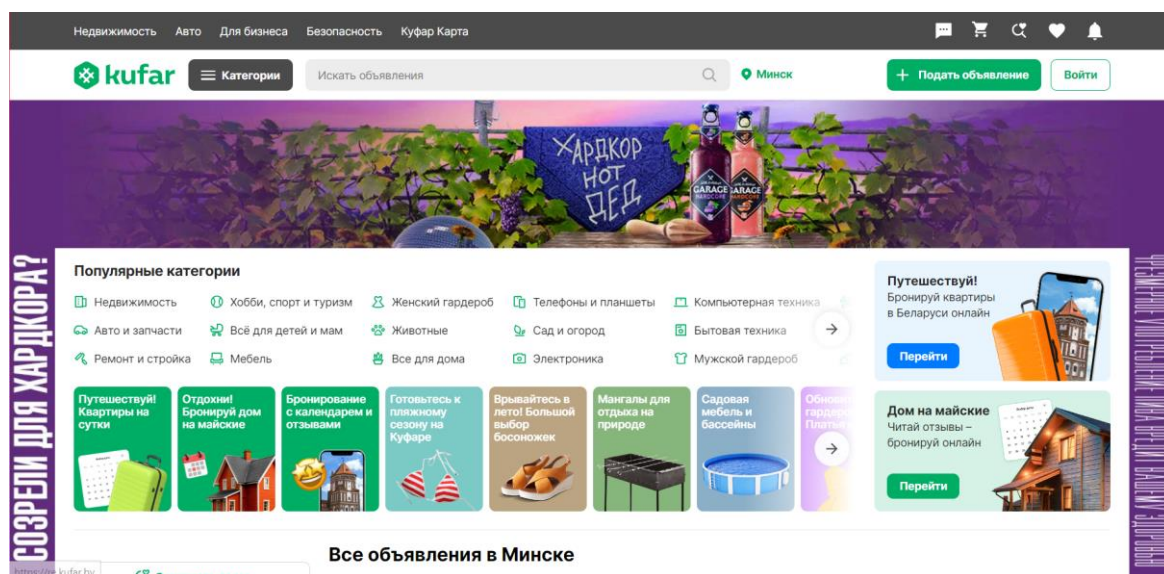


Рисунок 1.3 – Приложение «Kufar»

Следуя из вышеописанного, можно отметить следующие преимущества и недостатки.

Преимущества:

- возможность самостоятельно описывать характеристики предметов;
- разнообразие типов предметов;
- удобная фильтрация;

Недостатки:

- очень много лишних элементов интерфейса и дизайна;
- отсутствие сохранения информации о покупках и арендах.

#### **1.1.4 Вывод**

Было рассмотрено три аналогичных решения Airbnb и Turo и Kufar. Исследовав аналоги, было принято решение что приложение по количеству функционала будет ориентироваться на Airbnb, из Turo взята идея о просмотре информации о владельцах, а из Kufar – гибкое описание характеристик помещений, выставленных на аренду.

### **1.2 Разработка функциональных требования и вариантов использования**

Приложение аренды помещений должно обеспечивать функционал для регистрации и авторизации, который позволит идентифицировать пользователей и для каждого пользователя предлагать только тот функционал, который относится к нему. Для обычного пользователя – арендатора, приложении должно предлагать возможность просмотра выставленных на аренду помещений и выбора их критериев, которые ему важны. Пользователь должен иметь возможность просмотра информации о помещении и отправке заявки на его аренду. После отправки заявки, информация о данной заявке должна отобразиться у владельца, и он должен её принимать. После принятия заявки, остальные заявки на аренду данного помещения должны удаляться, а само помещение больше не должно отображаться в списке помещений, выставленных на аренду. Владелец так же может отклонить заявку. Основным же функционалом владельца является добавления, редактирование и удаление помещений с аренды. При добавлении помещения, владелец должен выбрать его тип, а также описать его характеристики. При удалении помещения, из базы данных должны удаляться описания его характеристик. Администратор должен иметь возможность добавлять, изменять и удалять пользователей. При чём, при изменении учётных данных пользователя, администратор может изменять любую информация, в то время как каждый пользователь может у себя изменить только неаутентификационную информацию. Так же в возможности администратора должно входит выставление объявлений. Диаграмма вариантов использования представлена на рисунке 1.4.



Рисунок 1.4 – Диаграмма вариантов использования

Диаграмма UML вариантов использования позволяет понять, что доступно каждой роли, доступной в данном веб-приложении.

### 1.3 Выводы по разделу

В данном разделе мы рассмотрели аналогичные веб-приложения, проанализированы их преимущества и недостатки, и были взяты из них идеи для моего веб-приложения.

Также в данном разделе рассмотрели весь основной функционал, который должен присутствовать в конечном приложении. Данный функционал рассчитан на то, чтобы повелителю было просто разобраться, как пользоваться приложением, а также для удобной аренды помещений. Была приведена UML диаграмма, на которой все и показано.



## 2 Проектирование web-приложения

### 2.1 Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для общения должны входить функции для удобного отправления заявок на аренду, принятия заявок, добавления, удаления и редактирования помещений.

### 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.1.

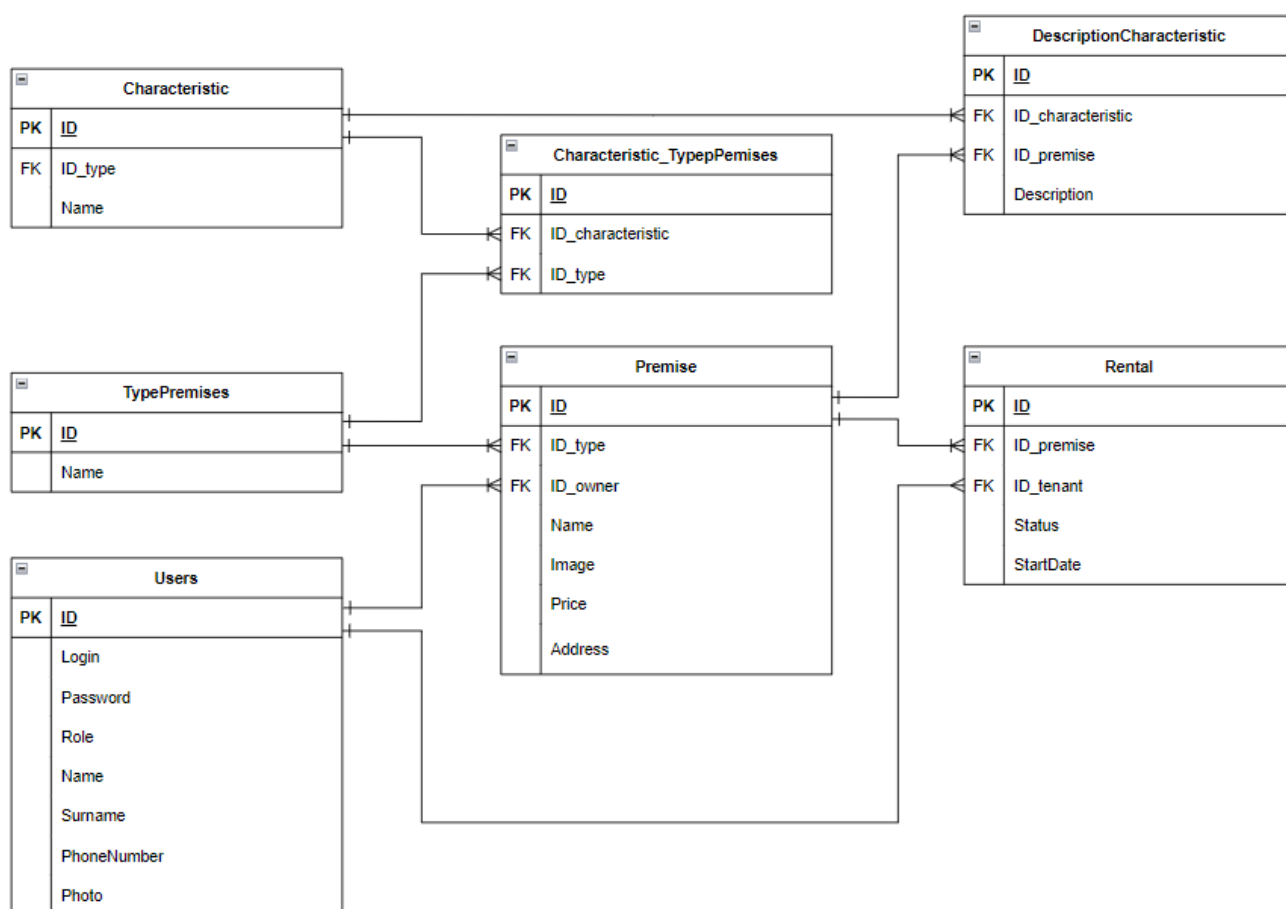


Рисунок 2.1 – Логическая схема базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи «один-ко-многим», «многие-ко-многим». Например, таблица Users связана с таблицами Premise и Rental через внешние ключи.

Для того понимания структуры данного приложения и как его разворачивать нам может помочь диаграмма развертывания. Данная диаграмма показана на рисунке 2.2.

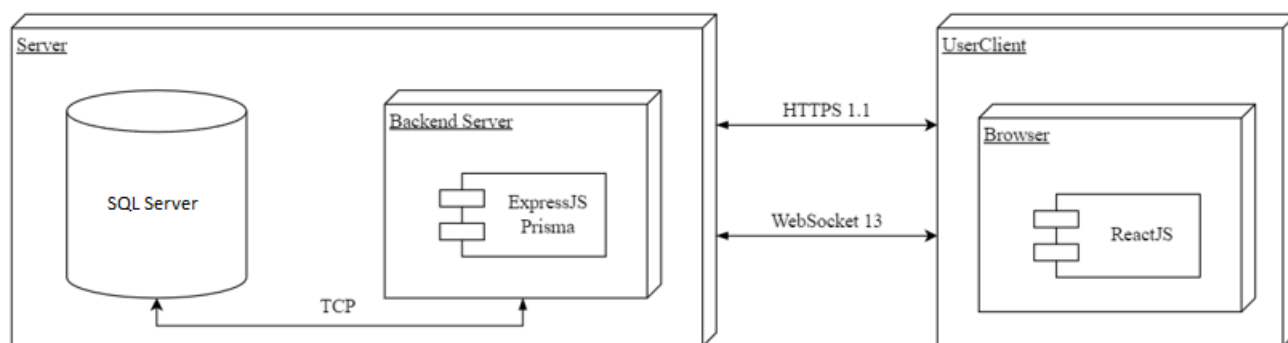


Рисунок 2.2 – Диаграмма развертывания

По данной диаграмме развертывания видно, что веб-приложение состоит из трех основных частей. Первая часть – это клиентская часть нашего приложения. Эта часть реализована при помощи одноименной библиотеки для языка программирования JavaScript React. Данная библиотека поможет создать динамически SPA приложение, которое будет делать запросы по протоколу HTTPS к серверной части приложения. Также оно будет отправлять и получать сообщения по протоколу WebSocket. Вторая часть данного веб-приложения – это серверная часть, разработанная на платформе NodeJS при помощи фреймворка ExpressJS, который позволяет построить структуру из всех основных модулей нашего приложения. Данная часть приложения будет обращаться к базе данных по протоколу TCP. Третья часть нашего приложения – это сам сервер базы данных SQL Server, который будет отвечать за хранение всех данных, способствующих работе нашего приложения.

## 2.4 Выводы по разделу

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

В данном разделе рассмотрели структуру базы данных SQL Server, ее основные сущности и как они связаны. Также рассмотрели, какие есть основные части нашего веб-приложения и как оно разворачивается.

## 3 Разработка web-приложения

### 3.1 Разработка клиентской части web-приложения

Клиентская часть – главная часть приложения. Она должна быть интуитивно понятна и интерактивна. Клиентская часть будет состоять из двух основных частей: библиотека React и WebSocket клиент.

Часть React состоит из тринадцати страниц. Страницы представляют с собой совокупность компонентов. Компоненты распределены по папкам. В проекте всего 8 папок: AdminPage, AnnouncementPage, AuthorizationPage, MainPage, OwnerMainPage, Premise, RentalPage, UserPage.

Когда пользователь заходит первый раз в веб-приложение ему доступна страница авторизации, на которой он может выбрать вход или регистрацию. В библиотеке React для маршрутизации используется компонент Router. Пример использования данной компоненты показан в листинге 3.1.

```

<Router>
  <Routes>
    <Route path="/login" element={<Authorization />}/>
    <Route path="/user" element={auth ? <User/> :
<Navigate to="/login" replace />}/>
    <Route path="/" element={auth ? (auth.role ===
0 ? <Main/> : auth.role === 1 ? <OwnerMain/> : <Admin/>) : <Navigate
to="/login" replace />}/>auth.role === 0 && (<><Route
path="/premise/:id" element={auth ? <Premise/> : <Navigate
to="/login" replace />}/>
    <Route path="/rental" element={auth ? <Rental/>
: <Navigate to="/login" replace />}/>
    <Route path="/announcement" element={auth ?
<Announcement/> : <Navigate to="/login" replace />}/>}}
    {auth.role === 1 && (<><Route
path="/application" element={auth ? <ApplicationList/> : <Navigate
to="/login" replace />}/>
    <Route path="/add_premise" element={auth ?
<AddPremise/> : <Navigate to="/login" replace />}/>
    <Route path="/announcement" element={auth ?
<Announcement/> : <Navigate to="/login" replace />}/>
    <Route path="/edit_premise/:id"
element={auth ? <EditPemise/> : <Navigate to="/login" replace />}/>
    </>}}{auth.role === 2 && (<><Route
path="/add_user" element={auth ? <AddUser/> : <Navigate to="/login"
replace />}/> Route path="/announcement" element={auth ?
<AnnouncementAdmin/> : <Navigate to="/login" replace />}/>
    <Route path="/edit_user/:id" element={auth ? <EditUser/> :
<Navigate to="/login" replace />}/></>}}
  </Routes>
</Router>

```

Листинг 3.1 – Настройки маршрутизации

Данный код реализован в компоненте App. По данному коду видно, что если пользователь авторизован, то он может переходить на основную страницу, на которой отобразятся элементы, в соответствии его роли, а также на те страницы, на которые есть разрешение у его роли. Если пользователь не авторизован, то ему будет доступна только страница авторизации, на которой он сможет войти или зарегистрироваться. Если он попытается на основную страницу зайти, его перебросит на страницу авторизации. Другие же страницы даже не создадутся, так у неавторизованного пользователя нет роли. При переходе на них, его перебросит на страницу, описанную в компоненте ErrorPage, которая отображает надпись 404. На эту страницу пользователя перебросит так же в том случае, если он попытается зайти по не описанному в данном компоненте URL.

Проверка, авторизован ли пользователь, выполняется следующим образом: сначала пользователь заходит на одну из страниц, затем вызовется хук, описанный в компоненте App, который отправляет на сервер запрос, с токенами в заголовке, сервер возвращает id пользователя и его роль. Эти данные сохраняются хук состояния auth. Если токен оказался недействительным, сервер отправляет ответ со статусом 401, в хуке этот статус обрабатывается и в состояние auth сохраняется пустой объект. Реализация данного хука представлена в листинге 3.2.

```
useEffect(() => {
    const isAuthenticated = async () => {
        try {
            const response = await
axios.get('https://localhost:3441/auth/resource', {
                headers: {'Authorization': `${accessToken}`};
            });
            console.log(response.data.role);
            setAuth(response.data);
        } catch (error) {if(error.response == undefined) return
setAuth({}) if(error.response.status === 401) {
                setAuth({});
            }
            console.error('Error fetching data:', error);
        } finally {
            setLoading(false);
        }
    };
    isAuthenticated();
}, []);
```

Листинг 3.2 – Хук для проверки авторизации

Если пользователь авторизован, то он может перейти на основную страницу, введя URL без дополнительных сегментов. В зависимости от роли, ему отобразятся разные компоненты. Если пользователь обычный, то ему отобразится страница с помещениями, выставленных на аренду. Основная часть данной страницы реализована в компоненте MainArea, код которого описан в листинге 3.3.

```

<div className="mainArea">
  <div className="filters">
    <div className="typesPremies">
      <button
onClick={setAllPremises}><p>Все</p></button>
      {typesPremies.map((typePremies) => (
        <button
onClick={changeTypePremise(typePremies.ID)}><p>{typePremies.Name}</p>
      </button>
      )))
    </div>
    <form className="mainFilters">
      <div className="filterElement">
        <p>Цена аренды:</p>
        <div className="inputFilter">
          <input type="number"
name="minPrice" placeholder="От" onChange={changeNumbersFilters}/>
          <input type="number"
name="maxPrice" placeholder="До" style={{marginLeft: '10px'}}
onChange={changeNumbersFilters}/>
        </div>
      </div>
      <div className="filterElement">
        <p>Площадь помещения:</p>
        <div className="inputFilter">
          <input type="number" name="minArea"
placeholder="От" onChange={changeNumbersFilters}/>
          <input type="number" name="maxArea"
placeholder="До" style={{marginLeft: '10px'}}
onChange={changeNumbersFilters}/>
        </div>
      </div>
      <div className="filterElement">
        <div
className="selectTypesPremiesElement">
          <input type="checkbox"
name="repair" checked={filters.repair} onChange={changeFilters}/>
          <label for="no">Ремонт</label>
        </div>
        <div
className="selectTypesPremiesElement">
          <input type="checkbox" name="pool"
checked={filters.pool} onChange={changeFilters}/>
          <label for="no">Бассейн</label>
        </div>
        <div className="selectTypesPremiesElement">
          <input type="checkbox"
name="billiards" checked={filters.billiards}
onChange={changeFilters}/>
          <label for="no">Бильярд</label>
        </div>
      </div>
    </form>
  </div>
</div>

```

```

                                <div
className="selectTypesPremiesElement">
                                <input type="checkbox"
name="furniture" checked={filters.furniture}
onChange={changeFilters}/>
                                <label for="no">Мебель</label>
                                </div>
                                </div>
                                </form>
                                </div>

                                <PromiseList premises={!filters.repair &&
!filters.pool && !filters.billiards && !filters.furniture &&
filters.maxArea === '' && filters.minArea === '' && filters.minPrice
=== '' && filters.maxPrice === '' ? premises : filteredPremises}/>
                                </div>
                                <div className="selectTypesPremiesElement">
                                <input type="checkbox"
name="furniture" checked={filters.furniture}
onChange={changeFilters}/>
                                <label for="no">Мебель</label>
                                </div>
                                </div>
                                </form>
                                </div>

                                <PromiseList premises={!filters.repair &&
!filters.pool && !filters.billiards && !filters.furniture &&
filters.maxArea === '' && filters.minArea === '' && filters.minPrice
=== '' && filters.maxPrice === '' ? premises : filteredPremises}/>
                                </div>

```

### Листинг 3.3 – Компонент MainArea

Если роль пользователя «владелец», то ему отобразится страница, основная часть которой реализована в компоненте OwnershipList. На данной странице отображается список помещений, которыми владеет пользователь. Помещения получают с сервера с через хук в компоненте, в состав которого входит этот компонент, и в этот компонент данные передаются через props. Код данного компонента описан в листинге 3.4.

```

<div className="userList">
    <a href='/add_premise'
className='addPremiseButton'><p>+</p></a>

    <div style={{marginTop: '20px'}}>
        {Array.isArray(premisesList) &&
premisesList.map((premise) => (
            <a href={`/edit_premise/${premise.ID}`}
className="rental" style={{marginTop: '20px'}}>

```

```

                {premise.Image !==
'data:image/png;base64,null' ? ( <img src={premise.Image} alt="" />)
: (<img src={noPhoto} alt="Placeholder" />)}
                <div className="name_price">
                    <div>{premise.Name}</div>
                    <div style={{marginTop:
'25px'}}>{premise.Price} BYN</div>
                </div>
                <button className='deleteButton'
onClick={ (e) => { e.preventDefault(); deletePremise(premise.ID);
}}><img src={deleteButton} /></button>
                </a>
            )}}
        </div>
    </div>

```

### Листинг 3.4 – Компонент OwnershipList

Если пользователь администратор, то ему отобразится страница со списком пользователей. На данной странице он сможет добавить нового пользователя, удалить пользователя, нажав на соответствующую кнопку на элементе пользователя, а также, нажав на элемент пользователя, перейти на страницу редактирования пользователя. Код данной странице реализован в компоненте UserList. Данный код отображён в листинге 3.5.

```

    <div className="userList" style={{paddingTop: '40px'}}>
        <a href='/add_user' className='addPremiseButton'
><p>+</p></a>

        <div style={{paddingTop: '30px'}}>
            {Array.isArray(usersList) &&
usersList.map((user) => (
                <a href={`/edit_user/${user.ID}`}
className="rental">
                    {user.Photo !==
'data:image/png;base64,null' ? ( <img src={user.Photo} alt=""
style={{width: '190px'}} />) : (<img src={noPhoto} alt="Placeholder"
style={{width: '190px'}} />)}
                    <div className="name_price">
                        <div>{user.Login}</div>
                    </div>
                    <button style={{marginLeft: '100px'}}
className='deleteButton' onClick={ (e) => { e.preventDefault();
deletePremise(user.ID); }}><img src={deleteButton} /></button>
                </a>
            )}}
        </div>
    </div>

```

### Листинг 3.4 – Компонент UserList

В данном React проекте присутствует много компонентов, описывающие разные страницы, необходимые для управления данными веб-приложения. Рассмотрим ещё компонент `UserInterface`, который является составляющей компонента `Header`. Компонент `Header`, используется во всех остальных страницах, и отображает логотип и часть пользовательского интерфейса, описанного в компоненте `UserInterface`. В компоненте `UserInterface` находятся кнопки и ссылки, в зависимости от роли пользователя, на страницы заявок, аренд, объявлений и управления своим аккаунтом. При нажатии на кнопку управления аккаунтом, появляется всплывающий элемент, в котором мы можем перейти на страницу редактирования своего аккаунта или выйти из аккаунта. При выходе из аккаунта, токены удаляются из локального хранилища. Код компонента `UserInterface` представлен в листинге 3.5.

```

    <div class="userInterface" style={{marginLeft: user.Role !== 2
? '848px' : '930px'}}>
      {user.Role === 0 && <a href="/rental"
className="userInterfaceComponent">
        <img src={myRent} alt=""/>
        <p>Аренды</p>
      </a>}

      {user.Role === 1 && <a href="/application"
className="userInterfaceComponent">
        <img src={myRent} alt=""/>
        <p>Заявки</p>
      </a> }

      <a href="/announcement"
className="userInterfaceComponent">
        <img src={message} alt=""/>
        <p>Объявления</p>
      </a>

      <div className="userInterfaceComponent"
onClick={handleUser} >
        {user.Photo !==
'data:image/png;base64,null' ? ( <img src={user.Photo} alt=""/>) :
(<img src={noPhoto} alt="Placeholder"/>)}
        <p>{user.Login}</p>
      </div>
      {isVisibleUserController && <div
className="selectAccountController">
        <a href='/user'><p>Настроить</p></a>
        <button onClick={handleLogout}>Выйти</button>
      </div>}
    </div>

```

Листинг 3.5 – Компонент `UserInterface`



Так же хочется отметить компонент `UserParams`, в котором пользователь может редактировать данные своего аккаунта. Код данного компонента представлен в листинге 3.6.

```

<div className="userParams">
  <div className='avaController'>
    {ava !== 'data:image/png;base64,null' ? (
<img src={ava} alt=""/>) : (<img src={noPhoto} alt="Placeholder"/>)}
    <div className='login'>{login}</div>
    <input type="file" accept="image/*"
onChange={changeAva}/>

  </div>

  <form className="userForm">
    <input placeholder='Имя' value={name}
onChange={(e) => setName(e.target.value)}/>
    <input placeholder='Фамилия' value={surname}
onChange={(e) => setSurname(e.target.value)}/>
    <input placeholder='Номер телефона'
value={phoneNumber} onChange={(e) =>
setPhoneNumber(e.target.value)}/>
  </form>

  <div className='underUserParams'>
    <p id="errorMessage" style={{marginTop:
'15px'}}>{errorText}</p>
    <button className='saveButton'
onClick={save}>Сохранить изменения</button>
  </div>
</div>

```

Листинг 3.6 – Компонент `UserInterface`

Для взаимодействия клиента и сервера по протоколу `HTTPS` использовался пакет `axios`, который позволяет удобно отправлять асинхронные запросы на сервер и обрабатывать ответы. Так же, для реализации объявлений, использовался `WebSocket`. Он был реализован в компонентах `AnnouncementAdmin` и `Announcement`. В хуках компонентов устанавливалось соединение с `WebSocket` на сервере и создавались обработчики событий `WebSocket`, в том числе и на обработку полученных сообщений. Полученные сообщения вставлялись в элемент страницы. В обработчике события щелчка мыши, выполнялась отправка сообщения через `WebSocket`. Реализация `WebSocket` на клиенте представлена в листинге 3.7.

```

const handleSubmit = async (e) => {
  e.preventDefault();

  ws.send(announcement);

};

```

```

useEffect(() => {
  const ws = new WebSocket('wss://localhost:3441');

  ws.addEventListener('open', function (event) {
    console.log('WebSocket соединение установлено');
    ws.send('Привет, сервер!');
  });

  ws.addEventListener('message', function (event) {
    console.log(event.data);
    setAnnouncements(prevAnnouncements =>
[...prevAnnouncements, event.data]);
  });

  ws.addEventListener('close', function (event) {
    console.log('WebSocket соединение закрыто');
  });

  ws.addEventListener('error', function (event) {
    console.error('Произошла ошибка:', event);
  });

  return () => {
    // При размонтировании компонента отключаемся от
WebSocket
    ws.close();
  };
},
[]);

```

Листинг 3.7 – Реализация WebSocket на клиенте

## 3.2 Разработка серверной части web-приложения

Серверная часть написана на фреймворке Express JS для платформы NodeJS. Данный фреймворк позволил разработать множество компонентов и собственную структуру сервера.

Для дальнейшего упрощения разработки был разработан собственный MVC. Он состоит из двух основных папок «Routers» и «Controllers», а также основного файла App.js.

Папка Controllers хранит в себе модули, которое экспортируют классы, представляющие из себя контролеры, которые предоставляют различные методы для обработки данных.

Папка Routers хранит в себе модули, которые вызывают методы контролеров в соответствии с URL, то есть определяют маршруты запросам. Листинги кода этих модуле представлены в Приложении А.

Компонент App.js инициализирует HTTPS и WebSocket сервер, так же здесь обрабатываются начальные пути URL. Код данного компонента представлен в листинге 3.8.

```

const express = require("express");
const AuthRouter = require('./Routers/AuthRouter')();
const PremiseRouter = require('./Routers/PremiseRouter')();
const TypePremiseRouter =
require('./Routers/TypePremiseRouter')();
const RentalRouter = require('./Routers/RentalRouter')();
const UserRouter = require('./Routers/UserRouter')();
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const cors = require('cors');
const https = require('https');
const fs = require('fs');
const WebSocket = require('ws');
const PORT = 3441;
const app = express();
app.use(cors());
const httpsServer = https.createServer(
  {
    key: fs.readFileSync('./ssl/key.pem', 'utf8'),
    cert: fs.readFileSync('./ssl/cert.pem', 'utf8')
  },
  app
);
const wss = new WebSocket.Server({ server: httpsServer });
wss.on('connection', function connection(ws) {
  console.log('Новое WebSocket-подключение');
  ws.on('message', function incoming(message) {
    wss.clients.forEach(function each(client) {
      if (client !== ws && client.readyState ===
WebSocket.OPEN) {
        client.send('' + message);
      }
    });
  });
  ws.on('close', function close() {
    console.log('WebSocket-подключение закрыто');
  });
});
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use('/auth', AuthRouter);
app.use('/premise', PremiseRouter);
app.use('/rental', RentalRouter);
app.use('/type_premise', TypePremiseRouter);
app.use('/user', UserRouter);
app.use((req, res) => res.status(404).send('404'));

httpsServer.listen(PORT, ()=>{console.log("Сервер прослушивает
запросы на порту: ", PORT)})

```

Листинг 3.8 – Компонент App

Для аутентификации пользователей был разработан специальный модуль TokenService. Данный модуль экспортирует функцию authenticateToken, которая извлекает из заголовка authorization accessToken и refreshToken, accessToken токен проверяет на корректность данных и извлекает из него аутентификационные данные пользователя, такие как id и роль. Если accessToken и refreshToken имеются и данные в accessToken корректны, то функция возвращает объект с аутентификационными данными пользователя, если же это всё не соблюдается, то функция возвращает false, после чего, в том месте, где была вызванная функция, клиенту возвращается ответ с статусом 401. Код данного модуля представлен в листинге 3.9.

```
const express = require("express");
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const cookieParser = require('cookie-parser');

const app = express();

const accessSecretKey = 'ILYHA_ACCES';
const refreshSecretKey = 'ILYHA_REFRESH';

app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());

module.exports = function authenticateToken(req, res) {
  try {
    const tokens =
req.headers['authorization'].split(';').map(token => token.trim());

    const accessToken = tokens[0].replace('Bearer ', '');
    const refreshToken = tokens[1];

    if (!accessToken && !refreshToken) return false;

    return jwt.verify(accessToken, accessSecretKey, (err,
user) => {
      if (err) {
        return false;
      }
      return user;
    })
  } catch {
    return false;
  }
}
```

Листинг 3.9 – Модуль TokenService

Для взаимодействия с базой данных использовалась ORM Prisma. ORM позволяет создать программную абстракцию над базой данных и очень удобно взаимодействовать с ней через специальные методы в программе. В листинге 3.10

рассмотрен пример получения данных из базы данных с помощью Prisma в функции `getMyPremise`. В этом же примере рассмотрен случай обработки изображений, полученных из базы данных. Мы проходим по полученным из базы данных объектам и заменяем бинарное значение свойств `Image` на `base64`.

```
getMyPremises = async (req, res) => {
  try {
    const userData = authenticateToken(req, res);
    if (!userData) return res.status(401).send();

    let premises = await prisma.premise.findMany({
      where: {
        ID_owner: parseInt(userData.id)
      },
    });

    premises = premises.map(premise => {
      const base64Image = premise.Image ?
premise.Image.toString('base64') : null;
      const imgSrc =
`data:image/png;base64,${base64Image}`;

      return {
        ...premise,
        Image: imgSrc
      };
    });

    return res.status(200).send(premises);
  } catch (error) {
    console.error('Ошибка при получении моих
помещений:', error);
    return res.status(500).send('Internal Server
Error');
  }
};
```

Листинг 3.10 – Пример получения данных из базы данных

### 3.3 Выводы по разделу

В данном разделе мы рассмотрели основные моменты разработки веб-приложения. Рассмотрели, как построено взаимодействие клиента с сервером по протоколу HTTPS и по протоколу WebSocket. Рассмотрели, как хранится информация, в каких БД она хранится и как отображается на клиенте. Рассмотрели, как построена безопасность данного веб-приложения.

## 4 Тестирование web-приложения

Вначале протестируем форму регистрации. Для этого в форме оставим поля пустыми, а затем нажмём кнопку регистрации. Ожидаем оповещение об незаполненности всех полей. Результат проделанного теста показан на рисунке 4.1.

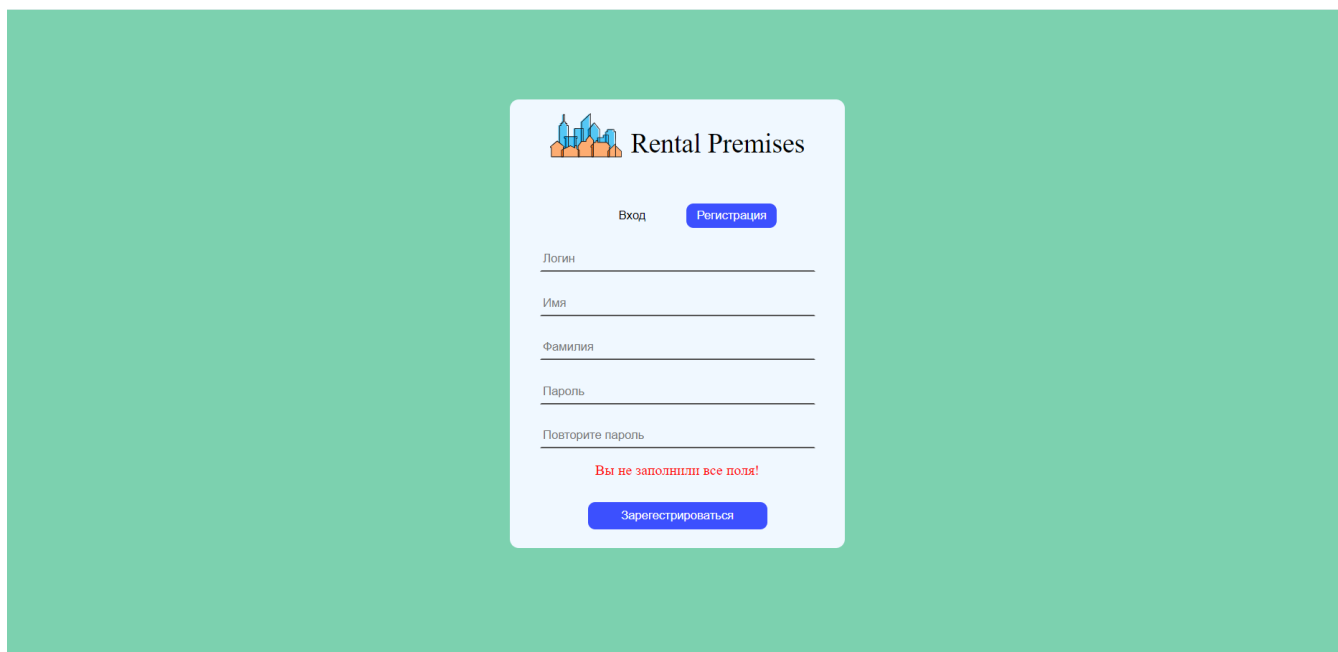


Рисунок 4.1 – Результат теста с оставлением незаполненной формы регистрации

По картинке 4.1 видно, что система отработала правильно и указала на ошибку.

Второй тест будет так же связан с формой регистрации, а именно с оповещением о неправильности подтверждения пароля при неправильном вводе повторного пароля. Ожидаем оповещение несовпадении паролей. Результат теста показан на рисунке 4.2.

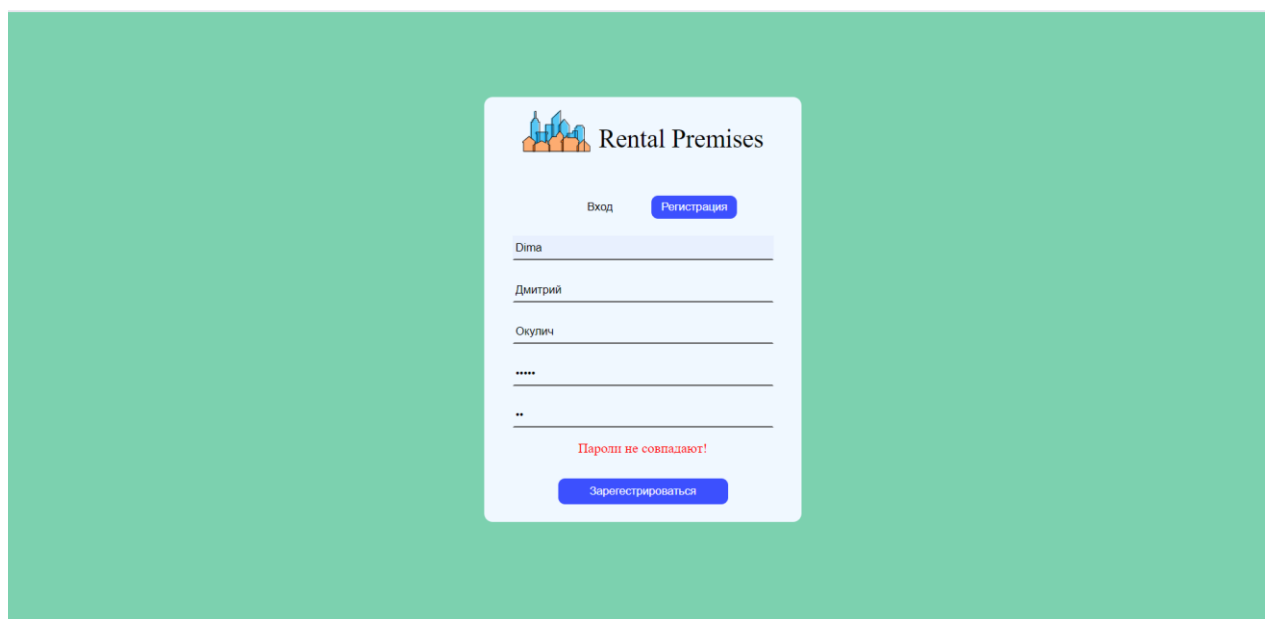


Рисунок 4.2 – Результат теста с неправильным вводом повторного пароля

По результатам второго теста, видно, что приложение указало на ошибку верно.

Третий тест будет проведён с целью проверки фильтрации помещений по цене аренды. Мы в поля фильтров введём предел цены от 200 до 500. Ожидаем то, что в списке помещений будут те помещения, цена которых соответствует данному диапазону. Результат прodelьвания теста показан на рисунке 4.3.

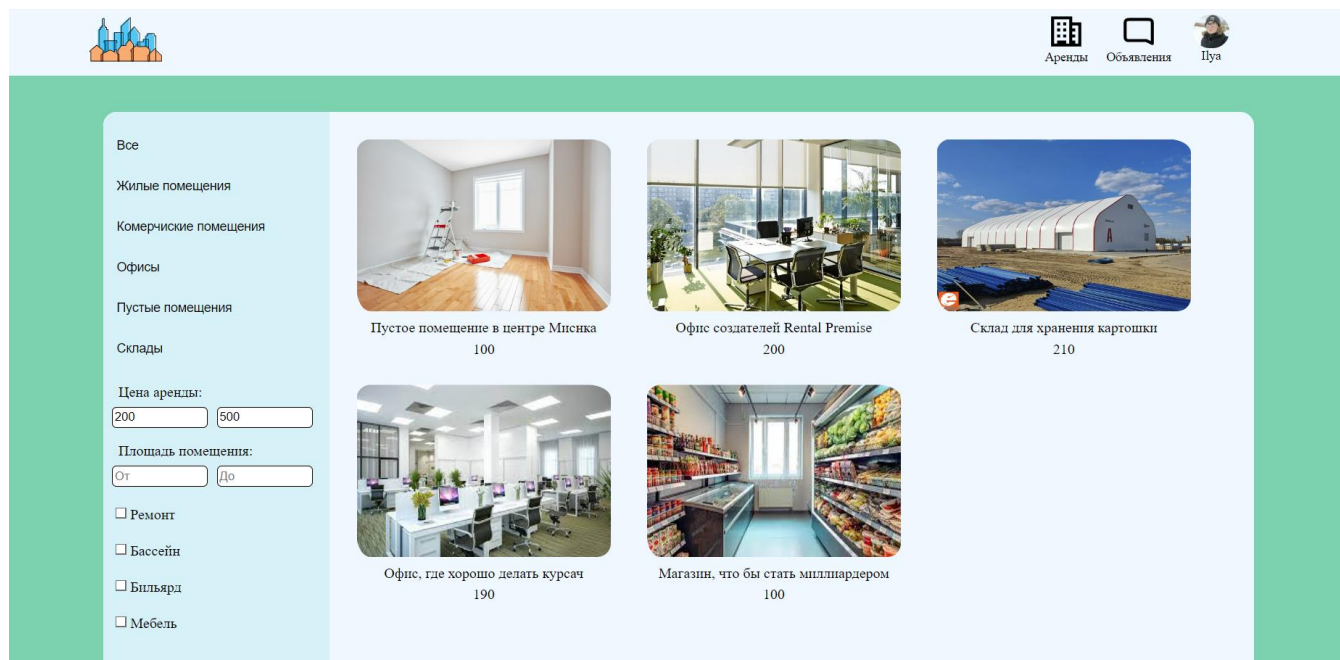


Рисунок 4.3 – Результат теста фильтрации помещений по цене

Как мы видим, в списке помещений только помещения, цены аренды которых от 200 до 500 рублей, что говорит о успешности теста.

Четвёртый тест заключается в проверке того, что бы в редактировании профиля пользователь мог вводит номер телефона только в соответствии с белорусским стандартом. При вводе неправильного номера, мы ожидаем соответствующее сообщение. Результат теста представлен на рисунке 4.4.

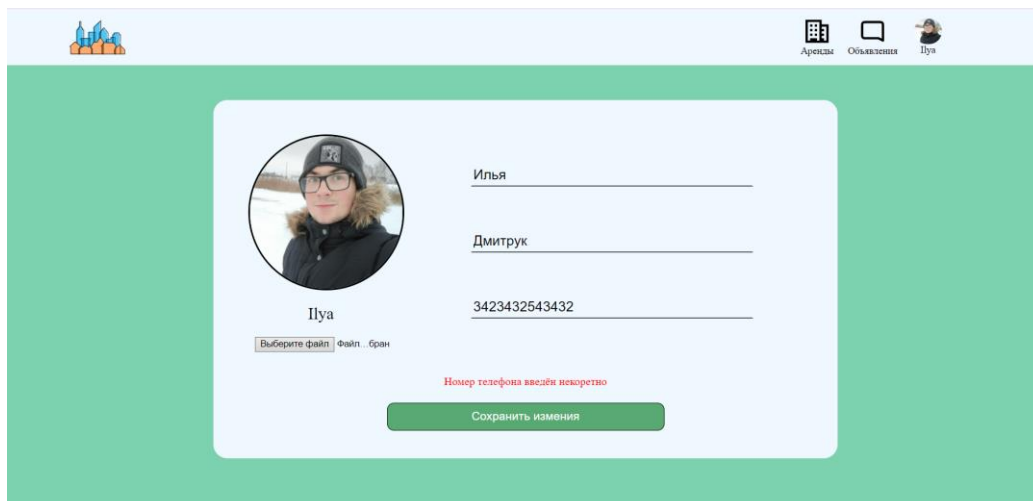
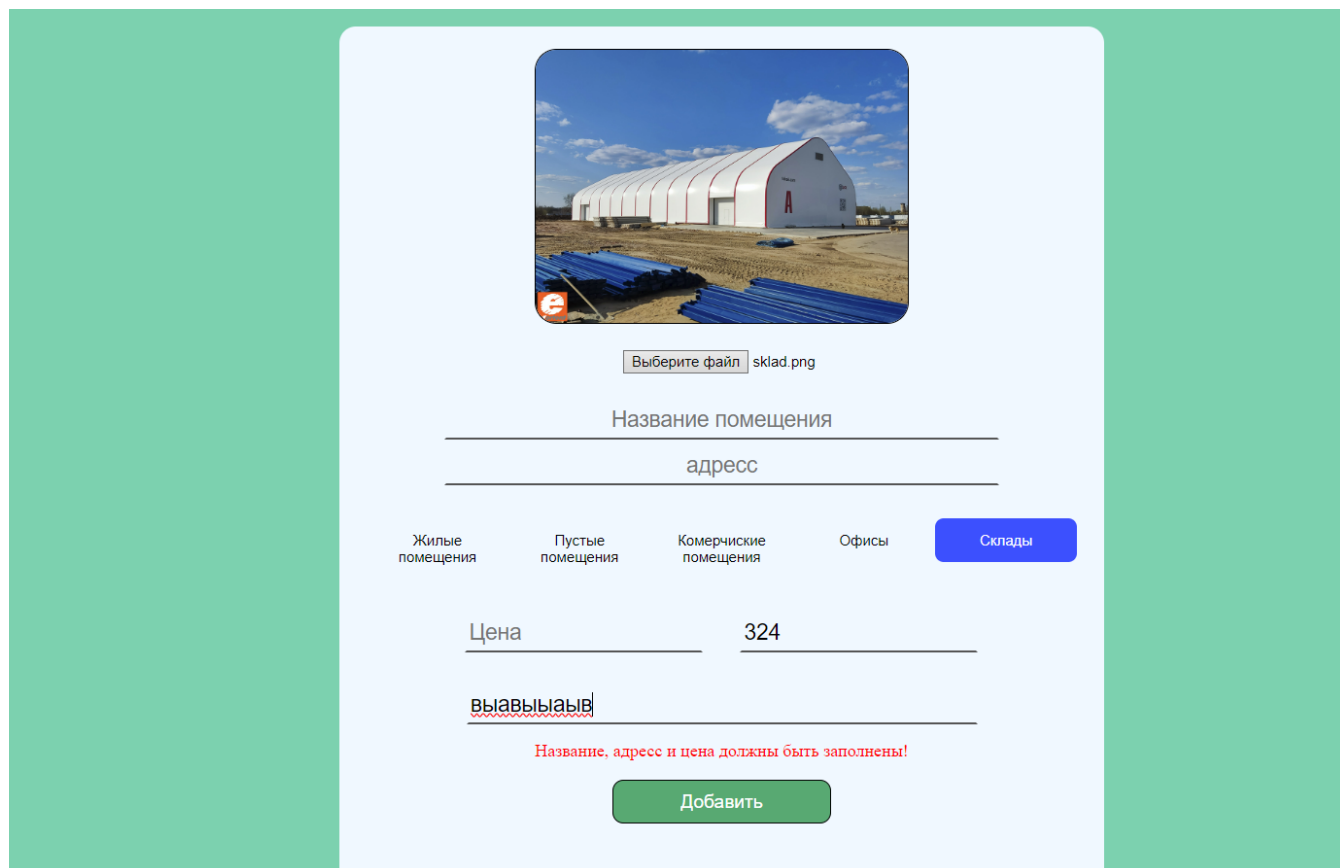


Рисунок 4.3 – Результат теста ввода неправильного номера телефона

Как мы видим, при вводе неправильного номера, вывелось соответствующее сообщение. Тест выполнен успешно.

В пятом тесте мы проверяем заполнение полей в форме добавления помещения. При добавлении помещения обязательными полями являются название помещения, адрес и цена. При оставлении этих полей пустыми, мы ожидаем вывод соответствующего сообщения. Результат представлен на рисунке 4.4.



Выберите файл sklad.png

Название помещения

адрес

Жилые помещения Пустые помещения Комерческие помещения Офисы Склады

Цена 324

вывывывыв

Название, адрес и цена должны быть заполнены!

Добавить

Рисунок 4.4 – Результат теста незаполнения обязательных полей в форме добавления помещения

Как видно, сообщение о пустых полях вывелось, соответственно тест выполнен успешно.

Шестой тест связан с защитой ресурсов от несанкционированного доступа. При попытке одним пользователем с ролью владельца изменить помещение другого владельца, у нас должен вернуться статус 403. Данный тест проводился в программе Postman. В запросе мы установили заголовок Authorization и в его значениях токены другого пользователя. После отправляем запрос, в теле указывая данные, которое надо заменить в помещении. Результат теста представлен на рисунке 4.5.



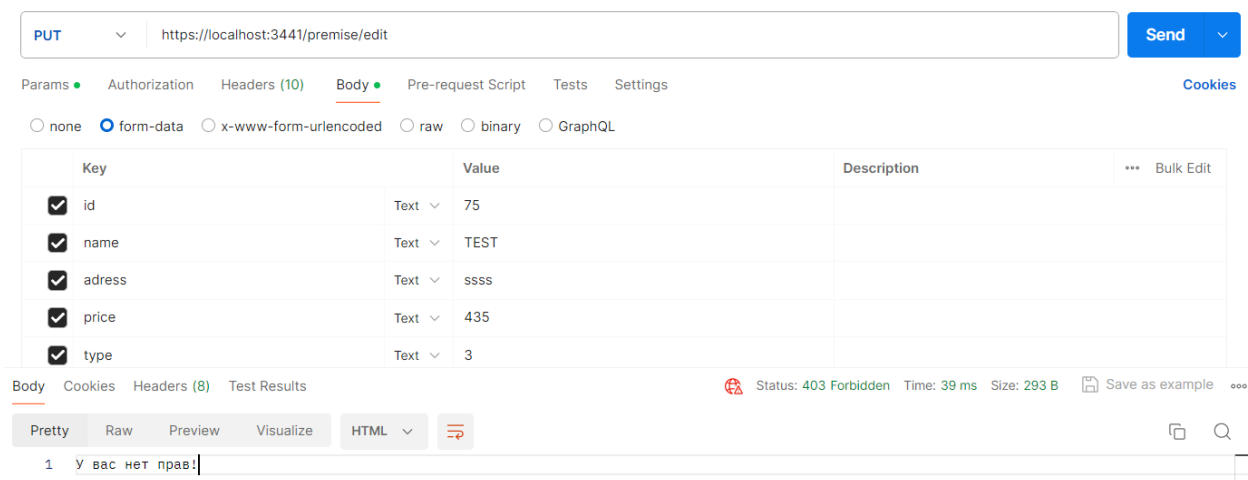


Рисунок 4.3 – Результат теста получения несанкционированного доступа к ресурсу

Как видно по рисунку, нам в ответе вернулся статус 403 и сообщение «У вас нет прав!», соответственно тест выполнен успешно.

В результате выполнения множества тестов, можно сказать, что приложение работает корректно и стабильно.

## 5 Руководство пользователя

### 5.1 Руководство пользователя

При открытии веб-приложения вас будет встречать форма авторизации, в которой вы можете ввести свой логин и пароль, а затем зайти под определенным пользователем, или же вы можете сверху над формой переключиться на форму регистрации, и, заполнив форму, зарегистрироваться и войти под созданным пользователем. Рисунок формы авторизации представлен на рисунке 5.1.

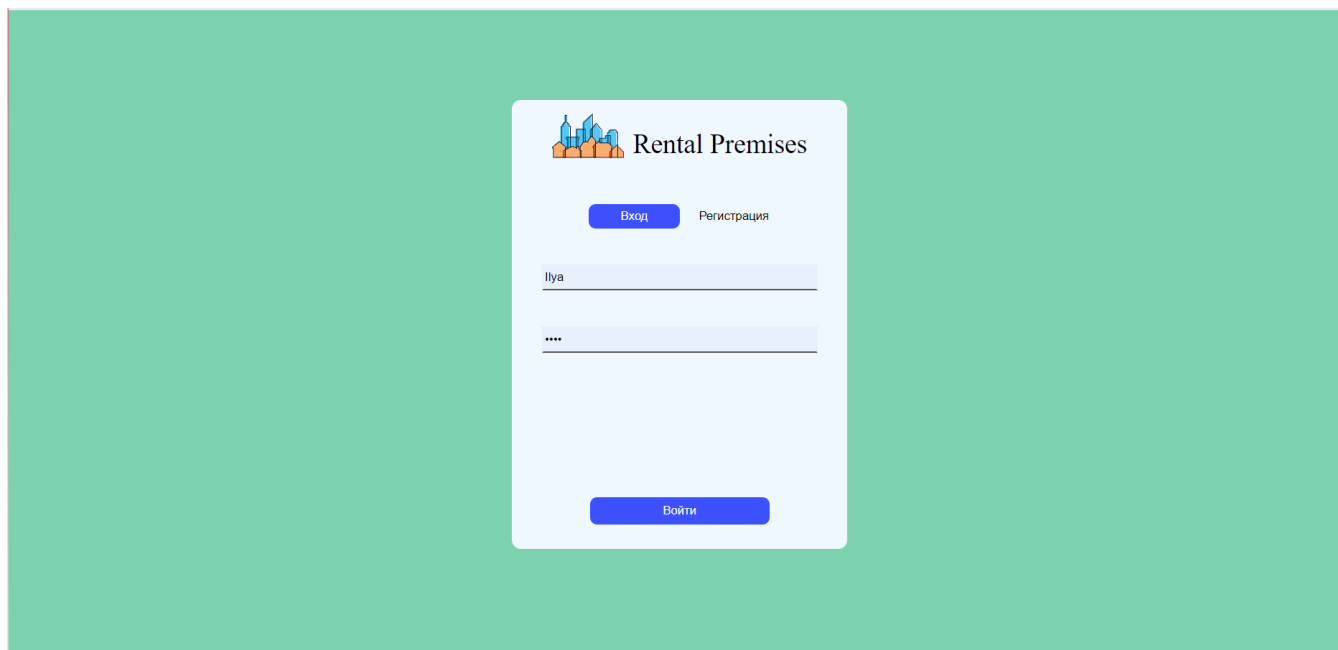


Рисунок 5.1 – Форма авторизации

После входа под пользователем с ролью арендатора, вам откроется главная страница приложения, где вы можете увидеть помещения, выставленные на аренду. Вы можете фильтровать помещения, левую боковую панель, выбирая интересующие вас типы и критерии, которые вам важны. Важно отметить, если вы выбрали, к примеру «бассейн», то это значит, что у вас отобразятся те помещения, в которых что-то указано в характеристике «бассейн». Главная страница представлена на рисунке 5.2.

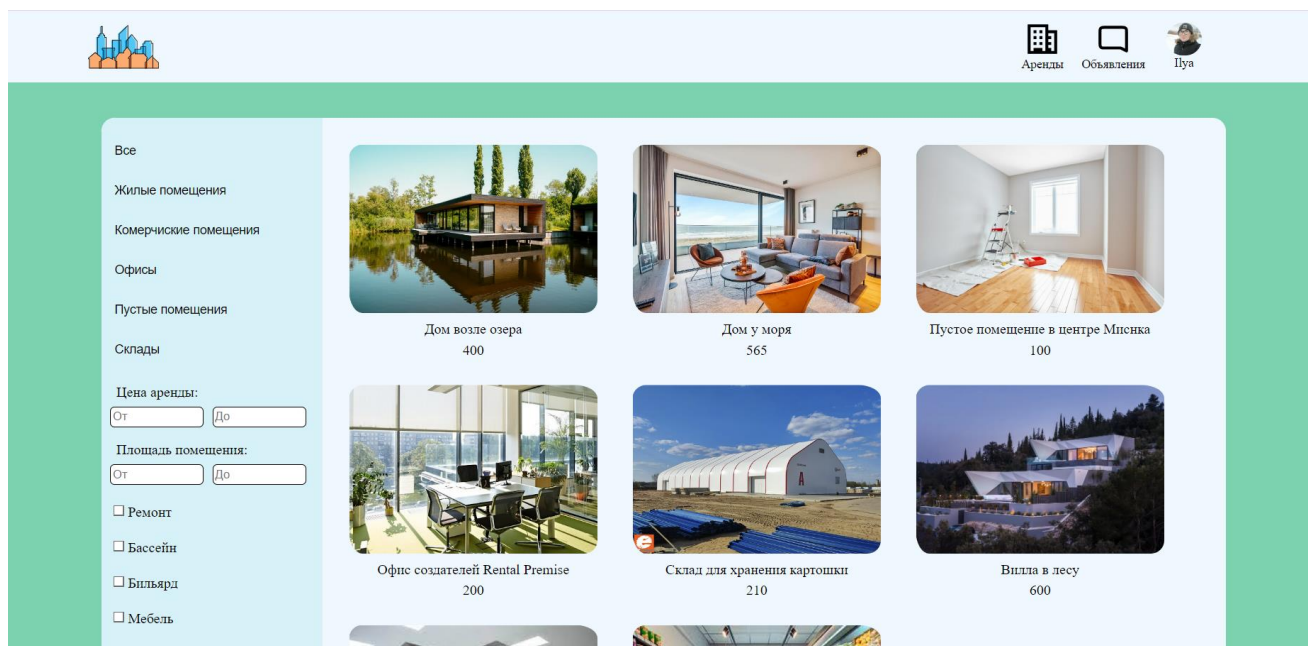


Рисунок 5.2 – Главная страница

Затем вам нужно выбрать помещение, которые вы хотите арендовать и нажать на него. Вам откроется страница с описанием данного помещения. Под изображением помещения располагается его название, ниже адрес, ещё ниже описание, которое установил владелец. Правее от описания находится цена и изображение владельца с его логином. Если нажать на изображение владельца, то вам откроется в всплывшем элементе краткая информация о нём в виде имени, фамилии и номера телефона, если он есть. Под ценой находится кнопка «Отправить заявку на аренду». Нажав на неё, она станет серой и сменится надписью на «Отменить заявку на аренду», а владельцу будет отправлена заявка, которую он должен принять. Если вы повторно нажмёте на ту кнопку, то она опять станет зелёной и заявка у владельца пропадёт. Данная страница представлена на рисунке 5.3.

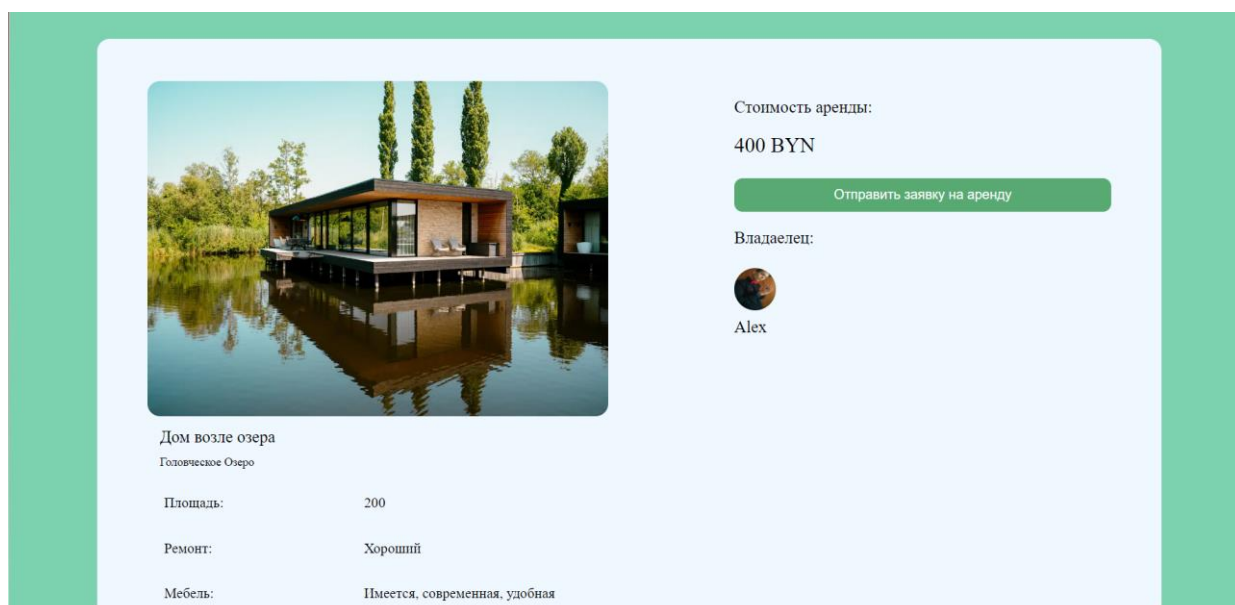


Рисунок 5.2 – Страница помещения

После того как владелец принял отправленную вами заявку, данное помещение больше не будет отображаться в списке неаредованных помещений. Вы же можете посмотреть свои аренды, нажав на кнопку «Аренды» в правом верхнем углу. После нажатия на эту кнопку, вас перекинет на страницу с вашими арендами. На данной странице, помещения будут располагаться вертикально. Здесь так же будет отображена основная информация о помещении, в виде изображения, информации и цены, а так же справа даты, когда началась аренда. Нажав на помещение, вы так же перейдёте на страницу с его описанием, на которой вы сможете отменить аренду. Страница аренд представлена на рисунке 5.3.

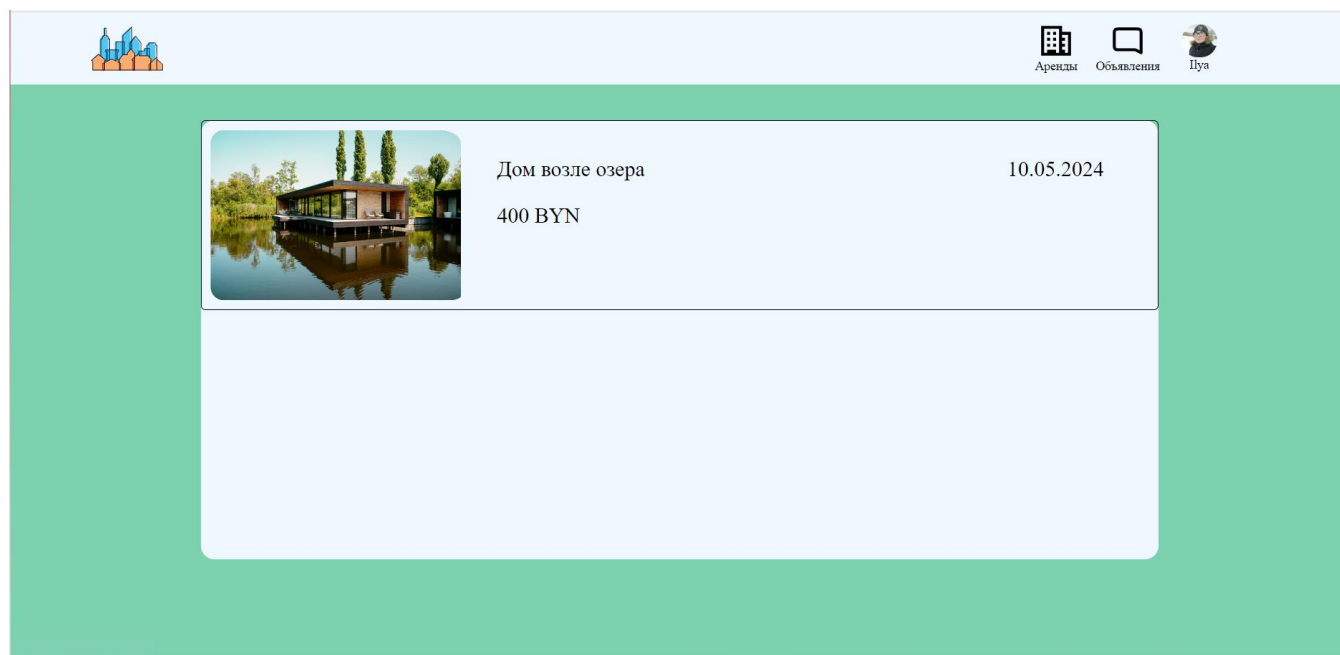


Рисунок 5.3 – Страница аренд

Теперь рассмотрим владельца. При входе с аккаунта, у которого роль владелец, вас отправит на главную страницу для владельца. На ней будет два списка помещений: один отображает помещения владельца, второй помещения, которые арендуют. Что бы переключаться между этими списками, необходимо нажимать на кнопки над списками. Если вы выбрали список «Мои помещения», то сверху на список будет так же кнопка в виде плюса. Нажав на неё, вас перекинет на страницу добавления помещений. Помещения в списке располагаются вертикально. В списке отображена основная информация о помещении, в виде изображения, информации и цены. Справа будет круглая кнопка с изображением урны. Нажав на неё, вы удалите помещение. Если вы нажмёте на само помещение, то вас перекинет на страницу редактирования данного помещения. Главная страница владельца со списком помещений представлена на рисунке 5.4.

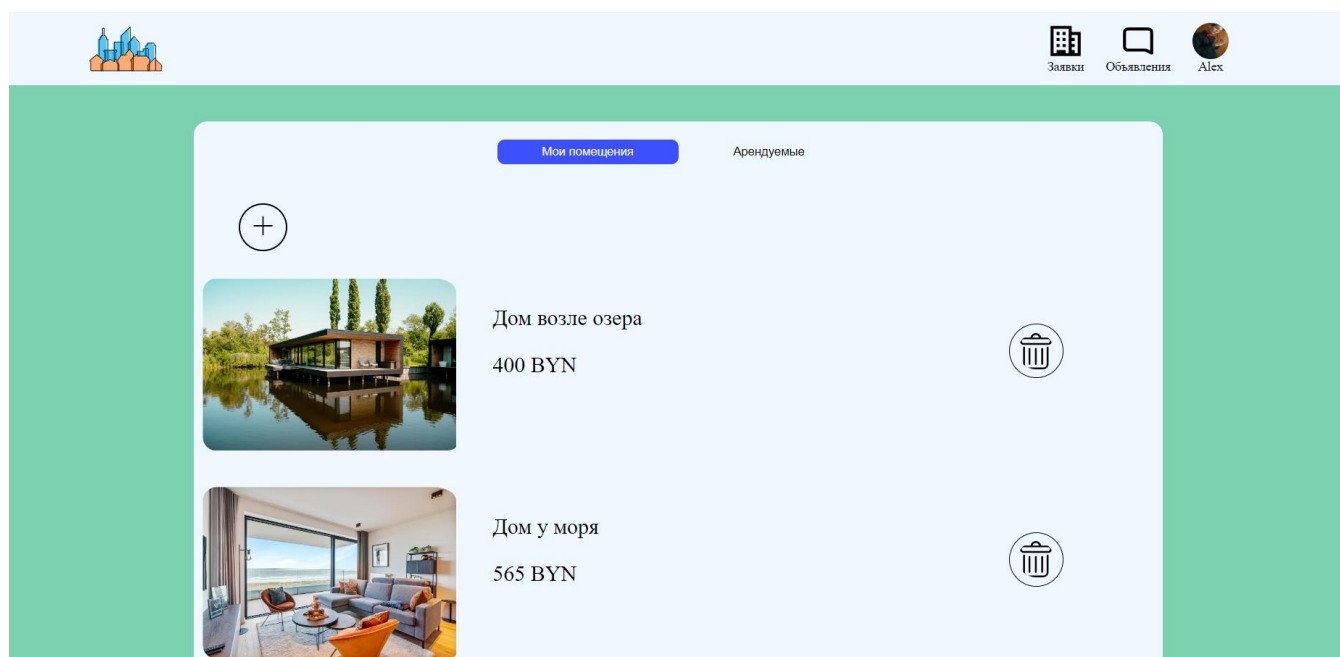


Рисунок 5.4 – Главная страница владельца

Выбрав список «Арендуемые», у вас отобразятся только те ваши помещения, которые кто-то уже арендует. Данный список похож на основной список помещений владельца, только справа отображается дата начала аренды и под неё кнопка с крестиком для отмены аренды. Если вы отмените аренду, у обычного пользователя данное помещение пропадёт из списка аренд и помещение отобразится в основном списке неарендуемых помещений на главной странице обычного пользователя. Список «Арендуемые» представлен на рисунке 5.5.

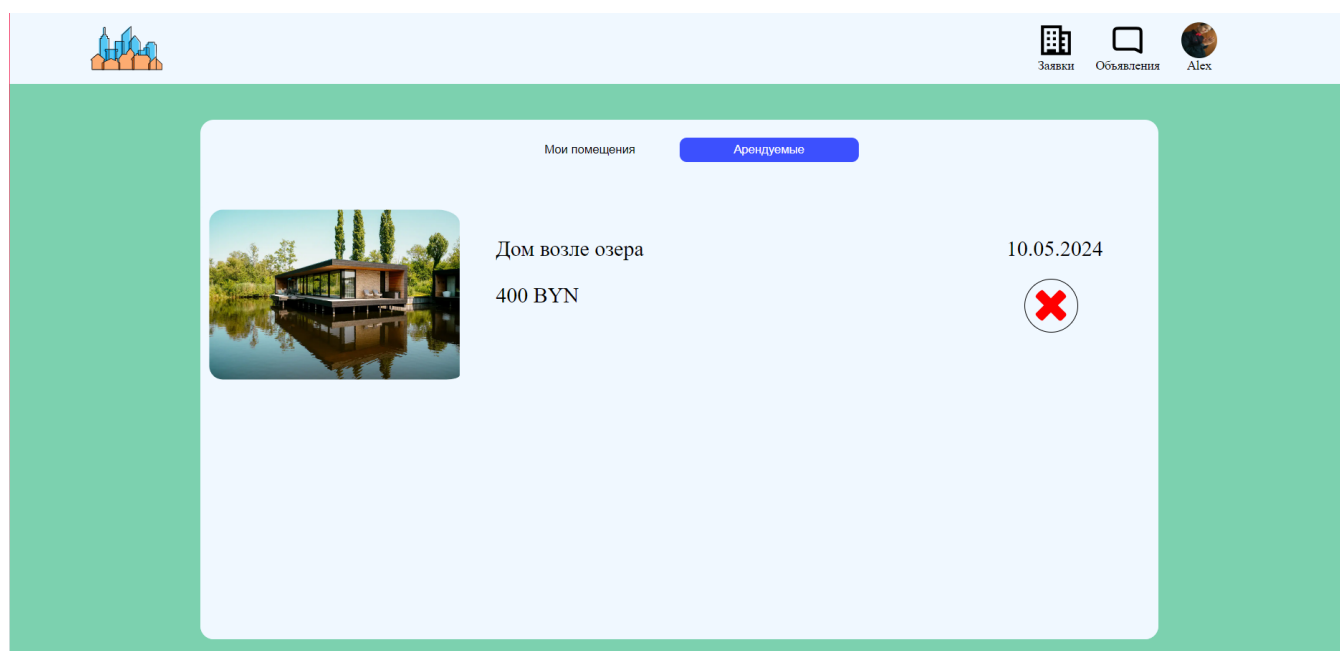


Рисунок 5.5 – Список «Арендуемые»

Теперь рассмотрим, как происходит принятие заявки на аренду. Для этого нам надо нажать на кнопку «Заявки» в правом верхнем углу страниц владельца, после чего у нас откроется страница со списком заявок. В списке информация о помещениях, на аренду которых были поданы заявки, располагается вертикально. Кроме основной информации о помещении, так же отображается информация о пользователе, который отправил заявку, в виде его аватарки и логина. Справа находятся две кнопки, одна в виде галочки, вторая в виде крестика. Та что в виде галочки, принимает заявку, та что в виде крестика, отменяет. При нажатии на принятие заявки, все остальные заявки на аренду этого помещения удаляются. Данная страница представлена на рисунке 5.6.

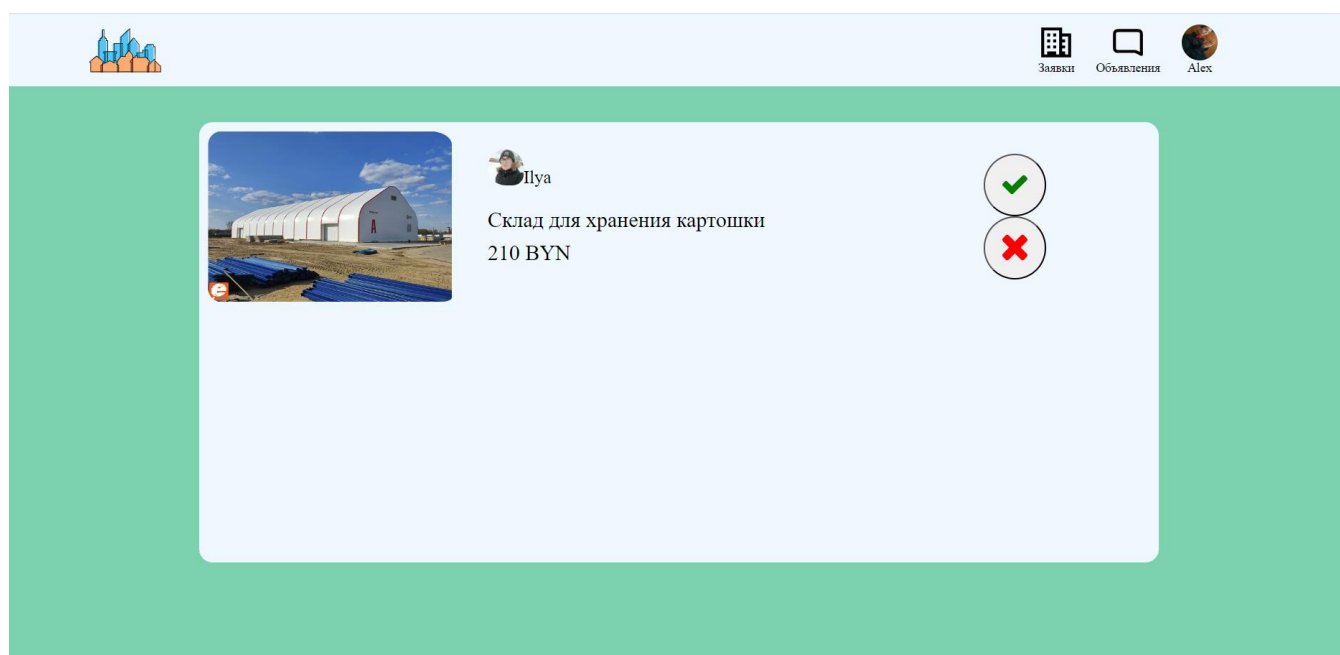


Рисунок 5.6 – Страница заявок

Теперь рассмотрим такой функционал, как добавление помещений. При нажатии на плюсик на главной странице владельца, вас перекинет на страницу добавления помещения. Данная страница представляет с собой форму для заполнения данных о помещении. В начале вам надо выбрать изображение помещения, или оставить без изображения. Далее ввести название и адрес. Эти поля являются обязательными. После необходимо выбрать тип помещения. В зависимости от типа, у вас будут разные характеристики. После вы должны заполнить описание характеристик. Однако, это не обязательно. Если вы ничего в какую-нибудь характеристику ничего не впишете, то тогда, когда пользователь будет выбираться в фильтрах данную характеристику, то ваше помещение просто не отобразится. Однако среди характеристик обязательным является ещё цена. При нажатии на помещение на главной странице владельца, вас перекинет на страницу редактирования помещения, которая похожа на эту страницу добавления. Внизу страницы находится кнопка «Добавить», нажав на которую, вас переместит на основную страницу, а помещение будет добавлено. Страница добавления помещения представлена на рисунке 5.7.

Рисунок 5.7 – Страница добавления

Осталось рассмотреть страницу редактирования аккаунта. Во всех видах пользователя справа сверху находится фотография пользователя и его логин. При нажатии на фотографию, вам в всплывающем элементе предложить перейти к настройкам профиля. Перейдя на страницу настроек профиля, вы можете установить себе аватарку, сменить имя, фамилию и номер телефона. Имя и фамилия являются обязательными полями, а номер телефона может и не быть, однако важно помнить, что номер должен соответствовать белорусскому стандарту. После того как вы изменили всё, что хотели, можете нажимать на кнопку «Сохранить изменения». Страница редактирования профиля представлена на рисунке 5.8.

Рисунок 5.7 – Страница редактирования профиля

Мы рассмотрели основной функционал данного приложения. Он должен помочь освоиться новому пользователю и с легкостью влиться в процесс работы с приложением. Мы увидели, что интерфейс понятный и отзывчивый, пользователь сможет легко ориентироваться и у него не должно возникнуть никаких проблем.

## **5.2 Установка приложения**

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить сервер базы данных SQL Server.
1. Запустить серверную часть приложения, которая соединяет базу данных и React приложение. Для этого необходимо запустить скрипт, который настроит соединение с базой данных и запустит сервер.
2. Запустить React приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером. Для этого необходимо запустить команду для сборки и запуска React приложения.

После выполнения этих шагов приложение будет полностью готово к работе, и пользователь сможет начать использовать его функционал.

## **5.3 Выводы по разделу**

В данном разделе мы рассмотрели, как пользоваться приложением «RentalPremise». Рассмотрели, какими функциями обладают пользователи. Рассмотрели авторизацию, регистрацию и все основные страницы пользователей.



## Заключение

Приложение «RentalPremise» для аренд помещений – полностью функциональное приложения для размещения помещений и их аренд. Оно позволяет пользователями удобно предоставлять друг другу услуги аренд.

За серверную часть отвечала платформа NodeJS, а именно фреймворк ExpressJS, который позволил реализовать быструю асинхронную серверную часть. Структура серверной части стала выглядеть как дерево, что позволило легко соединять модули, делать их независимыми от других модулей. Также ExpressJS с помощью ORM Prisma позволил создать абстракцию над данными базы данных.

Клиентская часть была написана при помощи библиотеки ReactJS для языка JavaScript. React позволил легко построить структуру клиентской части веб-приложения.

База данных SQL Server позволила создать структуру, которая позволила эффективно хранить взаимосвязанные данные. База данных содержит функционал для работы с ней.

В итоге были реализованы главные функции веб-приложения, такие как:

- обеспечивать возможность регистрации и авторизации;
- поддерживать роли администратора, владельца, арендатора;
- позволять выставлять на аренду помещения;
- позволять просматривать выставленные на аренду помещения;
- позволять отправлять заявки на аренду помещений;
- позволять принимать заявки на аренду помещений;
- позволять просматривать арендуемые помещения;
- давать возможность фильтровать выставленные на аренду помещения.

Тестирование проекта показало, что приложение функционирует правильно и все поставленные тесты прошло. В ходе тестирования ошибок не было найдено.

Данное приложение готово к эксплуатации и использованию пользователями.

### Список используемых источников

1. [react.dev/learn](https://react.dev/learn) документация к React [Электронный ресурс] / Режим доступа: <https://react.dev/learn> – Дата доступа: 25.03.2023.
2. [expressjs.com/ru](https://expressjs.com/ru/) документация к Express.JS [Электронный ресурс] / Режим доступа: <https://expressjs.com/ru/> – Дата доступа: 30.03.2023.
3. [Stackoverflow.com](https://stackoverflow.com) [Электронный ресурс] / Режим доступа: <https://stackoverflow.com> – Дата доступа: 10.04.2023.
4. [prisma.io](https://www.prisma.io/docs/getting-started/quickstart) документация к Prisma [Электронный ресурс] / Режим доступа: <https://www.prisma.io/docs/getting-started/quickstart> /– Дата доступа: 29.03.2023.

## Приложение А

```

    const express = require("express");
    const AuthController =
require('../Controllers/AuthController.js');

    module.exports = () => {
        const controller = new AuthController();
        const router = express.Router();

        router.get('/resource', (req, res) =>
controller.getResource(req, res));
        router.get('/login', (req, res) => controller.getLogin(req,
res));
        router.get('/logout', (req, res) =>
controller.executeLogout(req, res));
        router.post('/login', (req, res, next) =>
{console.log(req.body); controller.executeLogin(req, res, next) });
        router.post('/register', (req, res, next) =>
{console.log(req.body); controller.executeRegister(req, res) });
        router.use((req, res) => res.status(404).send('404'));

        return router;
    }

```

### Листинг 1 – модуль AuthRouter

```

    const express = require("express");
    const PremiseController =
require('../Controllers/PremiseController.js');
    const multer = require('multer');
    const path = require('path');

    module.exports = () => {
        const controller = new PremiseController();
        const router = express.Router();

        const storage = multer.diskStorage({
            destination: function (req, file, cb) {
                cb(null, 'uploads/');
            },
            filename: function (req, file, cb) {
                const fileName = 'premise' +
path.extname(file.originalname);
                cb(null, fileName);
            }
        });

        const upload = multer({ storage: storage });

        router.get('/all', (req, res) =>
controller.getPremises(req, res));
    }

```

```

        router.get('/my', (req, res) =>
controller.getMyPremises(req, res));
        router.get('/my_rented', (req, res) =>
controller.getMyRentedPremises(req, res));
        router.get('/element', (req, res) =>
controller.getPremise(req, res));
        router.post('/add', upload.single('photo'), (req, res) =>
controller.addPremise(req, res));
        router.put('/edit', upload.single('photo'), (req, res) =>
controller.editPremise(req, res));
        router.delete('/remove', (req, res) =>
controller.deletePremise(req, res));
        router.use((req, res) => res.status(404).send('404'));

    return router;
}

```

## Листинг 2 – модуль PremiseRouter

```

const express = require("express");
const RentalController =
require('../Controllers/RentalController.js');

module.exports = () => {
    const controller = new RentalController();
    const router = express.Router();

    router.get('/all', (req, res) => controller.getRentals(req,
res));
    router.get('/my', (req, res) =>
controller.getMyRentals(req, res));
    router.get('/applications', (req, res) =>
controller.getApplications(req, res));
    router.post('/add', (req, res) => controller.addRental(req,
res));
    router.get('/status', (req, res) =>
controller.getStatus(req, res));
    router.delete('/remove', (req, res) =>
controller.deleteRental(req, res));
    router.delete('/remove_by_id', (req, res) =>
controller.deleteRentalById(req, res));
    router.put('/accept', (req, res) =>
controller.acceptApplication(req, res));
    router.use((req, res) => res.status(404).send('404'));

    return router;
}

```

## Листинг 3 – модуль RentalRouter

```

    const express = require("express");
    const TypePremise =
require('../Controllers/TypePremiseController.js');

    module.exports = () => {
        const controller = new TypePremise();
        const router = express.Router();

        router.get('/all', (req, res) =>
controller.getTypesPremise(req, res));
        router.get('/type', (req, res) =>
controller.getTypePremise(req, res));
        router.use((req, res) => res.status(404).send('404'));

        return router;
    }

```

Листинг 4— модуль TypePremiseRouter

```

    const express = require("express");
    const UserController =
require('../Controllers/UserController.js');
    const multer = require('multer');
    const path = require('path');
    module.exports = () => {
        const controller = new UserController();
        const router = express.Router();
        const storage = multer.diskStorage({
            destination: function (req, file, cb) {
                cb(null, 'uploads/');
            },
            filename: function (req, file, cb) {
                const fileName = 'ava' +
path.extname(file.originalname);
                cb(null, fileName);
            }
        }); const upload = multer({ storage: storage });
        router.put('/edit', upload.single('photo'), (req, res) =>
controller.editUser(req, res));
        router.put('/edit_by_id', upload.single('photo'), (req,
res) => controller.editUserById(req, res));
        router.post('/add', upload.single('photo'), (req, res) =>
controller.addUser(req, res));
        router.get('/photo', (req, res) =>
controller.getUserPhoto(req, res));
        router.get('/person', (req, res) => controller.getUser(req,
res)); router.get('/my', (req, res) => controller.getMyUser(req,
res)); router.get('/all', (req, res) => controller.getAllUser(req,
res)); router.delete('/remove', (req, res) =>
controller.deleteUser(req, res));
        router.use((req, res) => res.status(404).send('404'));
        return router;
    }

```

Листинг 4— модуль UserRouter