**Министерство образования и науки Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИТМО”**

Факультет Инфокоммуникационных технологий

Образовательная программа Интеллектуальные системы в гуманитарной сфере (Академический бакалавр, Очная ф/о)

Направление подготовки (специальность) 45.03.04 - Интеллектуальные системы в гуманитарной сфере

О Т Ч Е Т

по курсовой работе по дисциплине «Основы web-программирования»

Тема задания: Разработка web-приложения для администратора гостиницы

Обучающийся: Ролинский Сергей Олегович, К3343

Руководитель курсовой работы: Говоров Антон Игоревич, ассистент

Оценка по курсовой работе \_\_\_\_

Подписи членов комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Говоров А. И.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чунаев А. В.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Антонов М. Б.

(подпись)

Дата \_\_\_\_

Санкт-Петербург

20 20

#### СОДЕРЖАНИЕ

Оглавление

[ВВЕДЕНИЕ 4](#_Toc45305478)

[1. Предметная область 5](#_Toc45305479)

[1.1. Общее описание предметной области 5](#_Toc45305480)

[1.2. Функциональные требования к web-сервису в рамках предметной области 5](#_Toc45305481)

[1.3. Выводы 6](#_Toc45305482)

[2. Модели данных 7](#_Toc45305483)

[2.1. Теория 7](#_Toc45305484)

[2.2. Разработка моделей данных для создаваемого web-приложения 7](#_Toc45305485)

[2.2.1. Модель клиента 8](#_Toc45305486)

[2.2.2. Модель служащего 8](#_Toc45305487)

[2.2.3. Модель комнаты 9](#_Toc45305488)

[2.2.4. Модель заселения 9](#_Toc45305489)

[2.2.5. Модель расписания работы 10](#_Toc45305490)

[2.3. Выводы 11](#_Toc45305491)

[3. Серверная часть 12](#_Toc45305492)

[3.1. Теория 12](#_Toc45305493)

[3.2. Админ-панель 12](#_Toc45305494)

[3.2.1. Теория 12](#_Toc45305495)

[3.2.2. Код 13](#_Toc45305496)

[3.2.3. Внешний вид 13](#_Toc45305497)

[3.3. Django REST Framework 15](#_Toc45305498)

[3.3.1. Общий принцип работы 15](#_Toc45305499)

[3.3.2. URL-пути серверной части web-приложения 15](#_Toc45305500)

[3.3.3. Сериализаторы 16](#_Toc45305501)

[3.3.4. Представления 17](#_Toc45305502)

[3.4. Выводы 26](#_Toc45305503)

[4. Клиентская часть 28](#_Toc45305504)

[4.1. Теория 28](#_Toc45305505)

[4.2. Компоненты 29](#_Toc45305506)

[4.2.1. Файл index.js 29](#_Toc45305507)

[4.3. Компонент Home 30](#_Toc45305508)

[4.3.1. Код 30](#_Toc45305509)

[4.3.2. Отображение на сайте 32](#_Toc45305510)

[4.4. Компонент Login 32](#_Toc45305511)

[4.4.1. Код 32](#_Toc45305512)

[4.4.2. Отображение на сайте 34](#_Toc45305513)

[4.5. Компонент Main 35](#_Toc45305514)

[4.5.1. Код 35](#_Toc45305515)

[4.5.2. Отображение на сайте 37](#_Toc45305516)

[4.6. Компонент 38](#_Toc45305517)

[4.6.1. Код 38](#_Toc45305518)

[4.6.2. Отображение на сайте 39](#_Toc45305519)

[4.7. Компонент Rooms 40](#_Toc45305520)

[4.7.1. Код 40](#_Toc45305521)

[4.7.2. Отображение на сайте 41](#_Toc45305522)

[4.8. Компоненты Worker, Worktable и Journal 42](#_Toc45305523)

[4.9. Компоненты Request1…5 и Report 46](#_Toc45305524)

[4.10. Muse-UI 50](#_Toc45305525)

[4.11. Выводы 51](#_Toc45305526)

[5. ЗАКЛЮЧЕНИЕ 52](#_Toc45305527)

[СПИСОК ЛИТЕРАТУРЫ 53](#_Toc45305528)

[Приложение А. Сериализаторы 54](#_Toc45305529)

[Приложение В. Страницы Django REST Framework’а 55](#_Toc45305530)

[Приложение Г. Код компонента Main 56](#_Toc45305531)

[Приложение Д. Код компонента Clients 60](#_Toc45305532)

[Приложение Е. Код компонента Rooms 63](#_Toc45305533)

[Приложение Ж. Код компонента Worktable 65](#_Toc45305534)

[Приложение З. Код компонента Journal 69](#_Toc45305535)

[Приложение И. Код компонента Request1 77](#_Toc45305536)

[Приложение К. Код компонента Report 80](#_Toc45305537)

[Приложение Л. Код компонента Request2 85](#_Toc45305538)

[Приложение М. Код компонента Request3 87](#_Toc45305539)

[Приложение Н. Код компонента Request4 91](#_Toc45305540)

[Приложение О. Код компонента Request5 93](#_Toc45305541)

# ВВЕДЕНИЕ

Курсовая работа заключается в разработке web-приложения, предназначенного для администратора некой гостиницы.

Цель выполнения данной курсовой работы в рамках прохождения обучения по дисциплине «Основы web-программирования» заключается в овладении навыками и умениями реализации web-сервисов с использованием таких программ библиотек и компонентов, как Django, Django REST Framework, Vue.js, Muse-UI…

Основные задачи курсовой работы:

1. Анализ предметной области;
2. Разработка функциональных требований к сайту в рамках данной предметной области;
3. Создание модели данных;
4. Реализация серверной части web-приложения средствами Django REST Framework;
5. Реализация клиентской части web-приложения средствами Vue.js и организация связи между серверной и клиентской частями.

В первой главе проведён анализ содержания работ и требований к их выполнению. Во второй главе рассмотрена разработка модели данных. В третьей главе разобрано создание серверной части реализуемого web-сервиса. В четвёртой главе представлена разработка клиентской части сервиса. В приложении А приведены сериализаторы, в приложении Б – представления, в приложении В – отображение страниц Django REST Framework’а, в приложениях Г, Д, Е, Ж и З – код компонентов Vue.

# Предметная область

## Общее описание предметной области

Общее описание предметной области было взято из текста индивидуального задания к курсовой работе.

Задание заключалось в том, чтобы создать программную систему, предназначенную для администратора гостиницы. Такая система должна обеспечивать хранение сведений об имеющихся в гостинице номерах, о проживающих в гостинице клиентах и о служащих, убирающихся в номерах.

Количество номеров в гостинице известно, и имеются номера трех типов: одноместный, двухместный и трехместный, отличающиеся стоимостью проживания в сутки. В каждом номере есть телефон.

О каждом проживающем должна храниться следующая информация: номер паспорта, фамилия, имя, отчество, город, из которого он прибыл, дата поселения в гостинице, выделенный гостиничный номер.

О служащих гостиницы должна храниться информация следующего содержания: фамилия, имя, отчество, где (этаж) и когда (день недели) он убирает. Служащий гостиницы убирает все номера на одном этаже в определенные дни недели, при этом в разные дни он может убирать разные этажи.

## Функциональные требования к web-сервису в рамках предметной области

Работа с системой также предполагает получение следующей информации:

• о клиентах, проживавших в заданном номере, в заданный период времени;

• о количестве клиентов, прибывших из заданного города,

• о том, кто из служащих убирал номер указанного клиента в заданный день

недели,

• сколько в гостинице свободных номеров;

• список клиентов с указанием места жительства, которые проживали в те же дни, что и заданный клиент, в определенный период времени.

Администратор имеет возможность принять на работу служащего, изменить его расписание работы, поселить или выселить клиента. Он также может получить отчет за указанный промежуток времени. Такой отчет должен содержать следующие сведения:

• число клиентов за указанный период в каждом номере с указанием ФИО клиента, города, откуда он прибыл, количества дней проживания,

• общая сумма дохода за каждый номер;

• суммарный доход по всей гостинице.

## Выводы

В результате проведённого анализа предметной области были сформулированы функциональные требования к реализуемому в рамках курсовой работы web-приложению. Разработка модели данных, а также серверной и клиентской части приложения будут описаны в последующих главах.

# Модели данных

## Теория

Модель данных в Django – это одиночный определённый источник информации о данных разработчика. Она содержит необходимые поля и поведение данных, которые он хранит. Обычно, каждая модель относится к одной единственной таблице базы данных. Через инструмент моделей Django позволяет разработчику вносить любые изменения в базу данных, не взаимодействуя с ней напрямую.

## Разработка моделей данных для создаваемого web-приложения

Модели данных для создаваемого web-приложения были разработаны на основе функциональных требований к приложению. Было решено создать 4 модели, представляющие 4 таблицы в базе данных:

1. Клиент;
2. Комната;
3. Рабочий;
4. Расписание работы.
5. Журнал поселений

На рис. 1 показана схема полученной базы данных.

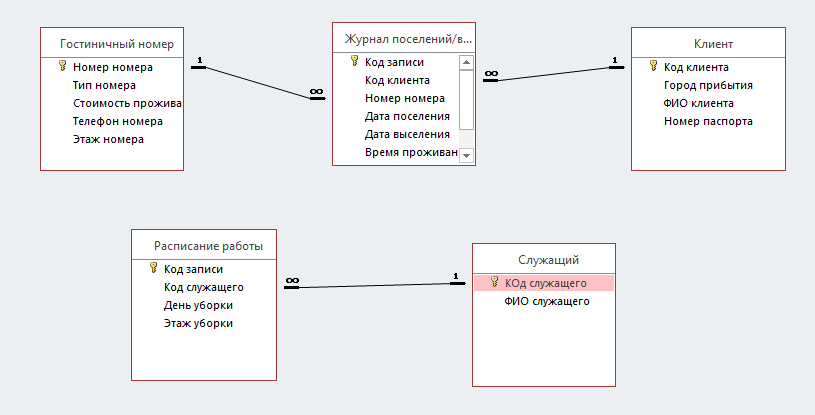


Рисунок 1 – Схема базы данных разрабатываемого web-приложения

Как было сказано ранее, каждая таблица базы данных была задана как модель Django. Эти модели прописываются в файле models.py.

### Модель клиента

Модель Django для клиентов (рис. 2) состоит из следующих полей:

1. *fio* – полное имя, фамилия и отчество клиента, макс. длина 50;
2. *hometown* – город, из которого прибыл клиент, макс. длина 15 ;
3. *id* – автоматически создаваемое поле-счётчик, не прописываемое при задании модели;
4. *passport* – паспортный номер клиента, целое число.

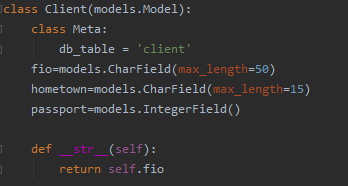


Рисунок 2 – Модель клиента в Django

Как видно на рис. 2, для самой модели с помощью метода \_\_str\_\_ прописано, как в админ-панели будут записаны экземпляры данной модели. Данный метод прописан для всех моделей, поэтому для последующих моделей это не будет описываться.

### Модель служащего

Модель Django для служащего (рис. 3) состоит из следующих полей:

1. *fio* – полное имя, фамилия и отчество;
2. *id* – автоматически создаваемое поле-счётчик, не прописываемое при задании модели.

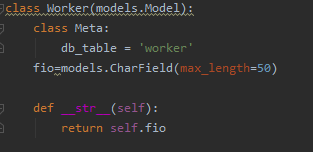


Рисунок 3 – Модель служащего в Django

### Модель комнаты

Модель Django для комнаты (рис. 4) состоит из следующих полей:

1. *number* – номер комнаты, использующийся при упоминании таковой, целое число;
2. *type* – тип комнаты, текстовое поле длинной до 15 символов;
3. *cost* – стоимость проживания за день, целое число;
4. *phone* – личный телефон номера, целое число;
5. *floor* – этаж комнаты, целое число.
6. *id* – автоматически создаваемое поле-счётчик, не прописываемое при задании модели.

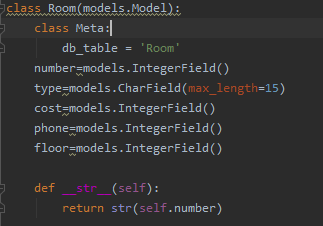


Рисунок 4 – Модель комнаты в Django

### Модель заселения

Модель Django для заселений – ассоциативная модель, осуществляющая связь многое-ко-многому между моделями комнаты и клиента. Она состоит из следующих полей (рис. 5):

1. *client* – клиент, поле-внешний ключ, осуществляющее связь данной модели с моделью Client;
2. *room* – комната, поле-внешний ключ, осуществляющее связь данной модели с моделью Room;
3. *checkin* – дата заселения, поле формата date;
4. *checkout* – дата открепления, поле формата date.
5. *id* – автоматически создаваемое поле-счётчик, не прописываемое при задании модели.

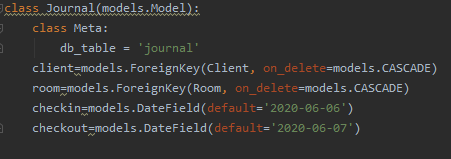


Рисунок 5 – Модель заселения в Django

### Модель расписания работы

Модель Django для расписания работы осуществляет связь один-ко-многим с моделью клиента. Она состоит из следующих полей (рис. 5):

1. *client* – клиент, поле-внешний ключ, осуществляющее связь данной модели с моделью Client;
2. *room* – комната, поле-внешний ключ, осуществляющее связь данной модели с моделью Room;
3. *checkin* – дата заселения, поле формата date;
4. *checkout* – дата открепления, поле формата date.
5. *id* – автоматически создаваемое поле-счётчик, не прописываемое при задании модели.

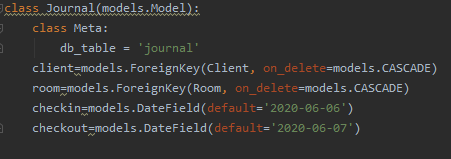


Рисунок 5 – Модель заселения в Django

## Выводы

В результате разработки структуры базы данных web-приложения были созданы схема базы данных и модели для каждой таблицы базы данных из этой схемы, прописаны все поля в созданных моделях. Также были заданы связи между различными моделями. Для каждой модели описан метод \_\_str\_\_.

# Серверная часть

Серверная часть разрабатываемого web-приложения была реализована с помощью средств Django и Django REST Framework.

## Теория

Некоторая информация про Django была представлена в предыдущей главе в контексте моделей Django. Теперь это будет рассмотрено несколько подробнее. Django – это свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC.

MVC – Model-View-Controller («Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

* Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

В отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных [1].

Django REST framework – это удобный и гибкий фреймворк, содержащий набор инструментов для конструирования API web-приложений [2].

## Админ-панель

### Теория

Админ-панель в Django – это автоматический интерфейс админа. Он считывает метаданные с записанных в файле models.py моделей и создаёт быстрый интерфейс, центрированный на моделях, с помощью которого пользователи с достаточным уровнем доступа могут управлять контентом сайта. Использование админ-панели рекомендовано ограничить внутренним управлением в организации. Она не создавалась, чтобы представлять собой весь фронтэнд сайта. Админ-панель наиболее важна на начальных этапах разработки web-сервиса.

### Код

Прописывается она в файле admin.py образом, представленным на рис. 6.

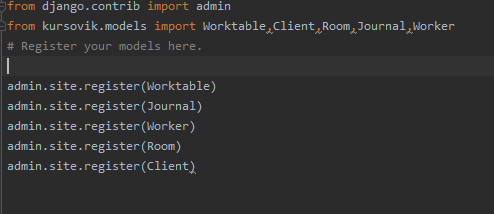


Рисунок 6 – Содержимое файла admin.py

Сначала импортируются модели приложения, далее при помощи встроенного инструмента admin из django.contrib они регистрируются в админ-панели.

### Внешний вид

Зайти на админ-панель можно по адресу, прописанному в файле urls.py всего сайта (рис.  7).

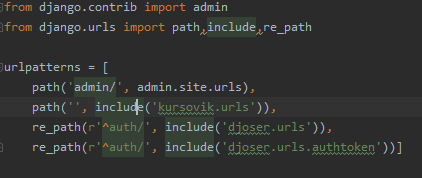


Рисунок 7 – Содержимое файла urls.py всего сайта

На рис. 9, 10 и 11 представлено, как выглядит админ-панель.

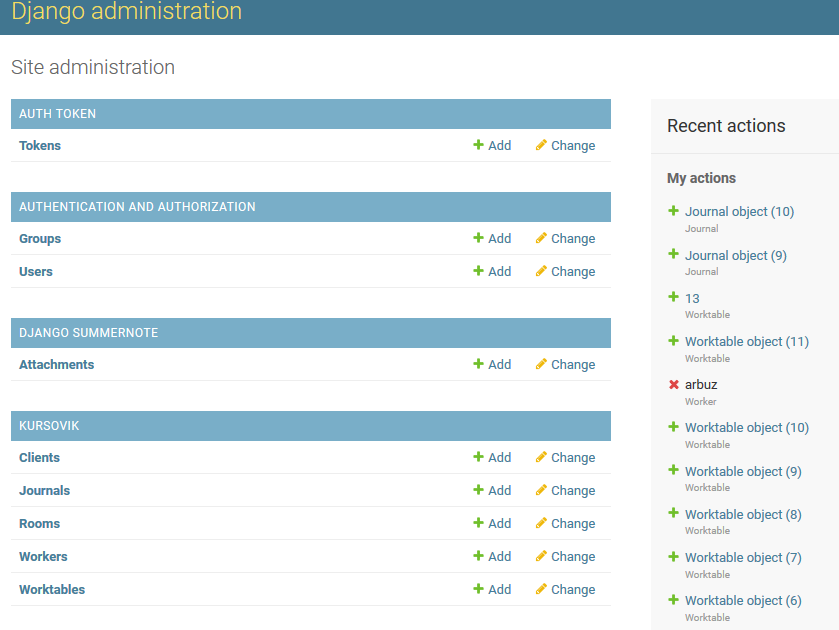


Рисунок 8 – Общий вид админ-панели

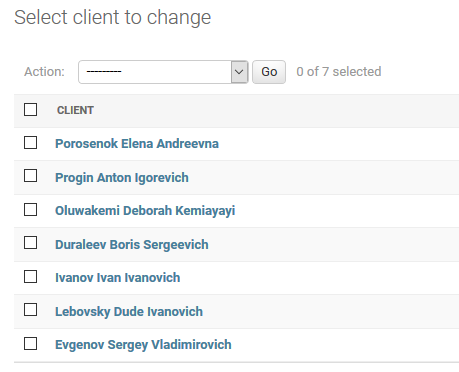


Рисунок 9 – Модель в админ-панели

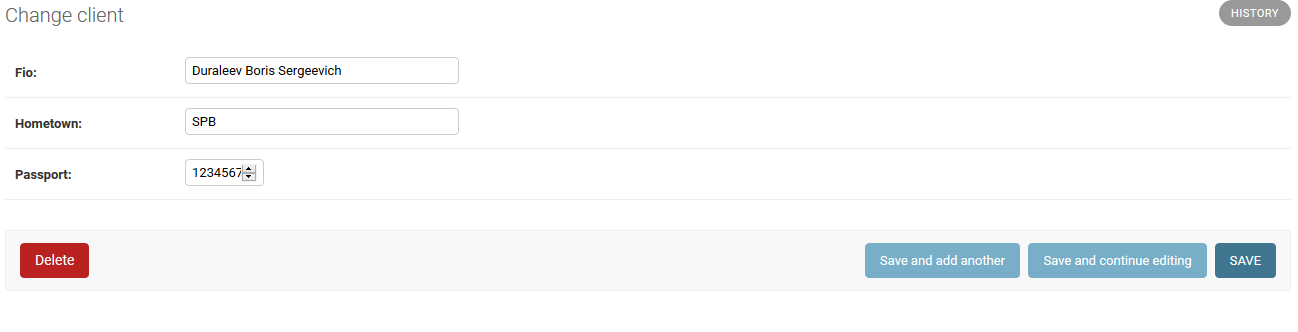


Рисунок 10 – Изменение экземпляра модели в админ-панели

## Django REST Framework

### Общий принцип работы

Django REST framework — это API фреймворк для Django в стиле REST. У создания фреймворков в таком стиле есть два основных принципа:

1. Сервер не должен ничего знать о текущем состоянии Клиента. В запросе от Клиента должна быть вся необходимая информация для обработки этого запроса Сервером.
2. Каждый ресурс на Сервере должен иметь определённый Id, а также уникальный URL, по которому осуществляется доступ к этому ресурсу.

На данный момент мы можем найти фреймворк для создания приложений в стиле REST практически для каждого языка программирования, используемого в веб-разработке. Логика построения Web API на Сервере в этих фреймворках реализована одинаково.

Действия для управления данными привязаны к определенным HTTP-методам. Существует несколько стандартных действий для работы с данными – это Create, Read, Update, Delete. Часто их обобщают как CRUD.

Для создания объекта используется http-метод POST.

Для чтения — http-метод GET.

Для изменения — http-метод PUT.

Для удаления — http-метод DELETE [3].

### URL-пути серверной части web-приложения

На рис. 7 видно, что прописан путь, по которому подключаются все пути, прописанные в файле urls.py разрабатываемого приложения. Содержимое этого файла представлено на рис. 12.

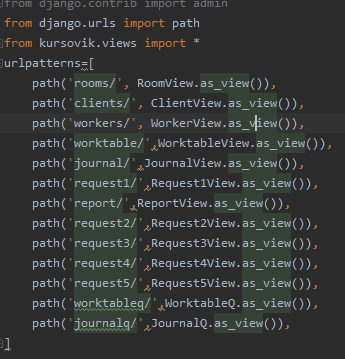


Рисунок 11 – Содержимое файла urls.py приложения

Здесь прописаны все пути, который вызывают определённые представления, находящиеся в файле views.py.

### Сериализаторы

Прежде чем переходить к рассматриванию представлений необходимо поговорить про сериализаторы. Сериализация — это преобразование объекта или дерева объектов в какой-либо формат с тем, чтобы потом эти объекты можно было восстановить из этого формата Сериализаторы, соответственно, – это то, что осуществляет сериализацию. В Django сериализаторы переводят данный из базы данных в формат json, и расположены в файле serializers.py (рис. 12).

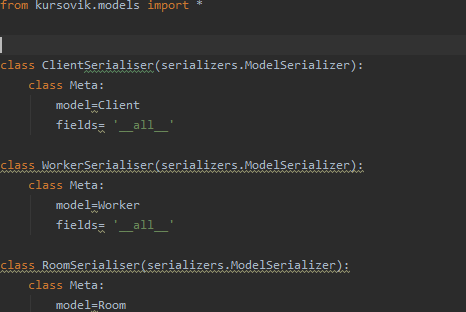


Рисунок 12 – Некоторые сериализаторы и шапка файла serializers.py

Сериализаторы создаются достаточно просто: объявляется модель и задаётся список необходимых полей.

### Представления

В Django представления задаются в файле views.py. На рис. 14 представлены первые строки данного файла.

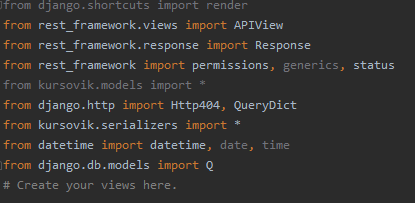


Рисунок 13 – Шапка файла views.py

Во views.py импортируются необходимые для создания представлений компоненты, а также модели и сериализаторы разрабатываемого web-приложения.

Далее рассмотрены представления, вызываемые конкретными путями (path).

1. Пути ‘rooms/’ и ‘clients/’ вызывают соответственно представления RoomView и ClientView. Их строение представлено на рис. 14.

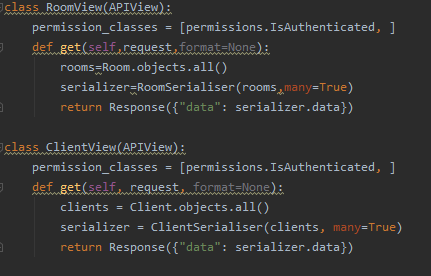


Рисунок 14 – Представление RoomView и ClientView

В начале задаётся класс пользователей, которые могут вызывать данное представление. Эти представления доступны для всех авторизованных пользователей. Далее описываются методы http-запросов. У этих двух представлений только один метод: get. Он получает все объекты соответственной модели, сериализирует их, используя Serializer (Приложение А) и выдаёт полученные сериализатором данные в качестве ответа. Посмотреть на эти данные можно по указанному адресу (рис. 15).

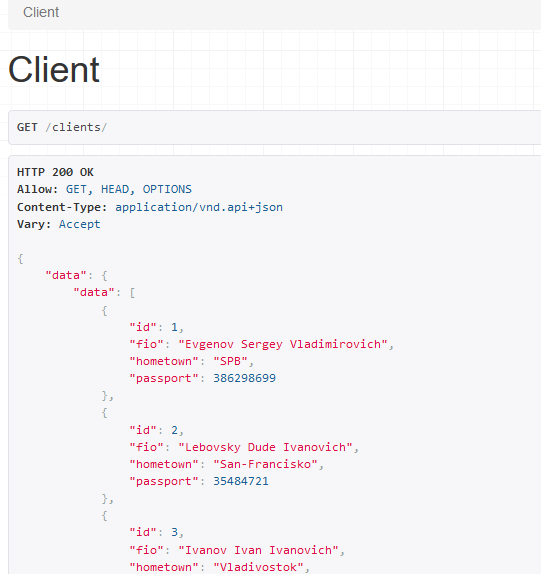


Рисунок 15 – Страница по адресу ‘127.0.0.1:8000/clients/

Страница получения API, по адресу соответственного отображения, выглядит приблизительно одинаково для всех отображений. Для удобства она не будет описана в каждом отображении, а перенесена в Приложение B.

1. Путь 'journal/' вызывает представление JournalView. Его строение представлено на рис. 16, 17, 18.

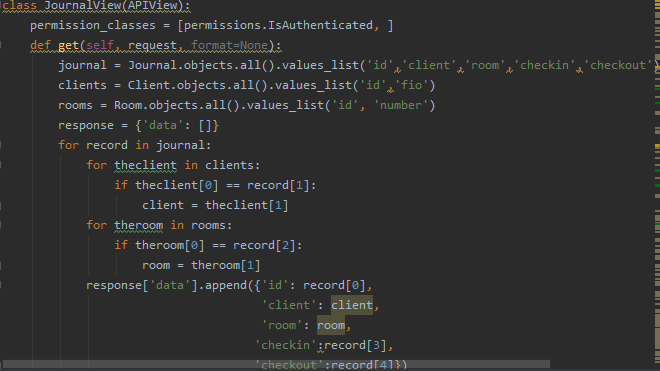


Рисунок 16 – Представление JournalView (метод get)

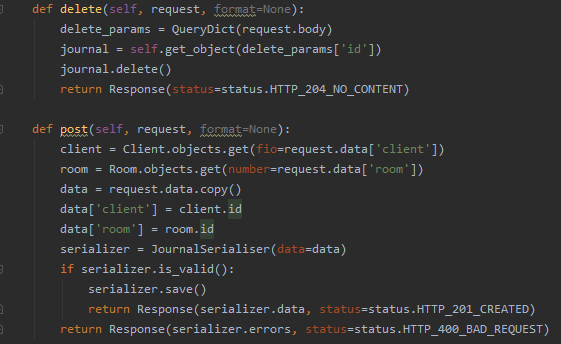


Рисунок 17 – Представление JournalView (методы delete и post)

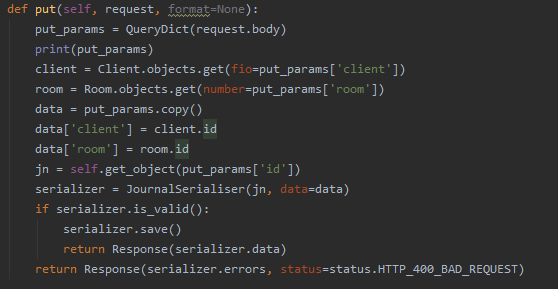


Рисунок 18 – Представление JournalView (метод put)

Данное представление имеет не только метод get, но и методы post, delete и put. Post заключается в том, что он получает из запроса данные, которые используются для создания нового объекта модели Reader. Затем эти данные проверяются на валидность, и сохраняются. Delete также получает параметры из запроса, однако при работе удаляет соответственные записи модели. Put нужен для того, чтобы сохранять, обновлять существующие данные при помощи новых поступивших параметров.

1. Путь 'worktable/' вызывает представление JournalView. Его строение представлено на рис. 19, 20, 21.

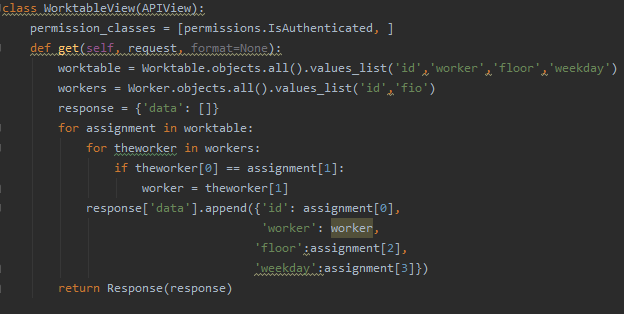


Рисунок 19 – Представление WorktableView (метод get)

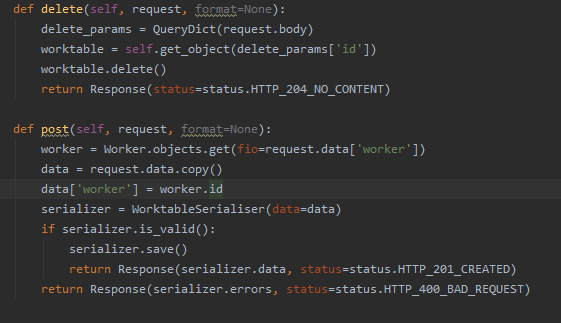


Рисунок 20 – Представление WorktableView (методы delete и post)

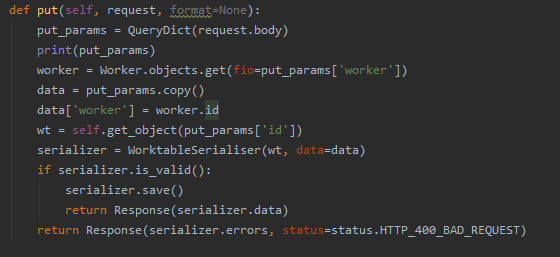


Рисунок 21 – Представление WorktableView (метод put)

Как и представление JournalView, данное представление имеет не только метод get, но и методы post, delete и put.

1. Путь 'worker/' вызывает представление WorkerView. Его строение представлено на рис. 22, 23

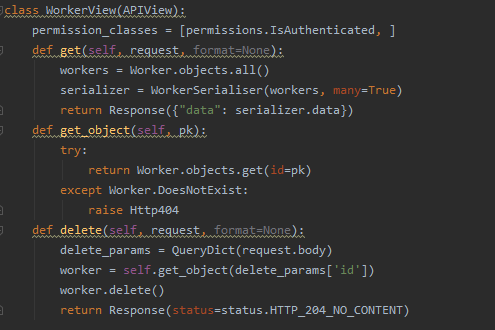


Рисунок 22 – Представление WorkerView (методы get и delete)

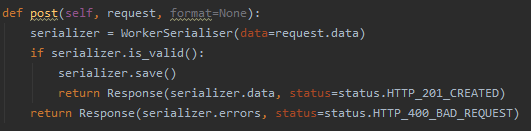


Рисунок 23 – Представление WorkerView (метод post)

Данное представление имеет не только метод get, но и методы post и delete. Post заключается в том, что он получает из запроса данные, которые используются для создания нового объекта модели Reader. Затем эти данные проверяются на валидность, и сохраняются. Delete также получает параметры из запроса, однако при работе удаляет соответственные записи модели.

1. Путь 'worktableq/' вызывает представление WorktableQ. Его строение представлено на рис. 24

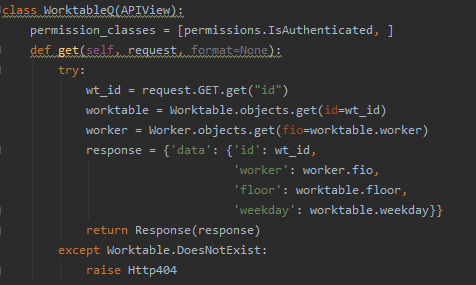


Рисунок 24 – Представление WorktableQ

Данное представление имеет только метод get. Оно используется для того, чтобы в процессе реального времени принимать некоторые введенные клиентом данные, обращаться в базу данных и выводить дополнительные опции. О его назначении более подробно написано в разделе о components/Worktable.vue

1. Путь 'journalq/' вызывает представление JournalQ. Его строение представлено на рис. 25

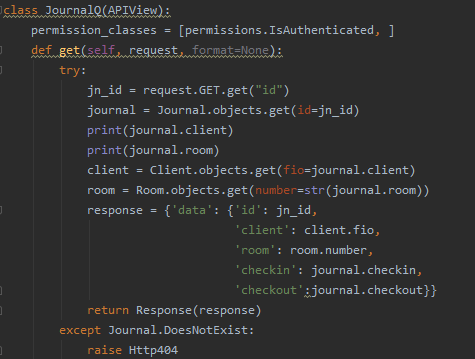


Рисунок 25 – Представление JournalQ

Данное представление имеет только метод get. Оно используется для того, чтобы в процессе реального времени принимать некоторые введенные клиентом данные, обращаться в базу данных и выводить дополнительные опции. О его назначении более подробно написано в разделе о components/Journal.vue

1. Путь 'request1/' вызывает представление Request1View. Его строение представлено на рис. 26

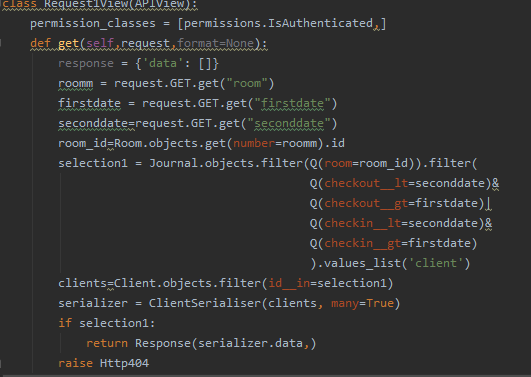


Рисунок 26 – Представление Request1

Данное представление имеет только метод get. Оно принимает на вход аргументы запроса (комнату и две даты), находит в БД объекты, удовлетворяющие условиям, после чего сериализует их и отправляет ответ.

1. Путь 'request2/' вызывает представление Request2View. Его строение представлено на рис. 27

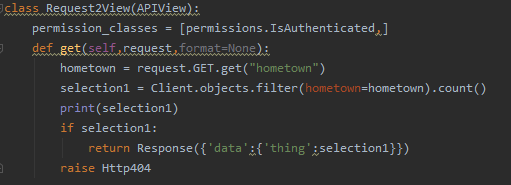


Рисунок 27 – Представление Request2

Данное представление имеет только метод get. Оно принимает на вход аргументы запроса (город), находит в БД клиентов, удовлетворяющих условиям, после чего сериализует их и отправляет ответ.

1. Путь 'request3/' вызывает представление Request3View. Его строение представлено на рис. 28

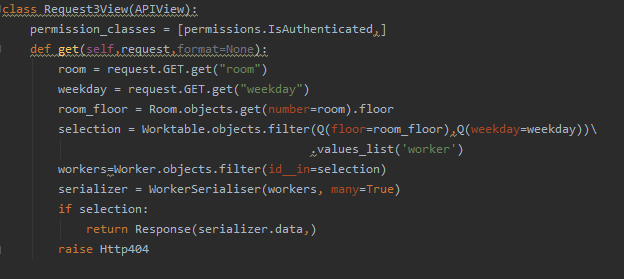


Рисунок 28 – Представление Request3

Данное представление имеет только метод get. Оно принимает на вход аргументы запроса (комнату и день недели), находит в БД записи уборки, удовлетворяющие условиям, затем сопоставляет их со служащим, после чего сериализует их и отправляет ответ.

1. Путь 'request4/' вызывает представление Request4View. Его строение представлено на рис. 29

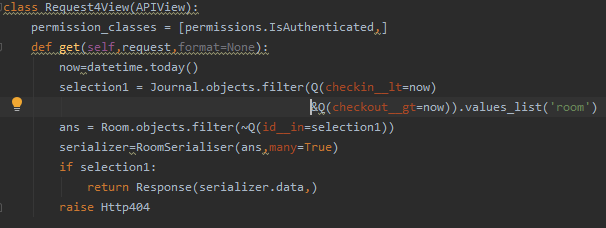


Рисунок 29 – Представление Request4

Данное представление имеет только метод get. Оно не принимает на вход аргументы, вместо этого оно узнает текущую дату и находит в БД записи заселений, удовлетворяющие условиям, сопоставляет их с комнатами, после чего сериализует их и отправляет ответ.

1. Путь 'request5/' вызывает представление Request5View. Его строение представлено на рис. 30, 31

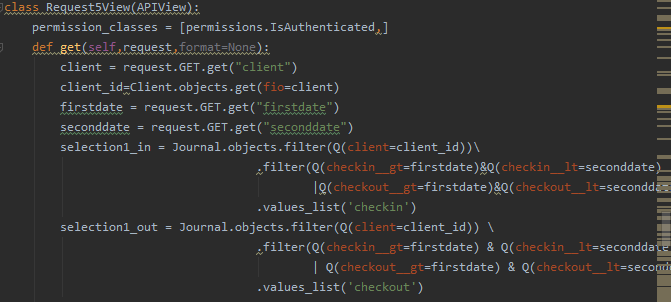


Рисунок 30 – Представление Request5 (часть 1)

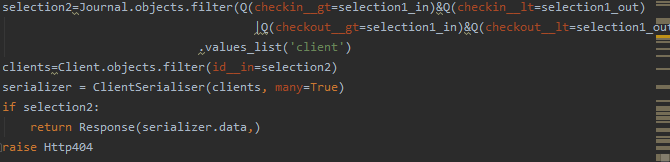


Рисунок 31 – Представление Request5 (часть 2)

Данное представление имеет только метод get. Оно принимает на вход аргументы ввода (имя клиента и 2 даты), после чего находит в БД записи заселений, удовлетворяющие условиям, присваивает их двум переменным даты заселения и выселения соответственно, сопоставляет их с клиентами, после чего сериализует их и отправляет ответ.

## Выводы

В результате разработки серверной части web-приложения были прописаны сериализаторы в файл serializers.py, пути – в файл urls.py и представления – в файл views.py. С помощью изменения данных трёх файлов были успешно решены почти все задачи, которые были поставлены в рамках выполнения функциональных требований, что будет продемонстрировано далее, в описании клиентской части разрабатываемого web-приложения.

# Клиентская часть

Клиентская часть разрабатываемого web-приложения была реализована с помощью средств JavaScript-фреймворка Vue.js.

## Теория

Для начала стоит сказать, что такое JavaScript. JavaScript – это мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Он обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Название «JavaScript» является зарегистрированным товарным знаком корпорации Oracle в США.

Также, необходимо определить, чем является Node.js. Node.js – это программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel, low.js и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

И теперь можно сказать, что Vue.js – это JavaScript-фреймворк на основе Node.js с открытым исходным кодом для создания пользовательских интерфейсов. Легко интегрируется в проекты с использованием других JavaScript-библиотек. Может функционировать как веб-фреймворк для разработки одностраничных приложений в реактивном стиле [4].

## Компоненты

Компоненты – это переиспользуемые экземпляры Vue со своим именем, файлы .vue. Их можно использовать как пользовательский тег внутри корневого экземпляра Vue, созданного с помощью new Vue. Так как компоненты это переиспользуемые экземпляры Vue, то они принимают те же опции что и new Vue, такие как data, computed, watch, methods, хуки жизненного цикла [4].

### Файл index.js

Файл index.js играет примерно ту же роль, что urls.py в серверной части web-приложения. В нём прописывается, какие компоненты будут загружаться по какому адресу. На рис. 32 и рис. 33 приведено содержание данного файла.

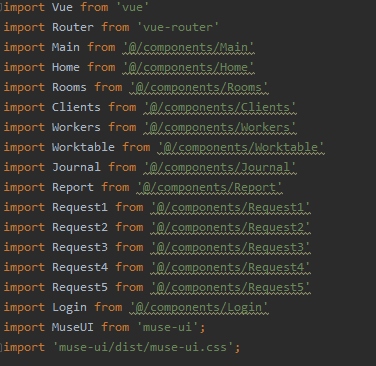


Рисунок 32 – Содержимое файла index.js (шапка)

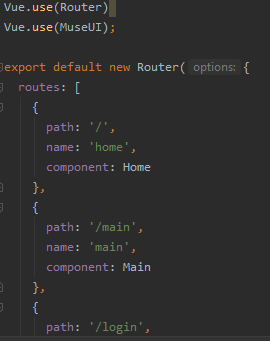


Рисунок 33 – Содержимое файла index.js (примеры путей)

Как мы видим, в шапке файла импортируются необходимые компоненты, далее запускается роутер, и ниже приведён список объектов, в каждом из которых прописан путь, название данного пути/объекта и загружаемый компонент. В данном файле присутствуют все ссылки, переход по которым вызывается взаимодействием с сайтом. Для краткости они также приведены в приложении.

## Компонент Home

Home – домашняя страница нашего сайта, появляющаяся при первом/незарегистрированном заходе.

### Код

В данном приложении в компоненте Home прописано меню приложения, а также в него загружается компонент Reader (рис. 34).

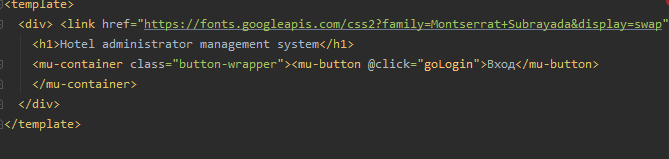


Рисунок 34 – Тэг <template ...> в файле Home.vue

Тэг <template> – это один из трёх тегов файла .vue. В нём прописывается html-код, описывающий внешний вид компонента. Два других тэга файла .vue – это <script> (рис. 35), в котором прописан код на языке JavaScript, который выполняется в данном компоненте, и <style> – необязательный тег, в котором прописываются стили css, используемые в данном файле (рис. 36).

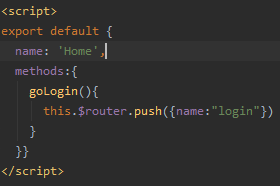


Рисунок 35 – Тэг <script …> компонента Home

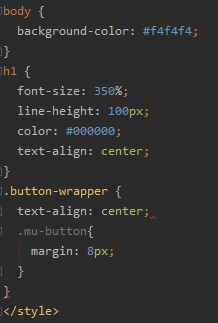


Рисунок 36 – Тэг <style ...> компонента Home

Как видно по рис. 30, в теге <script ...> импортируются необходимые компоненты с помощью команд(ы) import и экспортируются все данные, методы и т.д. с помощью команды export default {}. Первым экспортируется имя компонента, затем импортированные сюда другие компоненты, дальше вычисляемые значения и в конце методы. В случае большинства компонентов также экспортируются инструкции, которые выполняются при открытии страницы (created () {...}), и множество переменных с различными значениями (data () {return {...}}). Это будет присутствовать в других компонентах.

### Отображение на сайте

На рис. 37 представлен вид данного компонента в браузере.

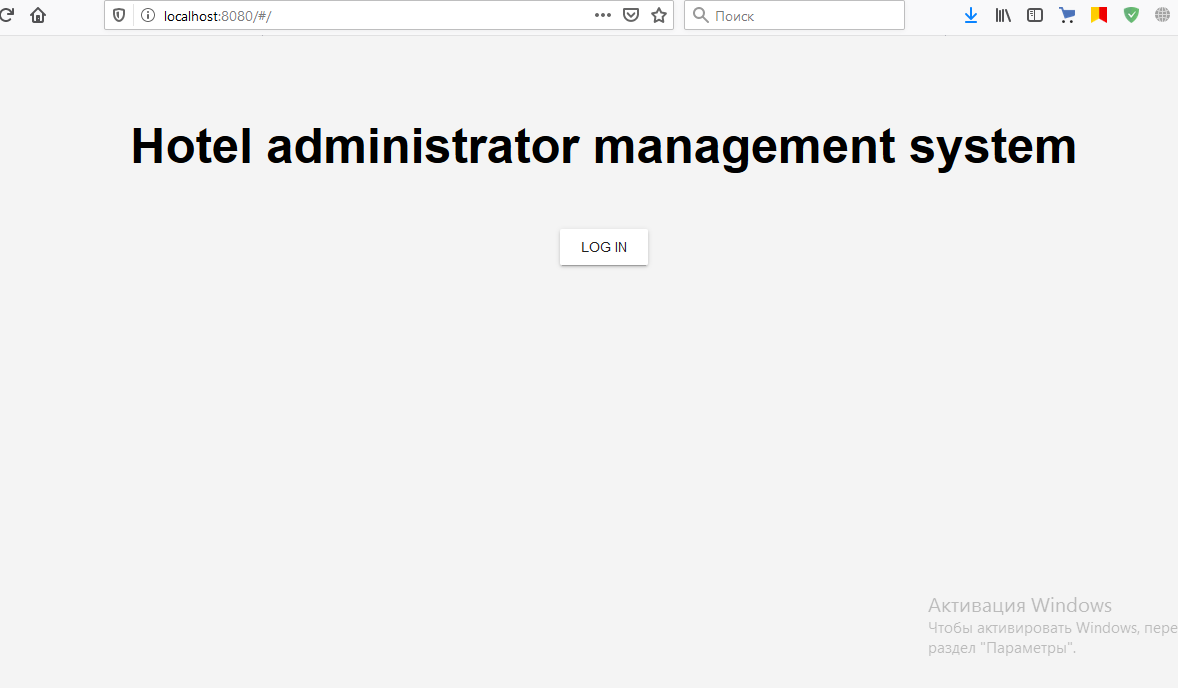


Рисунок 37 – Вид компонента Home в браузере

Базовая страница не имеет в себе ничего примечательного, кроме кнопки, перенаправляющей на login/, чтобы войти в систему.

## Компонент Login

Login – компонент, отвечающий за авторизацию на сайте, открывается по пути ‘/login’.

### Код

На рис. 38, рис. 39 и рис. 40 представлен весь код компонента Login.

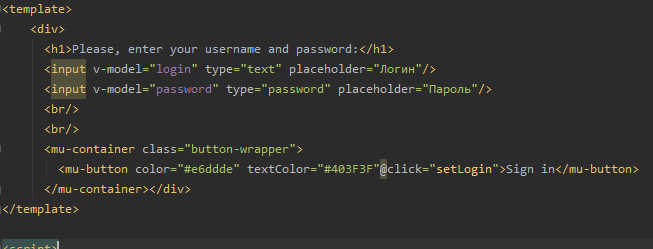


Рисунок 38 – Содержимое файла Login.vue (часть 1)

Внутри кода <template ...> (рис. 34) присутствуют тэги <input ...> и <button ...>, который позволяет делать простейшие формы.

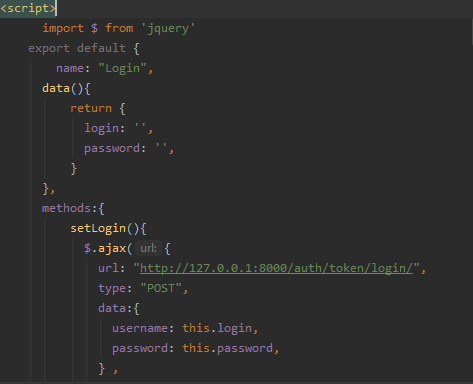


Рисунок 39 – Содержимое файла Login.vue (часть 2)

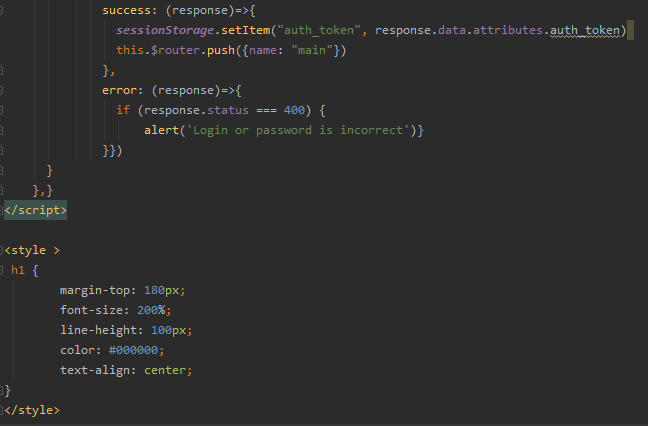


Рисунок 40 – Содержимое файла Login.vue (часть 3)

В тэге <script ...> в теле операции export default {} присутствует ранее упомянутый элемент data () {return {}} (рис. 35). Внутри return’а данного подобия функции задаются все переменные, которые могут использоваться далее в методах, при обращении к ним ‘*this.имяПеременной*’. Далее в той же операции прописан метод setLogin () {...}, который авторизует пользователя с использованием ajax-запроса и, в случае успеха, переносит его на главную страницу.

### Отображение на сайте

На рис. 41 представлен вид данного компонента в браузере.

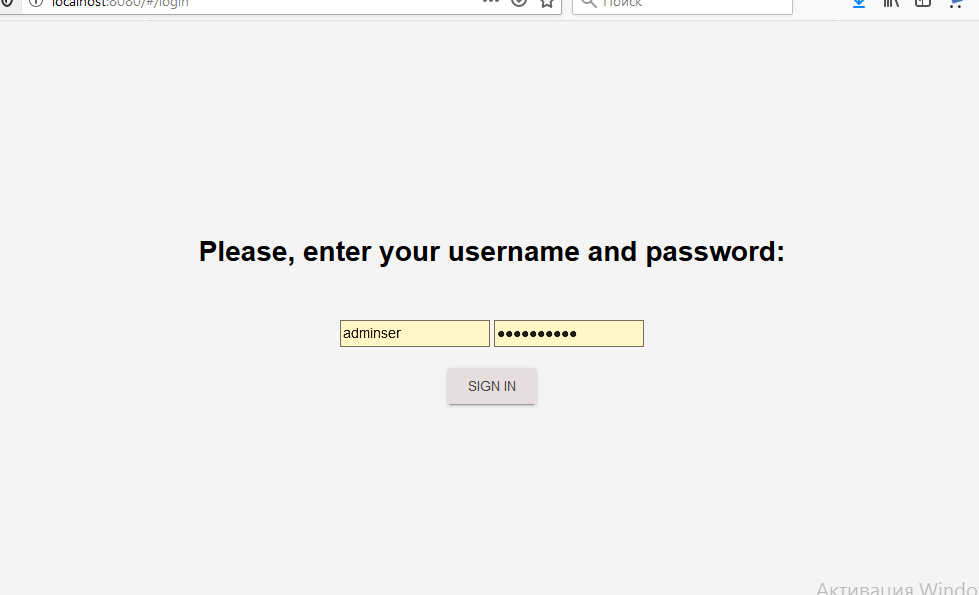


Рисунок 41 – Вид компонента Login в браузере

При правильном вводе логина и пароля и нажатии кнопки «Войти» открывается основная страница сайта.

## Компонент Main

Данный компонент отвечает за основную страницу системы, где администратор может сделать выбор среди предоставленных возможностей.

### Код

На рис. 42, 43, 44 представлен код тэга <template ...> компонента Main.

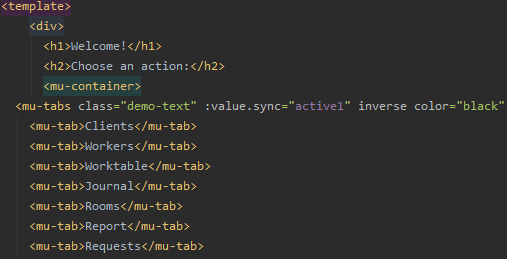


Рисунок 42 – Тэг <template ...> компонента Main (табло)



Рисунок 43 – Тэг <template ...> компонента Main (появляющиеся после нажатия на табло кнопки)

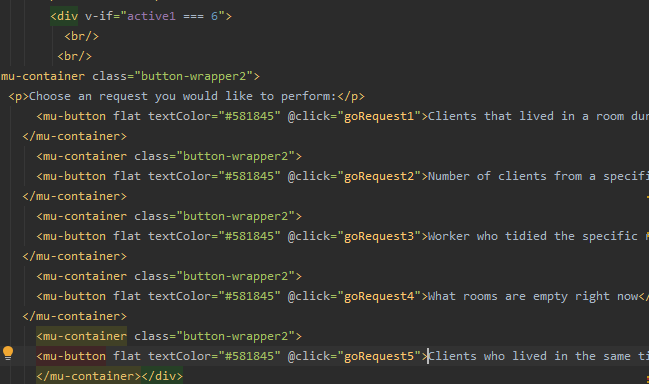


Рисунок 44 – Тэг <template ...> компонента Main (отдельный вид для запросов)

Как видно в тэге <template ...>, в данном компоненте прописаны функции goClients () {} , goWorkers () {} и еще несколько других – для каждой ссылки.



Рисунок 45 – Примеры функций компонента Main

Целиком код тэгов <script ...> и <style ...> данного компонента приведён на рисунках 1-4 в Приложении Г.

### Отображение на сайте

Вид данного компонента в браузере представлен на рис. 46.

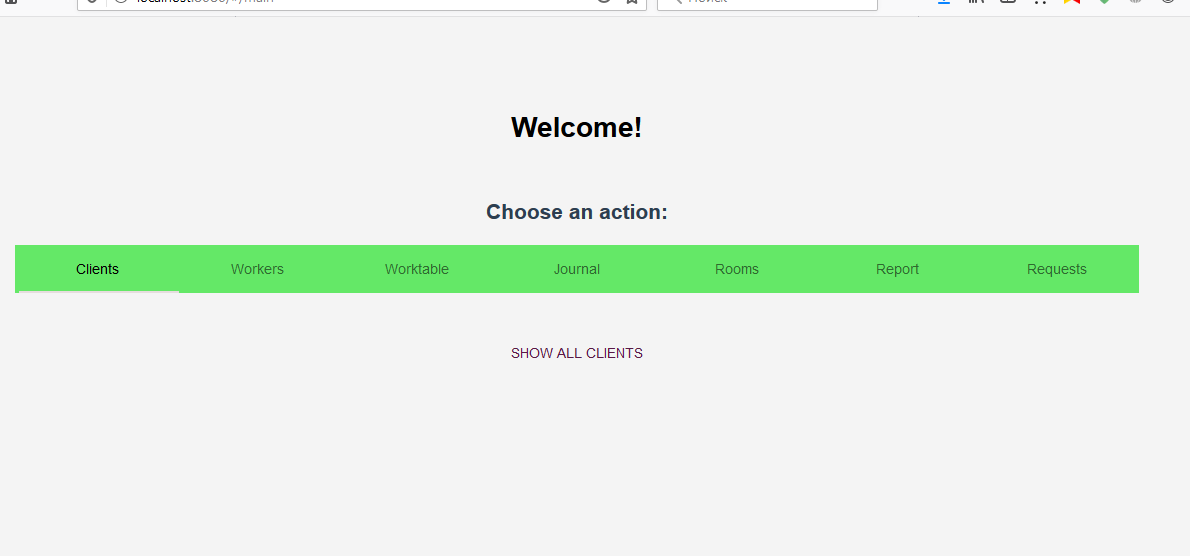


Рисунок 46 – Отображение компонента Main на главной странице сайта

## Компонент

Данный компонент отвечает за отображение карточки читателя и книг, которые за ним закреплены.

### Код

На рис. 47 представлен код тэга <div ...>, расположенного внутри тэга <template ...> компонента Clients, отвечающего за отображение всех клиентов.



Рисунок 47 – Template компонента Clients

Основная вещь в этом шаблоне – таблица данных, построенная на описанных ниже принятых данных и столбцах. Она также отсортирована с помощью отдельного метода.

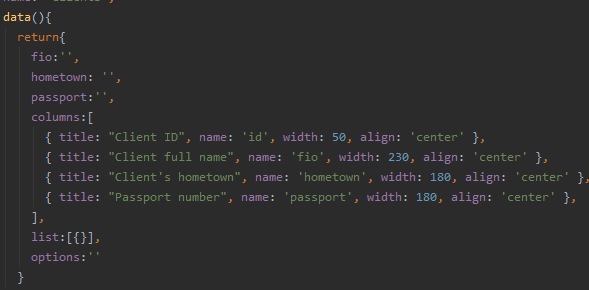


Рисунок 48– Передаваемые поля, данные и столбцы

Главный метод компонента Clients – это метод по выгрузке данных пользователей, loadClients () {} (рис. 49).

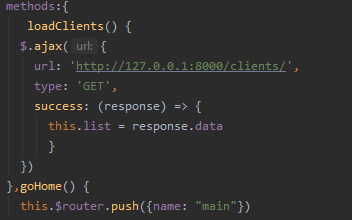


Рисунок 49 – Метод loadClients компонента Clients

На этом рисунке также можно увидеть метод goHome. Он отвечает за кнопку «Home» в углу экрана, которая при нажатии переносит на главную страницу. Эти кнопка и метод встречаются во всех дальнейших компонентах. Целиком код данного компонента приведён на рисунках 5-12 в Приложении Г.

### Отображение на сайте

Вид данного компонента в браузере представлен на рис. 50

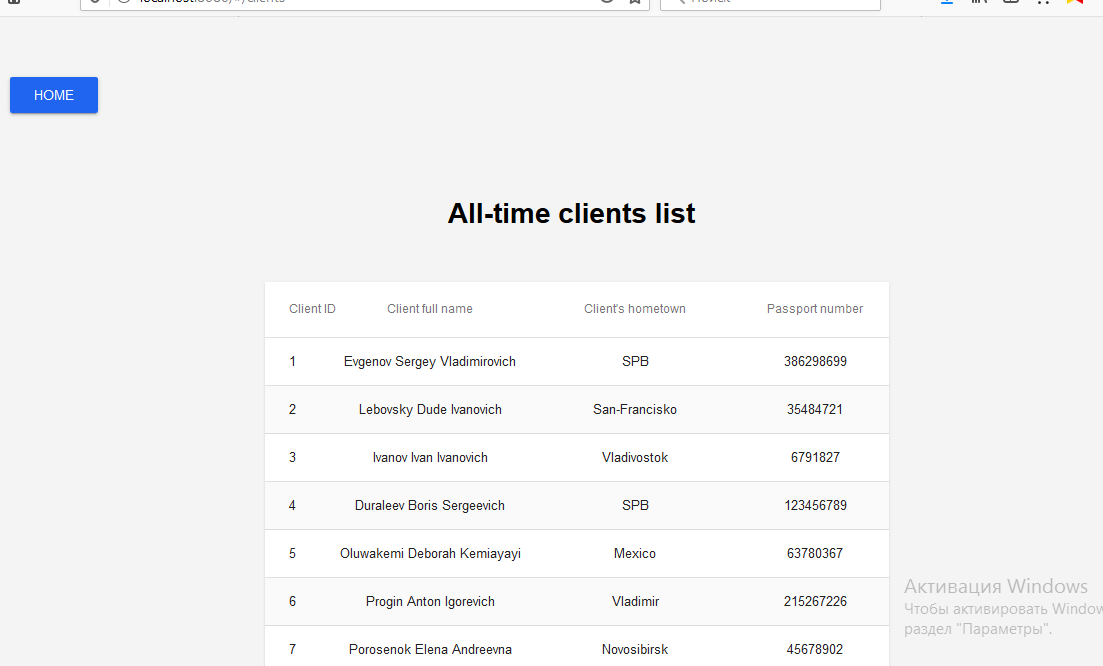


Рисунок 50– Отображение компонента Clients

## Компонент Rooms

Данный компонент отображает все комнаты, существующие в гостинице.

### Код

Код Rooms почти полностью повторяет код Clients, не считая переменных, зависящих от получаемых данных (рис.51, 52).

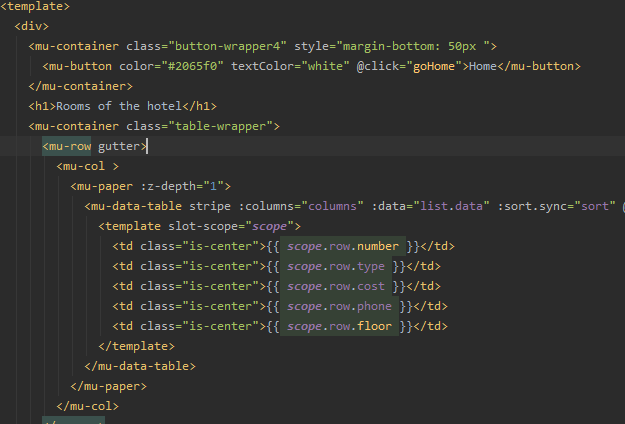


Рисунок 51 – Компонент Rooms (часть 1)

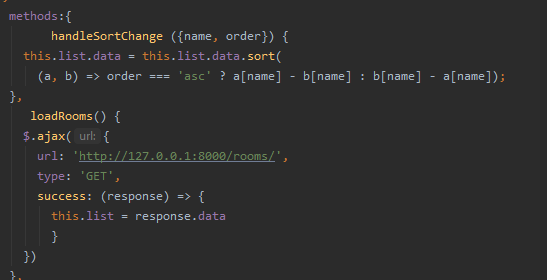


Рисунок 52 – Компонент Rooms (часть 2)

Здесь можно увидеть также метод handleSortChange. Он применяется в тех отображениях, где в таблице данных нужна возможность сортировки.

Целиком код данного компонента приведён в Приложении Д.

### Отображение на сайте

Вид данного компонента в браузере представлен на рис. 53

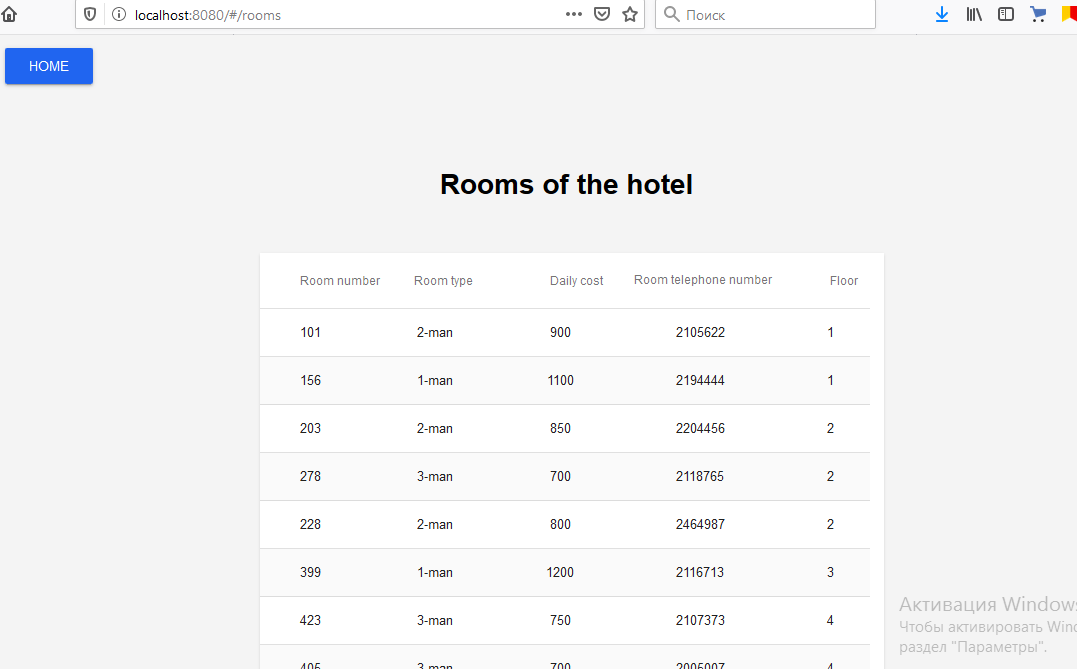


Рисунок 53 – Отображение Rooms на сайте

## Компоненты Worker, Worktable и Journal

Данные компоненты отображают всех служащих, все записи в журнале работы и все записи заселений соответственно, а также позволяют их редактировать, удалять и вносить новые. Полный код данных трёх компонентов приведён в Приложениях Е, Ж и З соответственно. Разберем на примере Journal их общую структуру:

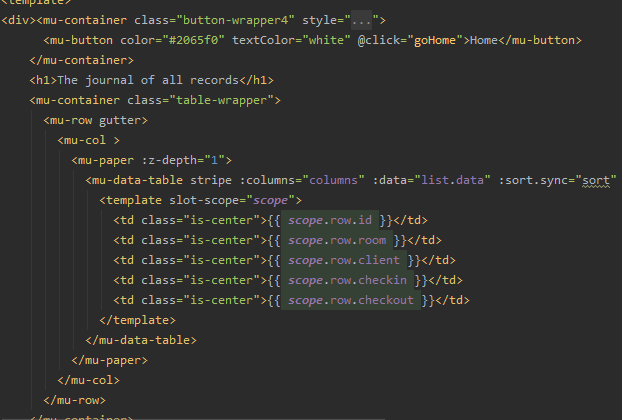


Рисунок 54 – компонент Journal (таблица данных)

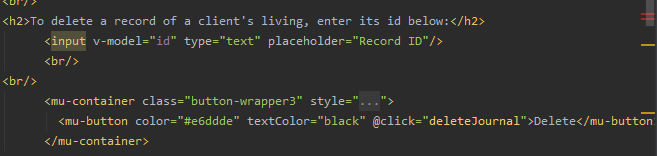


Рисунок 55 – компонент Journal (удаление)

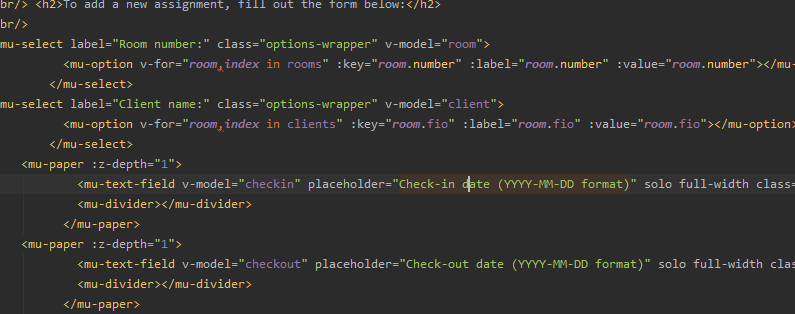


Рисунок 56 – компонент Journal (добавление данных)

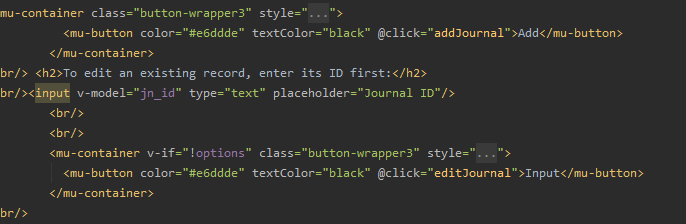


Рисунок 57 – компонент Journal (выбор ID ля изменения записи)

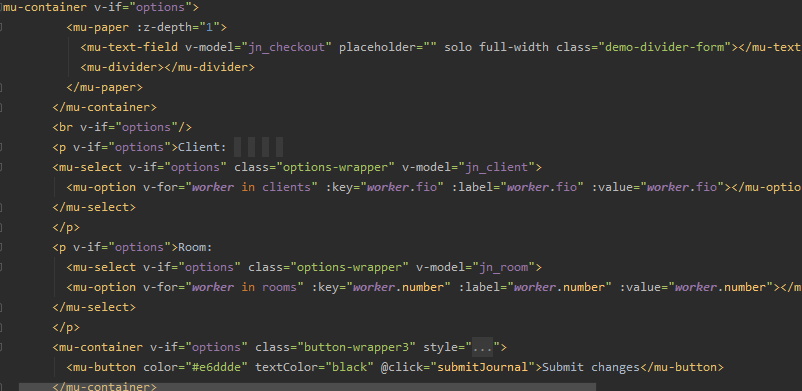


Рисунок 58 – компонент Journal (изменение соответствующих данных)

Далее представлены не описанный ранее метод, используемый выше для изменения данных, связанный с JournalQ:

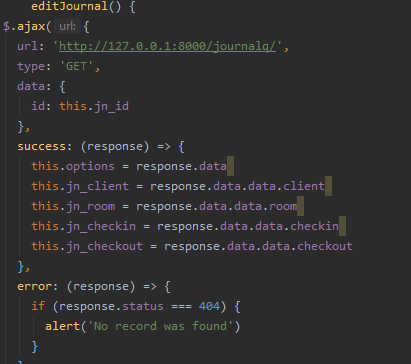


Рисунок 59 – компонент Journal (метод для выбора записи)

Их отображение представлено на рис. 60-63.

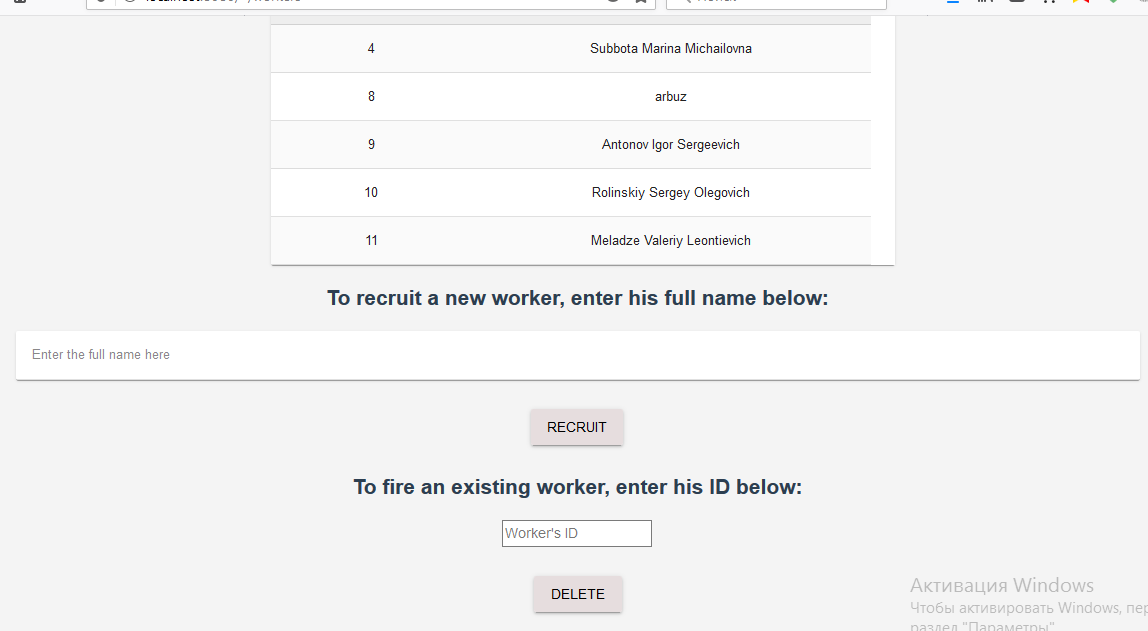


Рисунок 60 – Отображение компонента Workers

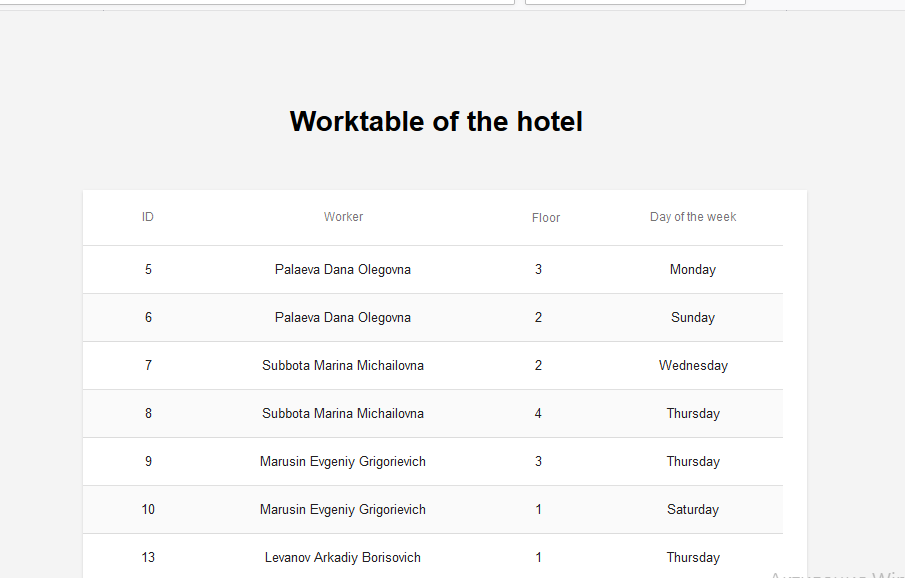


Рисунок 61 – Отображение компонента Worktable (часть 1)

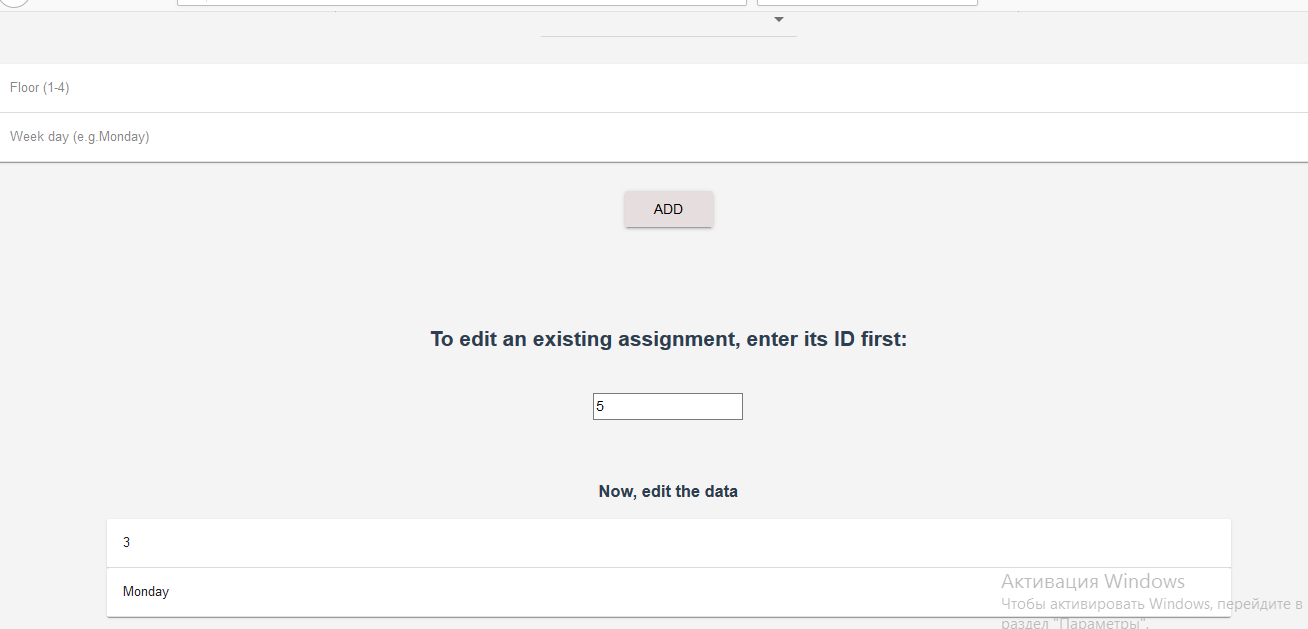


Рисунок 62 – Отображение компонента Worktable (часть 2)

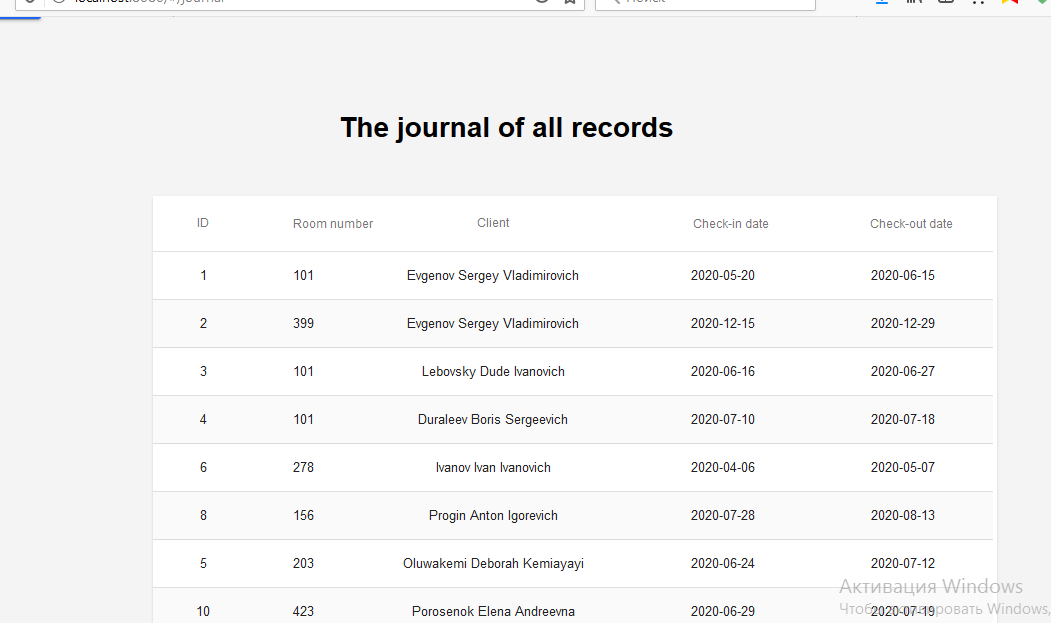


Рисунок 63 – Отображение компонента Journal

## Компоненты Request1…5 и Report

Данные компоненты предлагают сделать конкретные запросы к базе данных и получить информацию. Их View уже разбирались выше – это запросы на сложную выборку данных. Их отображение в шаблоне отличается лишь различными HTML или MUSE-UI тэгами, которые ранее уже рассматривались в этой работе. Методы vue-js, описывающие их, также практически идентичны уже рассмотренным ранее методам. Полный код для них вы можете найти в приложениях И и К. На рисунках 64-68 изображены страницы сайтов, за которые отвечают эти компоненты:

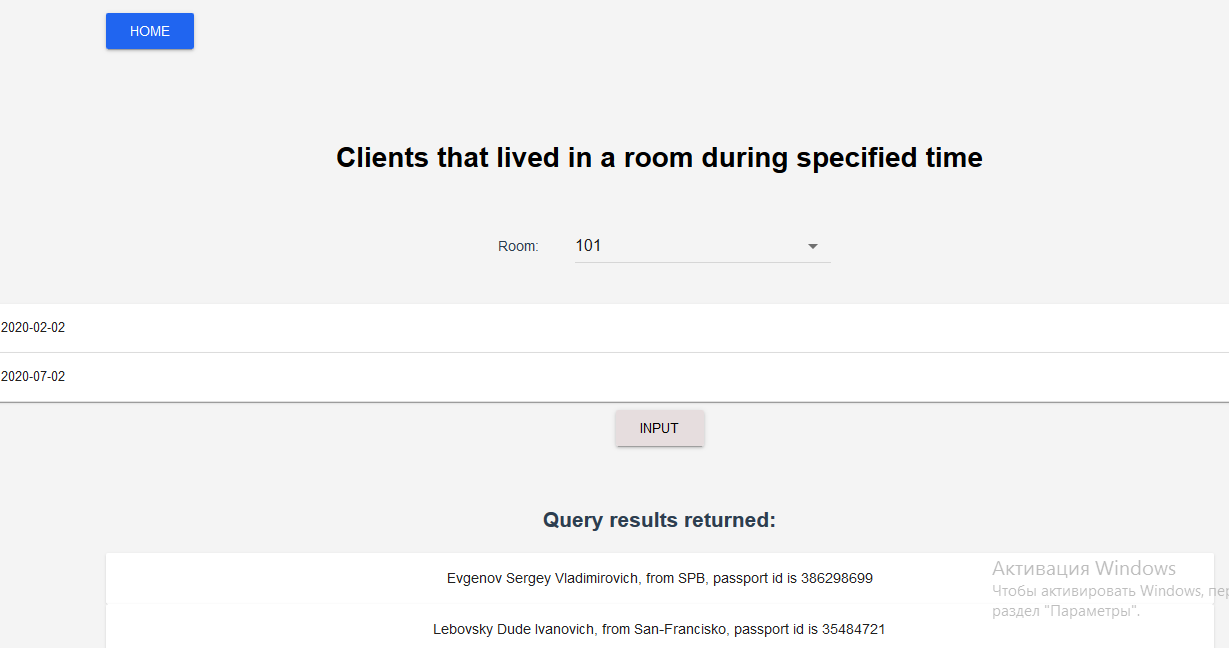


Рисунок 64 – Отображение компонента Request1

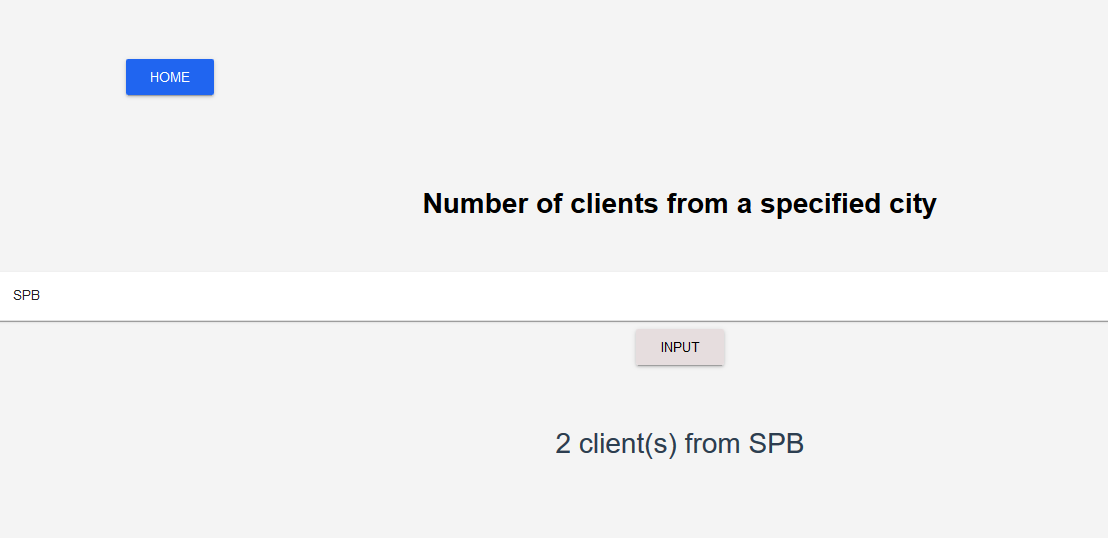


Рисунок 65 – Отображение компонента Request2

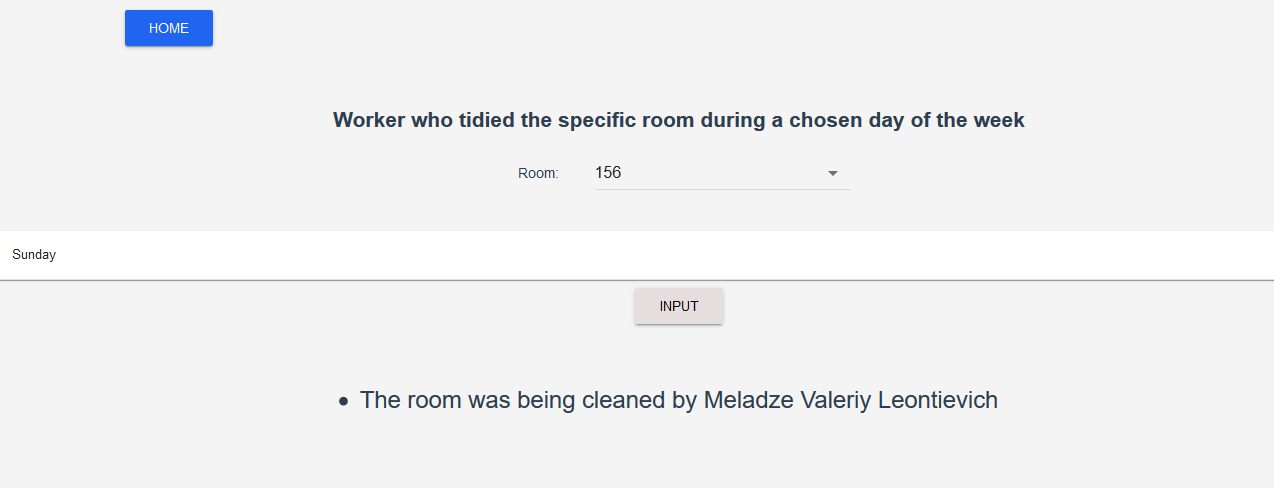


Рисунок 66 – Отображение компонента Request3

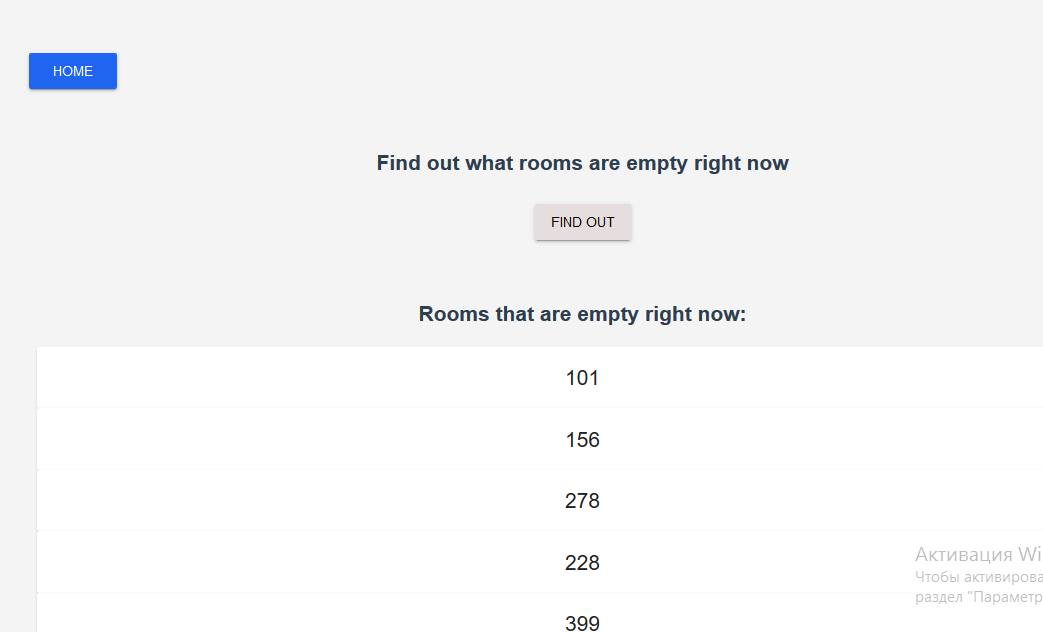


Рисунок 67 – Отображение компонента Request4

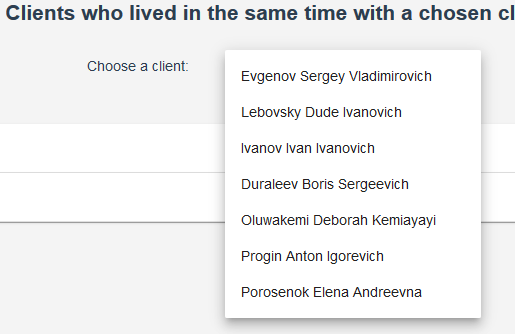


Рисунок 68 – Отображение выпадающего списка из компонента Request5 (присутствует и в других)

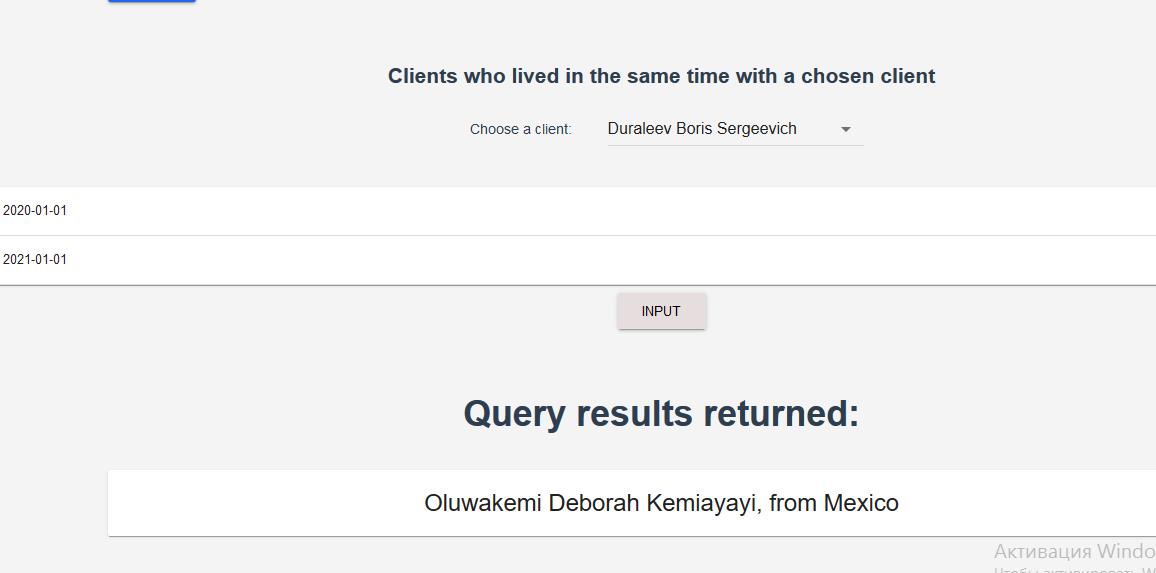


Рисунок 69 – Отображение компонента Request5

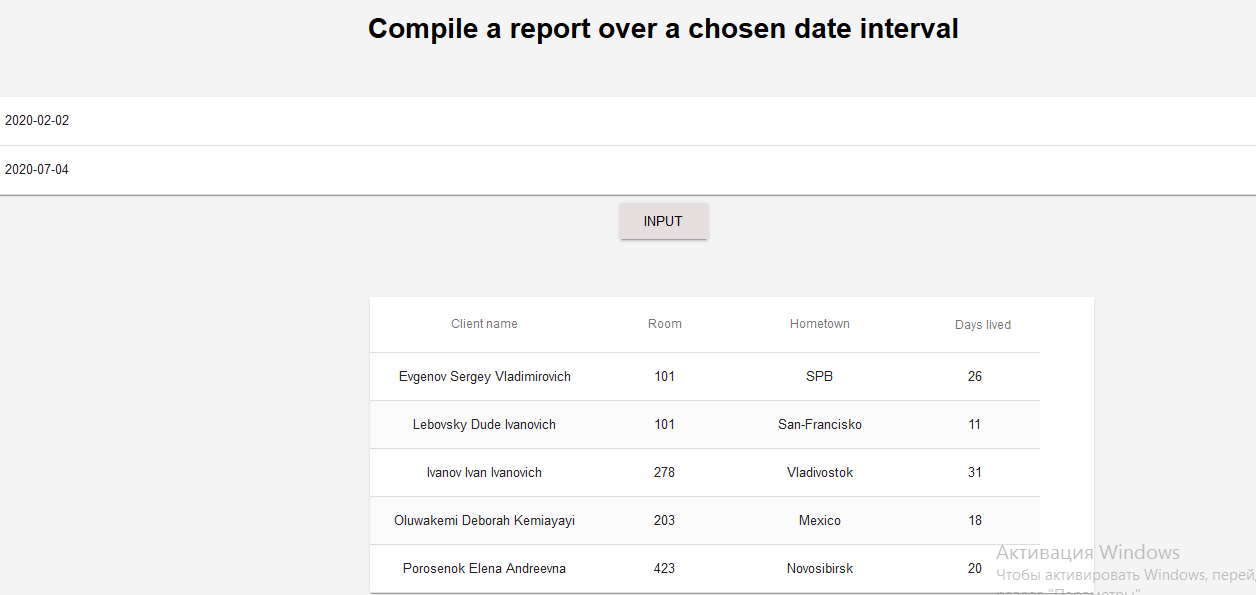


Рисунок 70 – Отображение компонента Report (часть 1)



Рисунок 70 – Отображение компонента Report (часть 2)

## Muse-UI

Во всех компонентах разработанного web-приложения в тэгах <template ...> были использованы тэги Muse-UI. Muse-Ui – это библиотека дизайна компонентов Vue.js [5].

## Выводы

В результате разработки клиентской части web-приложения были созданы следующие компоненты Vue.js:

1. Home,
2. Main,
3. Rooms,
4. Clients,
5. Journal,
6. Workers,
7. Worktable,
8. Login,
9. Request1,
10. Request2,
11. Request3,
12. Request4,
13. Request5,
14. Report.

Данные компоненты позволяют отобразить всё необходимое для web-приложения в заданной предметно области исходя из установленных в первой главе отчёта по данной курсовой работе функциональных требований. Дизайн всех данных компонентов сделан с помощью библиотеки Muse\_UI.

# ЗАКЛЮЧЕНИЕ

За время выполнения курсовой работы были выполнены все поставленные функциональные задачи.

Для реализации страницы сайта, связанной с администрированием при помощи различных возможностей, была использована конструкция с несколькими вкладками, что позволило уместить много функционала на одной странице.

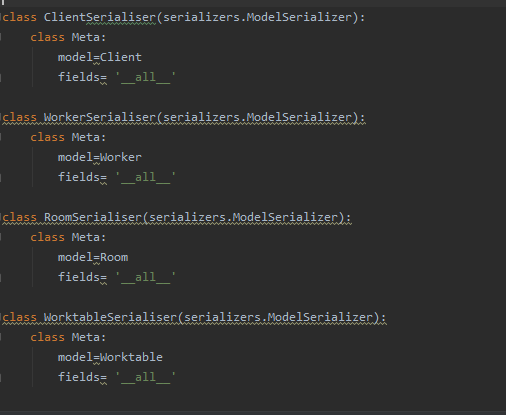
Также были созданы служебные страницы для авторизации на сайте, изменения записей о заселении и работе и служащих, а также добавления новых.

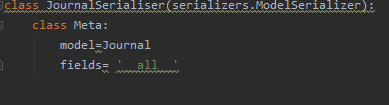
Полученное web-приложение уже готово к использованию, однако может быть развито в направлении увеличения удобства отображения.

# СПИСОК ЛИТЕРАТУРЫ

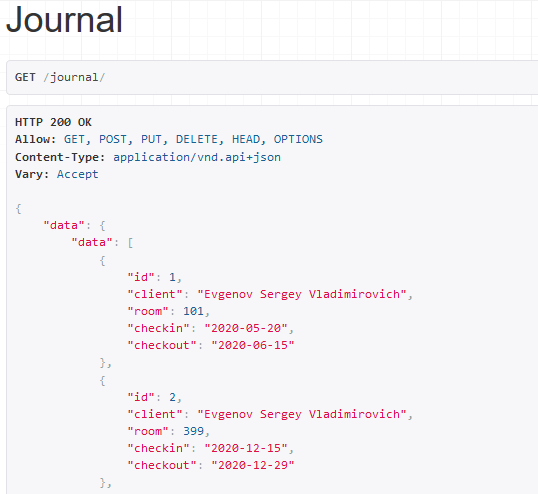
1. Django documentation // The Web framework for perfectionists with deadlines [Электронный ресурс] URL: <https://www.djangoproject.com/> (дата обращения: 20.06.2020)
2. Home // Django REST framework [Электронный ресурс] URL: <https://www.django-rest-framework.org/> (дата обращения: 20.06.2020)
3. Курылев А. Что такое Django REST Framework? // MkDev Personal [Электронный ресурс] URL: <https://mkdev.me/posts/chto-takoe-django-rest-framework> (дата обращения: 20.06.2020)
4. Introduction // Vue.js [Электронный ресурс] URL: <https://vuejs.org/v2/guide/> (дата обращения: 25.06.2020)
5. Muse UI [Электронный ресурс] URL: <https://muse-ui.org/#/en-US> (дата обращения: 25.06.2020)

# Приложение А. Сериализаторы





# Приложение В. Страницы Django REST Framework’а







# Приложение Г. Код компонента Main

<template>  
 <div>  
 <h1>Welcome!</h1>  
 <h2>Choose an action:</h2>  
 <mu-container>  
 <mu-tabs class="demo-text" :value.sync="active1" inverse color="black" text-color="rgba(0, 0, 0, .54)" indicator-color="#e6ddde" center>  
 <mu-tab>Clients</mu-tab>  
 <mu-tab>Workers</mu-tab>  
 <mu-tab>Worktable</mu-tab>  
 <mu-tab>Journal</mu-tab>  
 <mu-tab>Rooms</mu-tab>  
 <mu-tab>Report</mu-tab>  
 <mu-tab>Requests</mu-tab>  
 </mu-tabs>  
 <div v-if="active1 === 0">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goClients">Show all clients</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 1">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goWorkers">Show all workers</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 2">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goWorktable">Show the worktable</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 3">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goJournal">Show the reception journal</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 4">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goRooms">Show all rooms</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 5">  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goReport">Compile a report of a chosen date interval</mu-button>  
 </mu-container></div>  
 <div v-if="active1 === 6">  
 <br/>  
 <br/>  
<mu-container class="button-wrapper2">  
 <p>Choose an request you would like to perform:</p>  
 <mu-button flat textColor="#581845" @click="goRequest1">Clients that lived in a room during specified time</mu-button>  
 </mu-container>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goRequest2">Number of clients from a specific city</mu-button>  
 </mu-container>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goRequest3">Worker who tidied the specific room during a chosen day of the week</mu-button>  
 </mu-container>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goRequest4">What rooms are empty right now</mu-button>  
 </mu-container>  
 <mu-container class="button-wrapper2">  
 <mu-button flat textColor="#581845" @click="goRequest5">Clients who lived in the same time with a chosen client</mu-button>  
 </mu-container></div>  
 </mu-container></div>  
</template>  
  
<script>  
import $ from 'jquery'  
export default {  
 data() {  
 return {  
 active1: 0  
 };  
 },  
 name: "Main",  
 created() {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 })  
 },  
 methods: {  
 logout() {  
 ***sessionStorage***.removeItem("auth\_token")  
 ***window***.location = '/'  
 },  
 goRequest1() {  
 this.$router.push({name: "request1"})  
 },  
 goRequest2() {  
 this.$router.push({name: "request2"})  
 },  
 goRequest3() {  
 this.$router.push({name: "request3"})  
 },  
 goRequest4() {  
 this.$router.push({name: "request4"})  
 },  
 goRequest5() {  
 this.$router.push({name: "request5"})  
 },  
 goReport() {  
 this.$router.push({name: "report"})  
 },  
 goRooms() {  
 this.$router.push({name: "rooms"})  
 },  
 goClients() {  
 this.$router.push({name: "clients"})  
 },  
 goWorkers() {  
 this.$router.push({name: "workers"})  
 },  
 goJournal() {  
 this.$router.push({name: "journal"})  
 },  
 goWorktable() {  
 this.$router.push({name: "worktable"})  
 }  
 }  
 }  
</script>  
  
<style scoped>  
 .demo-text{  
 background:#64e867;  
 }  
 h1 {  
 margin-top: 50px;  
 font-size: 200%;  
 line-height: 100px;  
 color: #000000;  
 text-align: center;  
  
}.button-wrapper {  
 text-align: right;  
 .mu-button{font-size: 500%;  
 margin: 8px;  
 }  
}  
</style>

# Приложение Д. Код компонента Clients

<template>  
<div>  
 <mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>All-time clients list</h1>  
 <mu-container class="table-wrapper">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns" :data="list.data" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row.id }}</td>  
 <td class="is-center">{{ ***scope***.row.fio }}</td>  
 <td class="is-center">{{ ***scope***.row.hometown }}</td>  
 <td class="is-center">{{ ***scope***.row.passport }}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
 <br/>  
 <br/>  
  
 </div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: "Clients",  
 data(){  
 return{  
 fio:'',  
 hometown: '',  
 passport:'',  
 columns:[  
 { title: "Client ID", name: 'id', width: 50, align: 'center' },  
 { title: "Client full name", name: 'fio', width: 230, align: 'center' },  
 { title: "Client's hometown", name: 'hometown', width: 180, align: 'center' },  
 { title: "Passport number", name: 'passport', width: 180, align: 'center' },  
 ],  
 list:[{}],  
 options:''  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " +  
 ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadClients()  
 },  
 methods:{  
 loadClients() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/clients/',  
 type: 'GET',  
 success: (response) => {  
 this.list = response.data  
 }  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 }  
 }  
  
 }  
</script>  
  
<style >  
 h1 {  
 margin-top: 50px;  
 font-size: 200%;  
 line-height: 100px;  
 color: #000000;  
 font-family: 'Questrial', sans-serif;  
 text-align: center;  
}  
 .demo-divider-form {  
 font-size: 90%;  
 padding: 0 16px;  
 height: 30px;  
}  
.table-wrapper {  
 text-align: center;  
 margin-left: 360px;  
 width: 640px;  
}  
.panel-wrapper {  
 width: 640px;  
}  
.button-wrapper4 {  
 text-align: left;  
  
 .mu-button {  
 margin: 8px;  
 }  
  
 }  
</style>

# Приложение Е. Код компонента Rooms

<template>  
 <div>  
 <mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>Rooms of the hotel</h1>  
 <mu-container class="table-wrapper">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns" :data="list.data" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row.number }}</td>  
 <td class="is-center">{{ ***scope***.row.type }}</td>  
 <td class="is-center">{{ ***scope***.row.cost }}</td>  
 <td class="is-center">{{ ***scope***.row.phone }}</td>  
 <td class="is-center">{{ ***scope***.row.floor }}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
  
 <br/>  
 <br/>  
  
 </div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: "Rooms",  
 data(){  
 return{  
 sort: {  
 name: '',  
 order: 'asc'  
 },  
 number:'',  
 type: '',  
 cost:'',  
 phone:'',  
 floor:'',  
 columns:[  
 { title: "Room number", name: 'number', width: 100, align: 'center' ,sortable:true},  
 { title: "Room type", name: 'type', width: 150, align: 'center' ,sortable:true},  
 { title: "Daily cost", name: 'cost', width: 100, align: 'center' ,sortable:true},  
 { title: "Room telephone number", name: 'phone', width: 180, align: 'center' },  
 { title: "Floor", name: 'floor', width: 80, align: 'center',sortable:true },  
 ],  
 list:[{}],  
 options:''  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " +  
 ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadRooms()  
 },  
 methods:{  
 handleSortChange ({name, order}) {  
 this.list.data = this.list.data.sort(  
 (a, b) => order === 'asc' ? a[name] - b[name] : b[name] - a[name]);  
 },  
 loadRooms() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/rooms/',  
 type: 'GET',  
 success: (response) => {  
 this.list = response.data  
 }  
 })  
 },  
 goHome() {  
 this.$router.push({name: "main"})  
 }  
 }  
 }  
</script>  
  
<style scoped>  
 h1 {  
 margin-top: 50px;  
 font-size: 200%;  
 line-height: 100px;  
 color: #000000;  
 font-family: 'Questrial', sans-serif;  
 text-align: center;  
}  
 .demo-divider-form {  
 font-size: 90%;  
 padding: 0 16px;  
 height: 30px;  
}  
.table-wrapper {  
 text-align: center;  
 margin-left: 360px;  
 width: 640px;  
}  
.panel-wrapper {  
 width: 640px;  
}  
.button-wrapper4 {  
 text-align: left;  
 .mu-button{  
 margin: 8px;  
 }  
}  
</style>

# Приложение Ж. Код компонента Worktable

<template>  
<div><mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>Worktable of the hotel</h1>  
 <mu-container class="table-wrapper">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns" :data="list.data" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row.id }}</td>  
 <td class="is-center">{{ ***scope***.row.worker }}</td>  
 <td class="is-center">{{ ***scope***.row.floor }}</td>  
 <td class="is-center">{{ ***scope***.row.weekday }}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
 <br/>  
 <h2>To delete an assignment of work, enter its id below:</h2>  
 <input v-model="id" type="text" placeholder="Record ID"/>  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="deleteWorktable">Delete</mu-button>  
 </mu-container>  
 <br/> <h2>To add a new assignment, fill out the form below:</h2>  
 <br/>  
 <mu-select label="Worker name:" class="options-wrapper" v-model="worker">  
 <mu-option v-for="***worker***,***index*** in workers" :key="***worker***.fio" :label="***worker***.fio" :value="***worker***.fio"></mu-option>  
 </mu-select>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="floor" placeholder="Floor (1-4)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="weekday" placeholder="Week day (e.g.Monday)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <br/>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="addWorktable">Add</mu-button>  
 </mu-container>  
 <br/> <h2>To edit an existing assignment, enter its ID first:</h2>  
 <br/><input v-model="wt\_id" type="text" placeholder=""/>  
 <br/>  
 <br/>  
 <mu-container v-if="!options" class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="editWorktable">Input</mu-button>  
 </mu-container>  
 <br/>  
 <mu-container v-if="options">  
 <h3>Now, edit the data </h3>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="wt\_floor" placeholder="Floor (1-4)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 </mu-container>  
 <mu-container v-if="options">  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="wt\_weekday" placeholder="Day of the week" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 </mu-container>  
 <br v-if="options"/>  
 <p v-if="options">Your worker: &nbsp; &nbsp; &nbsp; &nbsp;  
 <mu-select v-if="options" class="options-wrapper" v-model="wt\_worker">  
 <mu-option v-for="***worker*** in workers" :key="***worker***.fio" :label="***worker***.fio" :value="***worker***.fio"></mu-option>  
 </mu-select>  
 </p>  
 <mu-container v-if="options" class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="submitWorktable">Submit changes</mu-button>  
 </mu-container>  
  
</div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: "Worktable",  
 data(){  
 return{  
 id:'',  
 workers:'',  
 wt\_id:'',  
 worker: '',  
 floor:'',  
 weekday:'',  
 wt\_worker:'',  
 wt\_floor:'',  
 wt\_weekday:'',  
 columns:[  
 { title: "ID", name: 'id', width: 130, align: 'center' },  
 { title: "Worker", name: 'fio', width: 260, align: 'center' },  
 { title: "Floor", name: 'floor', width: 130, align: 'center',sortable:true },  
 { title: "Day of the week", name: 'weekday', width: 180, align: 'center' },  
 ],  
 list:[{}],  
 options:''  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadWorktable()  
 this.loadWorkers()  
 },  
 methods:{  
 loadWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktable/',  
 type: 'GET',  
 success: (response) => {  
 this.list = response.data  
 }  
 })},  
 loadWorkers() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/workers/',  
 type: 'GET',  
 success: (response) => {  
 this.workers = response.data.data  
 }  
 })  
 },  
 deleteWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktable/',  
 type: 'DELETE',  
 data: {  
 id: this.id,  
 },  
 success: (response) => {  
 this.loadWorktable()  
 },  
 error: (response) => {  
 if (response.status === 404) {  
 alert('No assignment was found')  
 }  
 }  
 })  
 },  
 addWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktable/',  
 type: 'POST',  
 data: {  
 id: this.id,  
 worker:this.worker,  
 floor:this.floor,  
 weekday:this.weekday,  
 },  
 success: (response) => {  
 this.loadWorktable()  
 }  
 })  
 },  
 editWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktableq/',  
 type: 'GET',  
 data: {  
 id: this.wt\_id  
 },  
 success: (response) => {  
 this.options = response.data  
 this.wt\_worker = response.data.data.worker  
 this.wt\_floor = response.data.data.floor  
 this.wt\_weekday = response.data.data.weekday  
 },  
 error: (response) => {  
 if (response.status === 404) {  
 alert('No assignment was found')  
 }  
 }  
 })  
 }, submitWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktable/',  
 type: 'PUT',  
 data: {  
 id: this.wt\_id,  
 worker: this.wt\_worker,  
 floor: this.wt\_floor,  
 weekday: this.wt\_weekday,  
 },  
 success: (response) => {  
 this.loadWorktable()  
 this.options = ''  
 }  
 })  
 },handleSortChange ({name, order}) {  
 this.list.data = this.list.data.sort((a, b) => order === 'asc' ? a[name] - b[name] : b[name] - a[name]);  
 },goHome() {  
 this.$router.push({name: "main"})  
 }  
 },}  
</script>  
  
<style>  
.table-wrapper {  
 text-align: center;  
 margin-left: 313px;  
 width: 740px;  
}  
.panel-wrapper {  
 width: 740px;  
}  
.button-wrapper4 {  
 text-align: left;  
 .mu-button{  
 margin: 8px;  
 }  
}  
</style>

# Приложение З. Код компонента Journal

<template>  
<div><mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>The journal of all records</h1>  
 <mu-container class="table-wrapper">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns" :data="list.data" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row.id }}</td>  
 <td class="is-center">{{ ***scope***.row.room }}</td>  
 <td class="is-center">{{ ***scope***.row.client }}</td>  
 <td class="is-center">{{ ***scope***.row.checkin }}</td>  
 <td class="is-center">{{ ***scope***.row.checkout }}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
 <br/>  
 <h2>To delete a record of a client's living, enter its id below:</h2>  
 <input v-model="id" type="text" placeholder="Record ID"/>  
 <br/>  
 <br/>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="deleteJournal">Delete</mu-button>  
 </mu-container>  
 <br/> <h2>To add a new assignment, fill out the form below:</h2>  
 <br/>  
 <mu-select label="Room number:" class="options-wrapper" v-model="room">  
 <mu-option v-for="***room***,***index*** in rooms" :key="***room***.number" :label="***room***.number" :value="***room***.number"></mu-option>  
 </mu-select>  
 <mu-select label="Client name:" class="options-wrapper" v-model="client">  
 <mu-option v-for="***room***,***index*** in clients" :key="***room***.fio" :label="***room***.fio" :value="***room***.fio"></mu-option>  
 </mu-select>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="checkin" placeholder="Check-in date (YYYY-MM-DD format)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="checkout" placeholder="Check-out date (YYYY-MM-DD format)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <br/>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="addJournal">Add</mu-button>  
 </mu-container>  
 <br/> <h2>To edit an existing record, enter its ID first:</h2>  
 <br/><input v-model="jn\_id" type="text" placeholder="Journal ID"/>  
 <br/>  
 <br/>  
 <mu-container v-if="!options" class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="editJournal">Input</mu-button>  
 </mu-container>  
 <br/>  
 <mu-container v-if="options">  
 <h3>Now, edit the data </h3>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="jn\_checkin" placeholder="" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 </mu-container>  
 <mu-container v-if="options">  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="jn\_checkout" placeholder="" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 </mu-container>  
 <br v-if="options"/>  
 <p v-if="options">Client: &nbsp; &nbsp; &nbsp; &nbsp;  
 <mu-select v-if="options" class="options-wrapper" v-model="jn\_client">  
 <mu-option v-for="***worker*** in clients" :key="***worker***.fio" :label="***worker***.fio" :value="***worker***.fio"></mu-option>  
 </mu-select>  
 </p>  
 <p v-if="options">Room:  
 <mu-select v-if="options" class="options-wrapper" v-model="jn\_room">  
 <mu-option v-for="***worker*** in rooms" :key="***worker***.number" :label="***worker***.number" :value="***worker***.number"></mu-option>  
 </mu-select>  
 </p>  
 <mu-container v-if="options" class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="submitJournal">Submit changes</mu-button>  
 </mu-container>  
  
</div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: "Journal",  
 data(){  
 return{  
 id:'',  
 client: '',  
 room:'',  
 checkin:'',  
 checkout:'',  
 jn\_id:'',  
 jn\_client:'',  
 jn\_room:'',  
 jn\_checkin:'',  
 jn\_checkout:'',  
 clients:'',  
 rooms:'',  
 columns:[  
 { title: "ID", name: 'id', width: 100, align: 'center' },  
 { title: "Room number", name: 'room', width: 100, align: 'center',sortable:true },  
 { title: "Client", name: 'client', width: 280, align: 'center' },  
 { title: "Check-in date", name: 'checkin', width: 180, align: 'center', sortable:true},  
 { title: "Check-out date", name: 'checkout', width: 180, align: 'center',sortable:true },  
 ],  
 list:[{}],  
 options:''  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " +  
 ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadJournal()  
 this.loadClients()  
 this.loadRooms()  
 },  
 methods:{  
 loadJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'GET',  
 success: (response) => {  
 this.list = response.data  
 }  
 })},  
 loadClients() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/clients/',  
 type: 'GET',  
 success: (response) => {  
 this.clients = response.data.data  
 }  
 })},  
 loadRooms() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/rooms/',  
 type: 'GET',  
 success: (response) => {  
 this.rooms = response.data.data  
 }})},  
  
 deleteJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'DELETE',  
 data: {  
 id: this.id,  
 },  
 success: (response) => {  
 this.loadJournal()  
 },  
 error: (response) => {  
 if (response.status === 404) {  
 alert('No record was found')  
 }  
 }  
 })  
 },  
 addJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'POST',  
 data: {  
 id: this.id,  
 client:this.client,  
 room:this.room,  
 checkin:this.checkin,  
 checkout:this.checkout,  
 },  
 success: (response) => {  
 this.loadJournal()  
 }  
 })  
 },  
 editJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journalq/',  
 type: 'GET',  
 data: {  
 id: this.jn\_id  
 },  
 success: (response) => {  
 this.options = response.data  
 this.jn\_client = response.data.data.client  
 this.jn\_room = response.data.data.room  
 this.jn\_checkin = response.data.data.checkin  
 this.jn\_checkout = response.data.data.checkout  
 },  
 error: (response) => {  
 if (response.status === 404) {  
 alert('No record was found')  
 }  
 }  
 })  
 }, handleSortChange ({name, order}) {  
 this.list.data = this.list.data.sort((a, b) => order === 'asc' ? a[name] - b[name] : b[name] - a[name]);  
 }, submitJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'PUT',  
 data: {  
 id: this.jn\_id,  
 client: this.jn\_client,  
 room: this.jn\_room,  
 checkin: this.jn\_checkin,  
 checkout: this.jn\_checkout,  
 },  
 success: (response) => {  
 this.loadJournal()  
 this.options = ''  
 }  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 }  
 }}  
</script>  
  
<style >  
 .table-wrapper {  
 text-align: center;  
 width: 860px;  
}  
p{margin-left: 10px}  
.button-wrapper4 {  
 text-align: left;}  
  
  
</style>

# Приложение И. Код компонента Request1

<template>  
<div><mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>Clients that lived in a room during specified time</h1>  
 <p>Room: &nbsp; &nbsp; &nbsp; &nbsp;  
 <mu-select class="options-wrapper" v-model="room">  
 <mu-option v-for="***worker***,***index*** in rooms" :key="***worker***.number" :label="***worker***.number" :value="***worker***.number"></mu-option>  
 </mu-select>  
 </p>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="firstdate" placeholder="First date (YYYY-MM-DD)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="seconddate" placeholder="Second date (YYYY-MM-DD)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Input</mu-button>  
 </mu-container>  
 <mu-container class="asa" v-if="list">  
 <h2>Query results returned:</h2>  
 <div v-for="***answer*** in list">  
 <mu-container style="text-align: center">  
 <mu-paper class="demo-paper" :z-depth="1" style="padding: 15px">{{ ***answer***.attributes.fio}}, from {{ ***answer***.attributes.hometown}}, passport id is {{ ***answer***.attributes.passport}}</mu-paper>  
 </mu-container>  
 </div></mu-container>  
</div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: 'Request1',  
 data () {  
 return {  
 labelPosition: 'top',  
 room: '',  
 firstdate:'',  
 seconddate:'',  
 rooms:[],  
 records:[],  
 list: ***NaN***,  
 columns:[  
 { title: "Client", name: 'client', width: 200, align: 'center' },  
 { title: "Hometown", name: 'hometown', width: 400, align: 'center' },  
 { title: "Passport", name: 'passport', width: 200, align: 'center' },  
 ],  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadJournal()  
 this.loadRooms()  
 },  
 methods: {  
 loadJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'GET',  
 success: (response) => {  
 this.records = response.data  
 }  
 })  
 },  
 loadRooms() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/rooms/',  
 type: 'GET',  
 success: (response) => {  
 this.rooms = response.data.data  
 }  
 })  
 },  
 loadAnswer() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/request1/',  
 type: 'GET',  
 data: {  
 room: this.room,  
 firstdate:this.firstdate,  
 seconddate: this.seconddate  
 },  
 success: (response) => {  
 this.list = response.data},  
 error: (response) => {  
 alert("somethings wrong")}  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 },  
 }}  
</script>  
  
<style>  
 .table-wrapper {  
 text-align: center;  
 margin-left: 260px;  
 width: 900px;  
}  
.button-wrapper4 {  
 text-align: left;  
 .mu-button{  
 margin: 8px;  
 }}  
 .asa{  
 margin-bottom: 100px;  
 }  
</style>

# Приложение К. Код компонента Report

<template><div>  
 <mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>Compile a report over a chosen date interval</h1>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="firstdate" placeholder="First date (YYYY-MM-DD)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="seconddate" placeholder="Second date (YYYY-MM-DD)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Input</mu-button>  
 </mu-container>  
 <mu-container v-if="ans">  
  
 <mu-container class="table-wrapper">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns" :data="ans.clients" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row.client\_fio }}</td>  
 <td class="is-center">{{ ***scope***.row.client\_room}}</td>  
 <td class="is-center">{{ ***scope***.row.client\_home }}</td>  
 <td class="is-center">{{ ***scope***.row.client\_days }}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
 <h2>Over this date, the hotel accustomed {{ans.totalclients.toString()}} clients, and in total have generated {{ans.totalcost.toString()}} value</h2>  
 <h4>More detailed information:</h4>  
 <mu-container class="table-wrapper2">  
 <mu-row gutter>  
 <mu-col >  
 <mu-paper :z-depth="1">  
 <mu-data-table stripe :columns="columns2" :data="ans.per\_room" :sort.sync="sort" @sort-change="handleSortChange">  
 <template slot-scope="***scope***">  
 <td class="is-center">{{ ***scope***.row[0] }}</td>  
 <td class="is-center">{{ ***scope***.row[1]}}</td>  
 </template>  
 </mu-data-table>  
 </mu-paper>  
 </mu-col>  
 </mu-row>  
 </mu-container>  
 <ul><li style="text-align: left" v-for="***i*** in ans.per\_room.per\_room ">  
 {{***i***}}  
 </li>  
 </ul>  
  
 </mu-container>  
</div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: 'Report',  
 data () {  
 return {  
 labelPosition: 'top',  
 firstdate:'',  
 seconddate:'',  
 ans:***NaN***,  
 columns:[  
 { title: "Client name", name: 'client\_fio', width: 230, align: 'center' },  
 { title: "Room", name: 'client\_room', width: 130, align: 'center' },  
 { title: "Hometown", name: 'client\_home', width: 180, align: 'center' },  
 { title: "Days lived", name: 'client\_days', width: 130, align: 'center',sortable:true },  
 ],  
 columns2:[  
 { title: "Room", name: 'room', width: 130, align: 'center',sortable :true },  
 { title: "Overall income", name: 'cost', width: 230, align: 'center' ,sortable:true},]  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadJournal()  
 this.loadRooms()  
 },  
 methods: {  
 handleSortChange ({name, order}) {  
 this.list.data = this.list.data.sort((a, b) => order === 'asc' ? a[name] - b[name] : b[name] - a[name]);  
 },  
 loadClients() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/clients/',  
 type: 'GET',  
 success: (response) => {  
 this.clients = response.data  
 }  
 })  
 },  
 loadAnswer() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/report/',  
 type: 'GET',  
 data: {  
 firstdate: this.firstdate,  
 seconddate:this.seconddate  
 },  
 success: (response) => {  
 this.ans = response.data},  
 error: (response) => {  
 alert("somethings wrong")}  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 },  
 }}  
</script>  
  
<style >  
.button-wrapper4 {  
 text-align: left;  
 .mu-button{  
  
 }}  
.table-wrapper {  
 text-align: center;  
 align-self: center;  
 width: 740px;  
}  
.table-wrapper2 {  
 text-align: center;  
 align-self: center;  
 width: 400px;  
 margin-bottom: 60px;  
}  
 .asa{  
 margin-bottom: 100px;  
 }  
</style>

# Приложение Л. Код компонента Request2

<template>  
 <div>  
 <mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h1>Number of clients from a specified city</h1>  
<mu-paper :z-depth="1">  
 <mu-text-field v-model="hometown" placeholder="Enter the city name" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Input</mu-button>  
 </mu-container>  
 <mu-container class="asa" v-if="ans">  
 {{ans.data.thing}} client(s) from {{hometown}}</mu-container>  
 </div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: 'Request2',  
 data () {  
 return {  
 labelPosition: 'top',  
 hometown: '',  
 clients:[],  
 ans:***NaN***,  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadJournal()  
 this.loadRooms()  
 },  
 methods: {  
 loadClients() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/clients/',  
 type: 'GET',  
 success: (response) => {  
 this.clients = response.data  
 }  
 })  
 },  
 loadAnswer() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/request2/',  
 type: 'GET',  
 data: {  
 hometown: this.hometown,  
 },  
 success: (response) => {  
 this.ans = response.data},  
 error: (response) => {  
 alert("somethings wrong")}  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 },  
 }}  
</script>  
  
<style >  
 .button-wrapper4 {  
 text-align: left;  
 .mu-button{  
 margin: 8px;  
 }}  
 .asa{  
 font-size: 200%;  
 }  
  
</style>

# Приложение М. Код компонента Request3

<template><div>  
 <mu-container class="button-wrapper4" style="margin-bottom: 50px ">  
 <mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button>  
 </mu-container>  
 <h2>Worker who tidied the specific room during a chosen day of the week</h2>  
<p>Room: &nbsp; &nbsp; &nbsp; &nbsp;  
 <mu-select class="options-wrapper" v-model="room">  
 <mu-option v-for="***room***,***index*** in rooms" :key="***room***.number" :label="***room***.number" :value="***room***.number"></mu-option>  
 </mu-select>  
 </p>  
 <mu-paper :z-depth="1">  
 <mu-text-field v-model="weekday" placeholder="Day of the week (e.g. Monday)" solo full-width class="demo-divider-form"></mu-text-field>  
 <mu-divider></mu-divider>  
 </mu-paper>  
 <mu-container class="button-wrapper3" style="margin-bottom: 50px ">  
 <mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Input</mu-button>  
 </mu-container>  
 <mu-container class="asa" v-if="list">  
 <li v-for="***thing*** in list">  
 The room was being cleaned by {{***thing***.attributes.fio}}  
 <br/>  
 </li>  
 </mu-container></div>  
</template>  
  
<script>  
 import $ from 'jquery'  
 export default {  
 name: 'Request3',  
 data () {  
 return {  
 labelPosition: 'top',  
 weekday:'',  
 id:'',  
 room:'',  
 rooms:[],  
 records:[],  
 list:***NaN***,  
 }  
 },  
 created () {  
 $.ajaxSetup({  
 headers: {'Authorization': "Token " + ***sessionStorage***.getItem('auth\_token')},  
 }),  
 this.loadJournal()  
 this.loadRooms()  
 this.loadWorkers()  
 this.loadWorktable()  
 },  
 methods: {  
 loadJournal() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/journal/',  
 type: 'GET',  
 success: (response) => {  
 this.records = response.data  
 }  
 })  
 },  
 loadRooms() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/rooms/',  
 type: 'GET',  
 success: (response) => {  
 this.rooms = response.data.data  
 }  
 })  
 },  
 loadWorkers() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/workers/',  
 type: 'GET',  
 success: (response) => {  
 this.records = response.data  
 }  
 })  
 },  
 loadWorktable() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/worktable/',  
 type: 'GET',  
 success: (response) => {  
 this.records = response.data  
 }  
 })  
 },  
 loadAnswer() {  
 $.ajax({  
 url: 'http://127.0.0.1:8000/request3/',  
 type: 'GET',  
 data: {  
 room: this.room,  
 weekday:this.weekday,  
 },  
 success: (response) => {  
 this.list = response.data},  
 error: (response) => {  
 alert("somethings wrong")}  
 })  
 },goHome() {  
 this.$router.push({name: "main"})  
 },  
 }}  
</script>  
  
<style >  
 .button-wrapper4 {  
 text-align: left;  
 .mu-button{  
 margin: 8px;  
 }}  
 .asa{  
 font-size: 170%;  
 margin-bottom: 100px;  
 }  
  
</style>

# Приложение Н. Код компонента Request4

<template>

<div>

<mu-container class="button-wrapper4" style="margin-bottom: 50px ">

<mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button></mu-container>

<h2>Find out what rooms are empty right now</h2>

<mu-container class="button-wrapper3" style="margin-bottom: 50px ">

<mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Find out</mu-button>

</mu-container>

<mu-container v-if="ans">

<h2>Rooms that are empty right now:</h2>

<mu-container v-for="room in ans">

<mu-container style="text-align: center; font-size: 150%">

<mu-paper class="demo-paper" :z-depth="1" style="padding: 15px">{{ room.attributes.number}}</mu-paper>

</mu-container>

</mu-container>

</mu-container></div>

</template>

<script>

import $ from 'jquery'

export default {

name: 'Request4',

data () {

return {

labelPosition: 'top',

hometown: '',

clients:[],

ans:NaN,

}

},

created () {

$.ajaxSetup({

headers: {'Authorization': "Token " + sessionStorage.getItem('auth\_token')},

}),

this.loadJournal()

this.loadRooms()

},

methods: {

loadClients() {

$.ajax({

url: 'http://127.0.0.1:8000/clients/',

type: 'GET',

success: (response) => {

this.clients = response.data

}

})

},

loadAnswer() {

$.ajax({

url: 'http://127.0.0.1:8000/request4/',

type: 'GET',

data: {

hometown: this.hometown,

},

success: (response) => {

this.ans = response.data},

error: (response) => {

alert("somethings wrong")}

})

},goHome() {

this.$router.push({name: "main"})

},

}}

</script>

<style >

.button-wrapper4 {

text-align: left;

.mu-button{

margin: 8px;

}}

.asa{

margin-bottom: 100px;

}

</style>

# Приложение О. Код компонента Request5

<template>

<div> <mu-container class="button-wrapper4" style="margin-bottom: 50px ">

<mu-button color="#2065f0" textColor="white" @click="goHome">Home</mu-button></mu-container>

<h2>Clients who lived in the same time with a chosen client</h2>

<p>Choose a client: &nbsp; &nbsp; &nbsp; &nbsp;

<mu-select class="options-wrapper" v-model="client">

<mu-option v-for="worker,index in clients" :key="worker.fio" :label="worker.fio" :value="worker.fio"></mu-option>

</mu-select>

</p>

<mu-paper :z-depth="1">

<mu-text-field v-model="firstdate" placeholder="First date" solo full-width class="demo-divider-form"></mu-text-field>

<mu-divider></mu-divider>

</mu-paper>

<mu-paper :z-depth="1">

<mu-text-field v-model="seconddate" placeholder="Second date" solo full-width class="demo-divider-form"></mu-text-field>

<mu-divider></mu-divider>

</mu-paper>

<mu-container class="button-wrapper3" style="margin-bottom: 50px ">

<mu-button color="#e6ddde" textColor="black" @click="loadAnswer">Input</mu-button>

</mu-container>

<mu-container class="asa" v-if="list">

<h2>Query results returned:</h2>

<div v-for="answer in list">

<mu-container style="text-align: center">

<mu-paper class="demo-paper" :z-depth="1" style="padding: 15px">{{ answer.attributes.fio}}, from {{ answer.attributes.hometown}}</mu-paper>

</mu-container>

</div>

</mu-container>

</div>

</template>

<script>

import $ from 'jquery'

export default {

name: 'Request5',

data () {

return {

labelPosition: 'top',

weekday:'',

client:'',

firstdate:'',

seconddate:'',

clients:[],

records:[],

list:NaN,

}

},

created () {

$.ajaxSetup({

headers: {'Authorization': "Token " + sessionStorage.getItem('auth\_token')},

}),

this.loadJournal()

this.loadClient()

},

methods: {

loadJournal() {

$.ajax({

url: 'http://127.0.0.1:8000/journal/',

type: 'GET',

success: (response) => {

this.records = response.data

}

})

},

loadClient() {

$.ajax({

url: 'http://127.0.0.1:8000/clients/',

type: 'GET',

success: (response) => {

this.clients = response.data.data

}

})

},

loadAnswer() {

$.ajax({

url: 'http://127.0.0.1:8000/request5/',

type: 'GET',

data: {

client: this.client,

firstdate:this.firstdate,

seconddate:this.seconddate

},

success: (response) => {

this.list = response.data},

error: (response) => {

alert("somethings wrong")}

})

},goHome() {

this.$router.push({name: "main"})

},

}}

</script>

<style >

.button-wrapper4 {

text-align: left;

.mu-button{

margin: 8px;

}}

.asa{

margin-bottom: 100px;

}

</style>