Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(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

о курсовой работе по дисциплине «Основы Web-программирования»

Тема задания: программная система, позволяющая отслеживать распределение по почтовым отделениям газет, печатающихся в типографиях города

Обучающийся Катаева Вероника Алексеевна, группа К3242

Руководитель курсовой работы: Говоров А.И., ассистент факультета инфокоммуникационных технологий Университета ИТМО

Оценка курсовой работы ____

Подписи членов комиссии:

_____ Говоров А. И.
(подпись)

_____ Чунаев А. В.
(подпись)

_____ Антонов М. Б.
(подпись)

Дата 04.07.2020

Санкт-Петербург
2020

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

AJAX (Asynchronous Javascript And Xml) – технология обращения к серверу без перезагрузки страницы.

API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

Backend – программно-аппаратная часть сервиса.

DELETE – HTTP метод, удаляющий указанный ресурс.

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC (Model-View-Controller, «Модель-Представление-Контроллер»).

Django REST Framework – это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.

Frontend – клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса.

GET – HTTP метод, позволяющий запрашивать представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript.

Muse-UI – фреймворк для создания интерфейсов, библиотека компонентов, основанная на Vue.js.

POST – HTTP метод, использующий для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.

PostgreSQL – свободная объектно-реляционная система управления базами данных.

PUT – HTTP метод, заменяющий все текущие представления ресурса данными запроса.

URL (Uniform Resource Locator) – унифицированный указатель ресурса.

Vue.js – это прогрессивный фреймворк для создания пользовательских интерфейсов.

Атрибут – это информационное отображение свойств сущности.

Контейнеризация – метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

Оркестрация – это координация взаимодействия нескольких контейнеров.

Связь один-ко-многим – в типе связей один ко многим одной записи первой таблицы соответствует несколько записей в другой таблице.

Сериализация – это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.

Сущность – это любой объект в базе данных, который можно выделить исходя из сути предметной области, для которой разрабатывается эта база данных.

Хук жизненного цикла – функции, с помощью которых можно выполнять код на определённых этапах.

Эндпоинт – (endpoint – конечная точка) – обращение к маршруту отдельным HTTP методом. Эндпоинт выполняет конкретную задачу, принимает параметры и возвращает данные клиенту.

ВВЕДЕНИЕ

Цель работы – создание программной системы, позволяющей отслеживать распределение по почтовым отделениям газет, печатающихся в типографиях города. Созданный продукт должен быть прост в использовании, отличаться удобным и понятным для любого человека интерфейсом. Проект также должен способствовать решению задач, с которыми сталкиваются сотрудники почтовых отделений в своей работе. От системы требуется работать быстро и не использовать большой объем вычислительный ресурсов, поскольку она взаимодействует с пользователем, который ожидает мгновенного результата.

Целевой аудиторией реализованного проекта являются сотрудники почтовых отделений, которые занимаются контролем количества поступающих газет и их распределением.

Задачи, требующие решения:

- ознакомление с требованиями, предъявляемыми к системе, и заданиями, выполнение которых поможет в работе сотрудникам почтовых отделений;
- выбор технологий и инструментов для реализации системы, которые бы совмещались между собой и отвечали требованиям эффективности;
- проектирование модели базы данных в соответствие с требованиями;
- проектирование архитектуры системы, которая бы отвечала всем требованиям и выполняла свою функцию оптимально, включающая разработку клиентской и серверной части системы;
- программирование системы;
- тестирование системы для выявления ее недостатков. Задача включает в себя тестирование различных компонентов системы, тестирование работы системы при различных данных;
- формирование планов по улучшению/развитию системы на основе оценки качества работы реализованных компонентов.

Таким образом, разработанная система послужит для сотрудников почтовых отделений актуальным инструментом, который позволит упростить и ускорить процесс выполнения рутинных задач.

В первой главе проведено описание системы. Описание включает в себя задачи, которые необходимо выполнять сотруднику, информацию, которая может понадобиться работнику, а также, полное описание компонентов системы.

Во второй главе описан процесс проектирования и разработки системы. В первом пункте содержится обоснование выбора технологий и инструментов для реализации системы. В следующем пункте приведена разработка модели данных, а также построенная на этой модели схема, были детально описаны сущности, их атрибуты и связи между ними. В третьем пункте определены основные интерфейсы системы и пояснена концепция дизайна. В последнем пункте непосредственно описана разработка архитектуры системы с описанием взаимодействия выбранных технологий.

В третьей главе представлено описание процесса программирования системы. В первом пункте главы приведены краткие описания моделей, сериализаторов, представлений, URL-адресов проекта. Во втором пункте объяснена структура Vue.js компонентов, взаимодействие клиентской и серверной частей, контейнеризация и оркестрация проекта.

В заключении приводится краткая характеристика проделанной работы, описание достигнутых целей, обозначение дальнейших перспектив развития проекта.

В приложении указаны версии использованных модулей и программ.

1. СИСТЕМА ГАЗЕТ

1.1. Описание системы

Система должна обеспечивать хранение, просмотр и изменение сведений о газетах, почтовых отделениях, получающих газеты, и о типографиях, выпускающих газеты. Сведения о газетах включают в себя: название газеты, индекс издания, фамилию, имя и отчество редактора, цену экземпляра газеты. Цены могут меняться. Возможно появление новых газет и изменение индекса существующего издания. Для типографий указываются их названия и адреса.

В типографии разными тиражами печатаются газеты нескольких наименований. Типография может быть закрыта, тогда необходимо скорректировать работу других типографий с учетом потребностей почтовых отделений в газетах.

Почтовое отделение имеет номер и адрес. На каждое почтовое отделение поступают в определенных количествах газеты разных наименований, причем часть экземпляров одной и той же газеты может быть напечатана в одной типографии, а часть – в другой.

Пользователям системы может потребоваться следующая информация:

– По каким адресам печатаются газеты данного наименования?

– Фамилия редактора газеты, которая печатается в указанной типографии самым большим тиражом?

– На какие почтовые отделения (адреса) поступает газета, имеющая цену, больше указанной?

– Куда поступает данная газета, печатающаяся по данному адресу.

Необходимо предусмотреть возможность выдачи справки об индексе и цене указанной газеты и отчета о работе типографий с почтовыми отделениями города. Отчет должен содержать по каждой типографии следующие сведения: общее количество печатающихся в типографии газет, количество газет каждого наименования, какие газеты и в каком количестве типография отправляет в каждое почтовое отделение.

1.2. Выводы

Таким образом, были поставлены задачи, которые должна помогать решать система. Также описаны компоненты системы, которые послужат основой для разработки базы и модели данных. Были обозначены запросы, ответы на которые дают информацию, являющейся важной для сотрудников почтовых отделений. Необходимым условием работы

системы является возможность получения отчета о работе типографии и справки об индексе и цене указанной газеты.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Выбор технологий и инструментов для реализации системы

На основе анализа компонентов и функционала системы были обозначены технологии и инструменты для реализации системы:

– Django [1] для написания веб-приложения;

– Django REST Framework [2] для создания API;

– Vue.js [3] и Muse-UI [4] для создания для пользовательских интерфейсов;

– PostgreSQL [5] для управления базой данных;

– Docker [6] для контейнеризации и оркестрации.

2.2. Разработка модели данных

Модель была разработана согласно описанию системы в главе 1 (рис. 1).

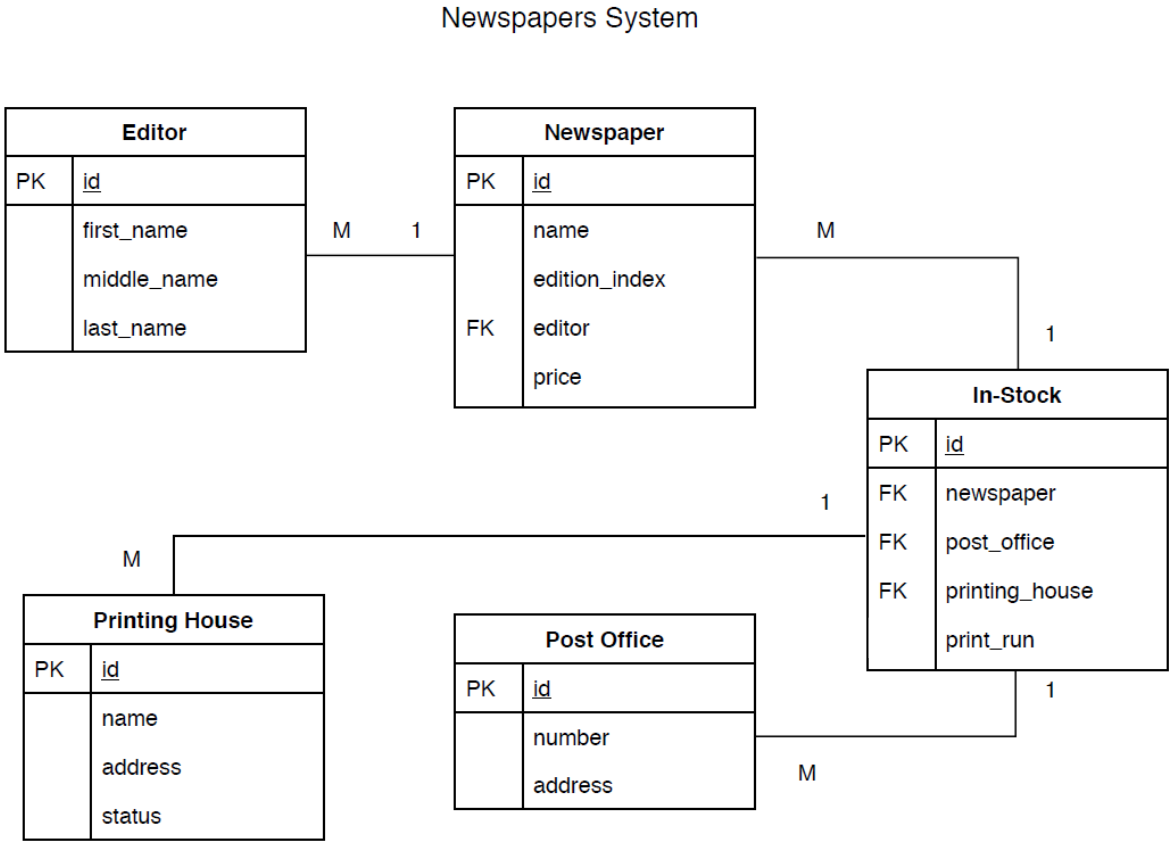

Рисунок 1 – Модель базы данных системы газет

Таким образом, модель состоит из следующих сущностей: *Редактор*, *Газета*, *Типография*, *Почтовое отделение*, *Склад*.

Атрибуты *Редактора*:

– ID (первичный ключ);

– имя;

– среднее имя;

– фамилия.

Атрибуты *Газеты*:

– ID (первичный ключ);

– название;

– индекс издания;

– редактор (внешний ключ);

– цена.

Атрибуты *Типографии*:

– ID (первичный ключ);

– название;

– адрес;

– статус (рабочая или закрытая).

Атрибуты *Почтового отделения*:

– ID (первичный ключ);

– номер;

– адрес.

Атрибуты *Склада*:

– ID (первичный ключ);

– газета (внешний ключ);

– почтовое отделение (внешний ключ);

– типография (внешний ключ);

– тираж.

Сущность *Наличие* имеет связи один-ко-многим с сущностями *Газета*, *Почтовое отделение*, *Типография*. Сущность *Газета* имеет связь один-ко-многим с сущностью *Редактор*.

2.3. Определение основных интерфейсов системы и концепции дизайна

Система имеет начальную страницу, на которой расположено название системы и кнопка входа (рис. 2).

Рисунок 2 – Начальная страница системы газет

После нажатия на данную кнопку перед пользователем появляется кнопка ввода данных для входа, а также предложение зарегистрироваться, если у пользователя еще нет аккаунта (рис. 3).

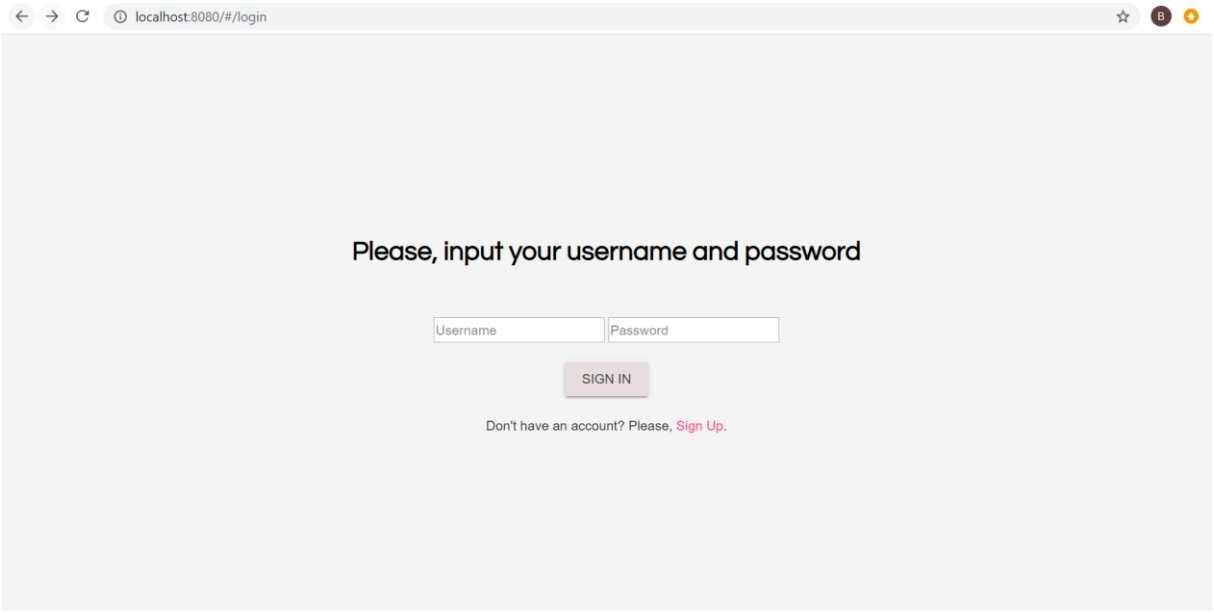

Рисунок 3 – Интерфейс входа в систему

После входа или регистрации пользователь наблюдает семь следующий вкладок: *Почтовые отделения*, *Газеты*, *Типографии*, *Склады*, *Запросы*, *Справка*, *Отчет* (рис. 4).

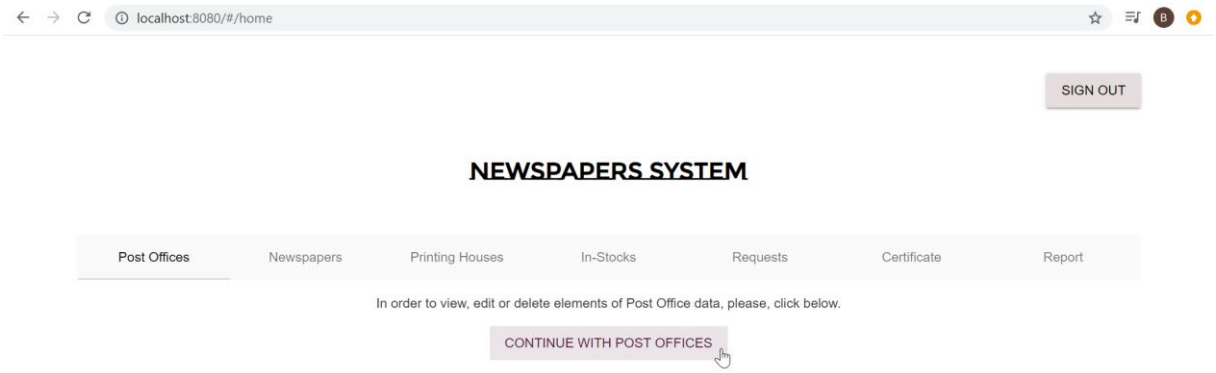

Рисунок 4 – Интерфейс домашней страницы системы

На каждой вкладке написано, чему она посвящена, и размещена кнопка для перехода к работе с данным элементом системы. Вкладка Газеты содержит не только кнопку для перехода к работе с газетами, а также кнопку для работы с редакторами. Вкладка с запросами содержит кнопки, на которых указан конкретных запрос (рис. 5).

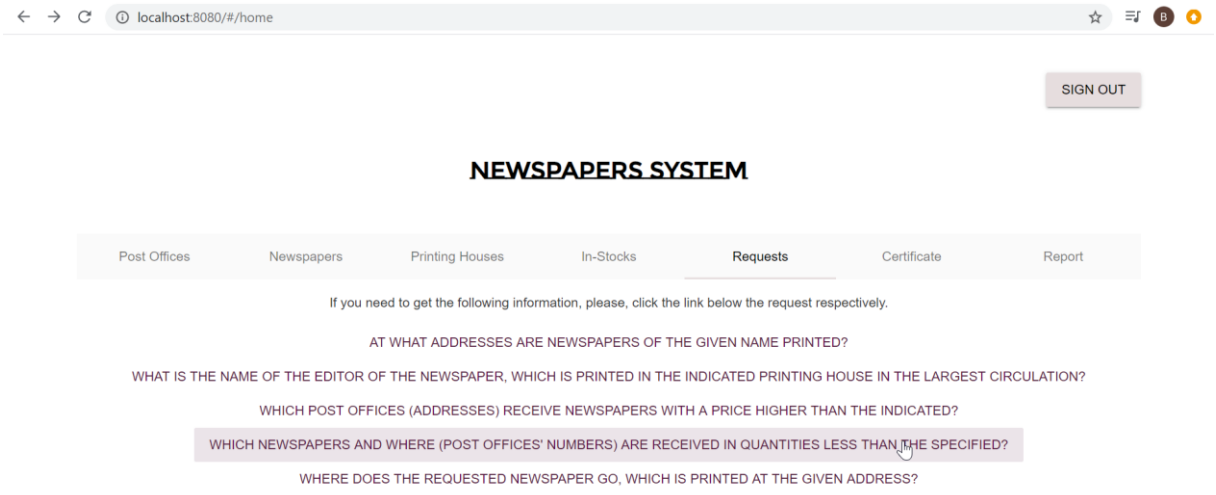

Рисунок 5 – Интерфейс вкладки запросов

При переходе к работе с конкретными таблицами базы данных (*Редакторы, Газеты, Почтовые отделения, Типографии, Склады*) появляется таблица, заполненная информацией из базы, а также вкладки «Удалить», «Добавить» и «Редактировать», предназначенные для изменения содержимого таблицы (рис. 6).

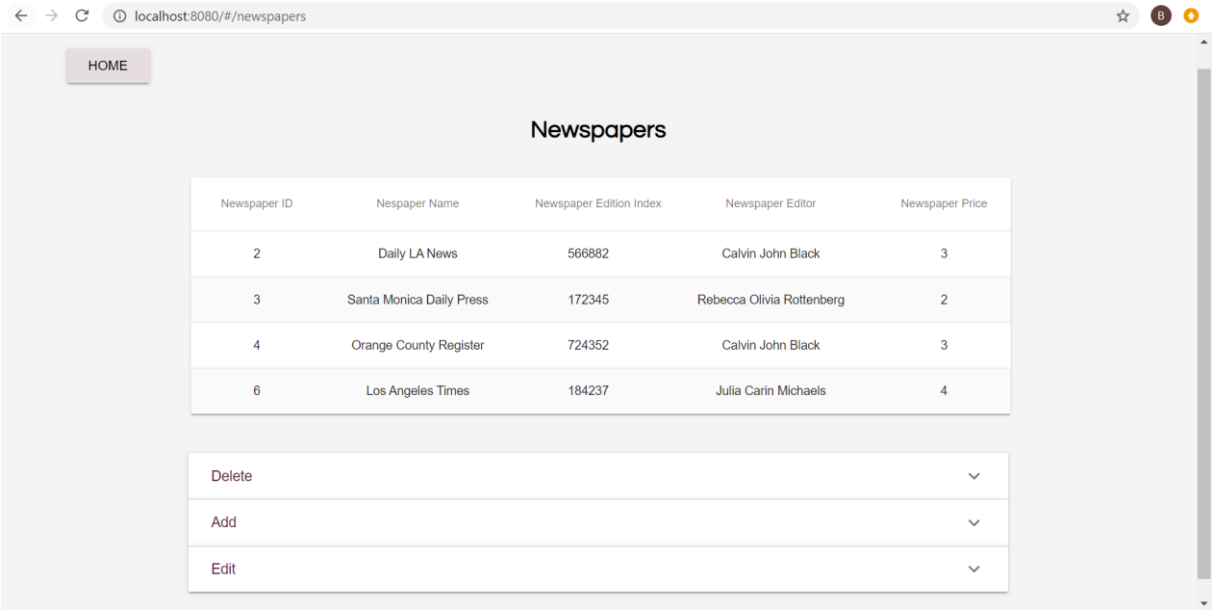

Рисунок 6 – Интерфейс для работы с газетами

Каждый интерфейс конкретного запроса имеет поле ввода данных, которые необходимы для его выполнения (рис. 7). После выполнения запроса отображается информация в виде таблицы или списка. Интерфейс справки имеет идентичный по функциональности интерфейс.

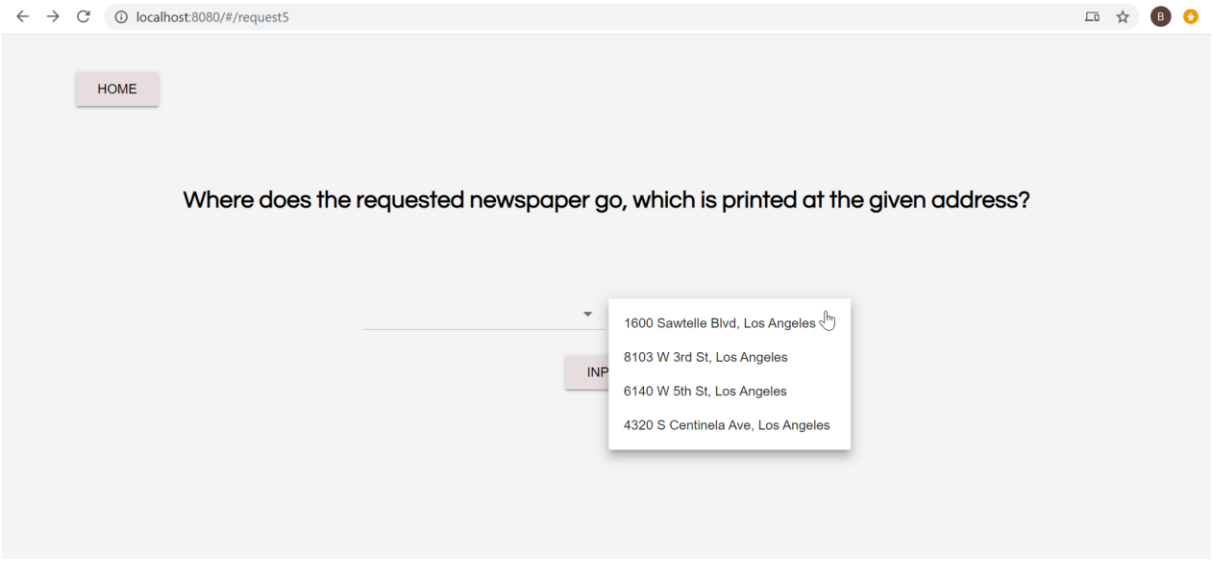

Рисунок 7 – Интерфейс запроса «Куда поступает данная газета, печатающаяся по данному адресу?»

Интерфейс отчета содержит сводную информацию по всем типографиям в виде текста (рис. 8).

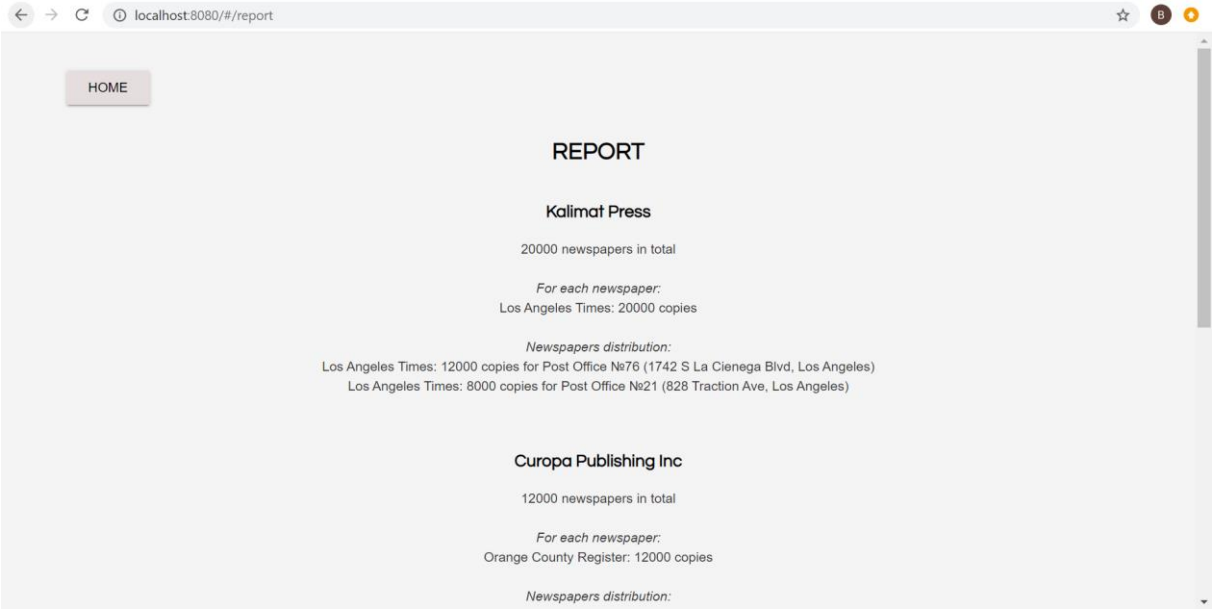

Рисунок 8 – Интерфейс отчета

Так как система должна быть удобна и приятна в использовании при длительном взаимодействии, дизайн системы является минималистичным, с нейтральной пастельной цветовой палитрой.

2.4. Проектирование архитектуры системы

Для реализации системы была выбрана клиент-серверная архитектура. Сервером в данном случае считается абстрактная машина в сети, способную получить HTTP-запрос, обработать его и вернуть корректный ответ. Под клиентом понимается программная оболочка, с которой взаимодействует пользователь.

Для разработки приложения используются средства Django, написанном на Python. Этот инструмент удобен для разработки сайтов, работающих с базами данных. Непосредственно в качестве базы данных используется PostgreSQL, в основе которого лежит свободная объектно-реляционная система управления базами данных. Система создает и управляет моделями и запросами базы данных.

Для создания веб-API интегрируется инструмент Django REST Framework, с помощью которого осуществляется сериализация данных из PostgreSQL базы в JSON формат, позволяющий обмениваться данными между клиентской и серверной частями. REST API [7]

интерфейс очень удобен для межпрограммного взаимодействия. REST API подразумевает под собой следующие правила: каждый URL является ресурсом; при обращении к ресурсу методом GET возвращается описание этого ресурса; метод POST добавляет новый ресурс; метод PUT изменяет ресурс; метод DELETE удаляет ресурс. Эти правила предоставляют простой CRUD (Create, Read, Update, Delete) интерфейс для других приложений, взаимодействие с которым происходит через протокол HTTP. В данном проекте GET запрос позволяет получить информацию из таблицы базы данных. В зависимости от целей запроса, результатом могут являться как все данные, так и только отдельный аспект данных, например, названия газет. POST запрос позволяет добавить новую запись в определенную таблицу базы данных. DELETE запрос позволяет удалять из таблицы определенную запись базы данных. PUT запрос позволяет изменить конкретную информацию уже существующей записи таблицы. Пользователь может изменить одно выбранное поле записи или несколько полей сразу.

Клиентская часть, разработанная средствами Vue.js и библиотеки Muse-UI для создания интерфейсов, передает данные в теле интересующего GET, POST, DELETE или PUT запроса через REST API. В результате запроса происходит изменение или извлечение данных из базы. Если данные введены не верно, или запрос выполнен некорректно, то появится сообщения об ошибке, вызванное соответствующим ошибке кодом состояния. Если запрос выполнен успешно, то программа продолжает свою работу.

После этого шага создается docker-обертка для проекта. Для базы данных, backend, frontend создаются контейнеры, которые позднее скоординируются за счет оркестрации.

2.5. Выводы

На основе анализа компонентов и функционала системы был выбран стек технологий, позволяющий эффективно и удобно создать системы, удовлетворяющую предъявляемым требованиям.

Спроектирована модель данных, которая в дальнейшем послужит основой для программирования модели с помощью обозначенных технологий. В модели обозначены сущности, их атрибуты и связи между ними. Выбранная структура базы данных отличается лаконичностью, но при этом сохраняет все необходимые элементы.

Далее была произведена разработка основных интерфейсов и продумана концепция дизайна. Каждый интерфейс посвященной одной из целей – вход в систему, регистрация в системе, работа с таблицами данных, получение результатов запросов, генерация отчета или получение справки.

Разработка архитектуры системы основывалась на выбранных инструментах, поставленных задачах и определенных интерфейсах.

Таким образом, этап разработки и проектирования системы достиг целей проекта в вопросах удобства система, ее эффективности и быстродействия.

3. ПРОГРАММИРОВАНИЕ СИСТЕМЫ

3.1. Backend

Первым шагом в создании сайта является инициализация Django проекта. Также был подключен PostgreSQL, что было отражено в настройках.

Следующим действием выступает создание моделей. Модель является единственным источником информации о данных. Она содержит основные поля данных, которые хранятся. Каждая модель отображается в одной таблице базы данных. Каждая модель представляет собой класс Python, который является подклассом django.db.models.Model. Каждый атрибут модели представляет поле базы данных. Таким образом, были созданы модели *Редактора, Газеты, Типографии, Почты, Склада*. Модели и их поля соответствуют сущностям и атрибутам базы данных (см. 2.3. Разработка модели данных). Для модели *Пользователя* используется get_user_model(), которая наследуется от django.contrib.auth. Этот метод возвращает текущую активную пользовательскую модель – User.

Также были созданы сериализаторы, которые представляет собой класс Python, который является подклассом rest_framework.serializers.ModelSerializer. Сериализаторы позволяют преобразовывать сложные данные, такие как наборы запросов и экземпляры модели, в собственные типы данных Python, которые затем можно легко преобразовать в JSON. Сериализаторы также обеспечивают десериализацию, что позволяет преобразовывать проанализированные данные обратно в сложные типы после предварительной проверки входящих данных. Сериализаторы наследуются от соответствующих моделей. Каждый сериализатор включает все поля модели.

Вслед за этим были созданы представления, основанные на функциях. Каждое представление принимает объект HttpRequest в качестве первого позиционного аргумента и возвращает объект HttpResponse или вызывает исключение. Представления было создано для каждой модели, кроме *Пользователя*, данных с функциями *get* (GET-запрос), *get_object* (поиск объекта по ID), *delete* (DELETE-запрос), *put* (PUT-запрос), *post* (POST-запрос). Были созданы представления под каждый запрос информации, справки и отчета (см. 1.1. Описание системы).

Для удобства редактирования и обновления информации, например, выбор одного из предложенных вариантов для заполнения поля, также были созданы представления.

Все представления отражены в URL-адресах приложения и URL-адресах проекта.

Таким образом, для взаимодействия API с сервером были созданы эндпоинты, некоторые из которых представлены ниже. Эндпоинт GET-запроса с URL-адресом сервера и /editorslist/, который возвращает полный список имен редакторов для представления в ниспадающем списке (рис. 9).

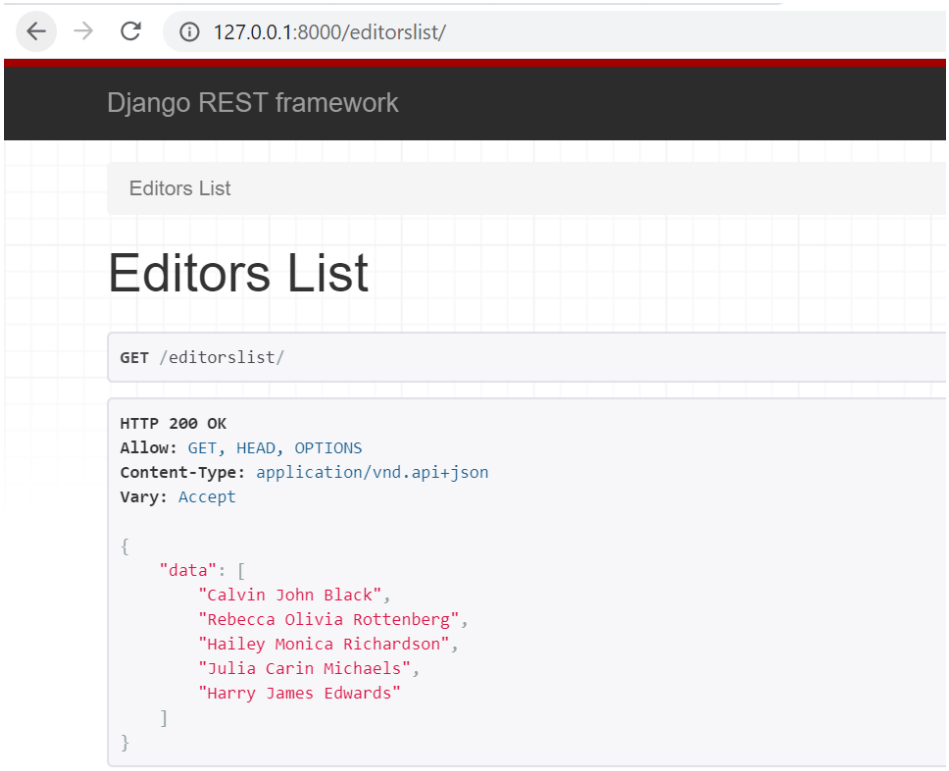

Рисунок 9 – Эндпоинт GET-запроса, возвращающий полные имя редакторов для представления в ниспадающем списке.

Также был создан эндпоинт GET-запроса с URL-адресом сервера и /printinghouse/, который возвращает данные об одной типографии по введенному значению ID (рис. 10).

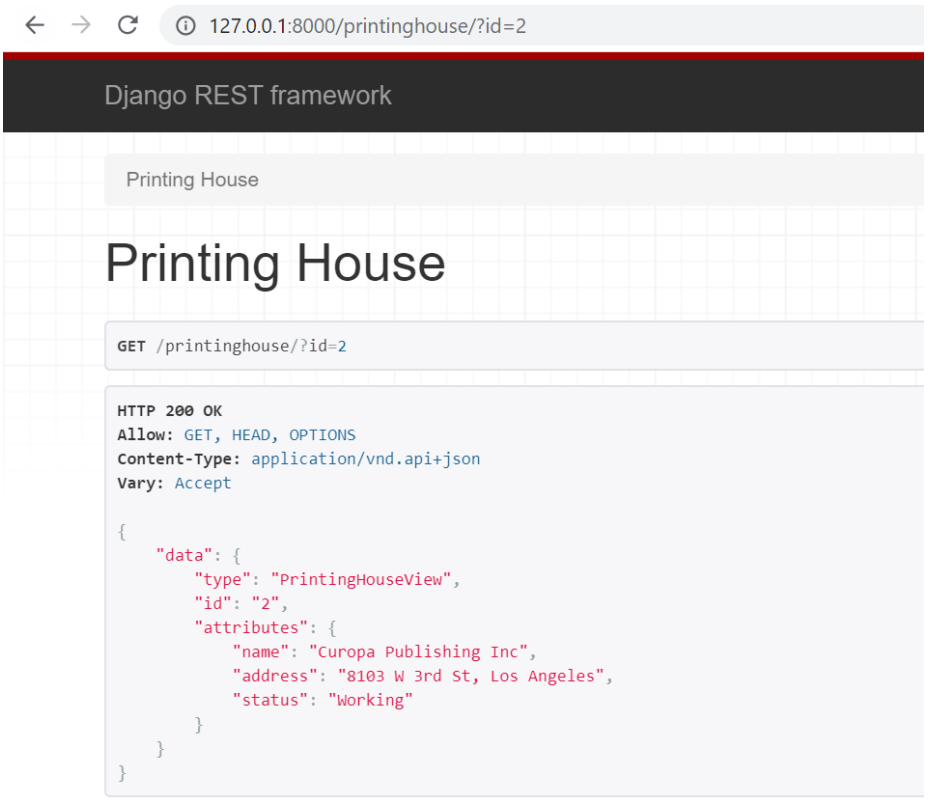

Рисунок 10 – Эндпоинт GET-запрос, возвращающий данные о типографии с ID = 2.

Был создан эндпоинт *Складов* с URL-адресом сервера и /instocks/, поддерживающий GET-, POST-, PUT-, DELETE-запросы. (рис. 11).

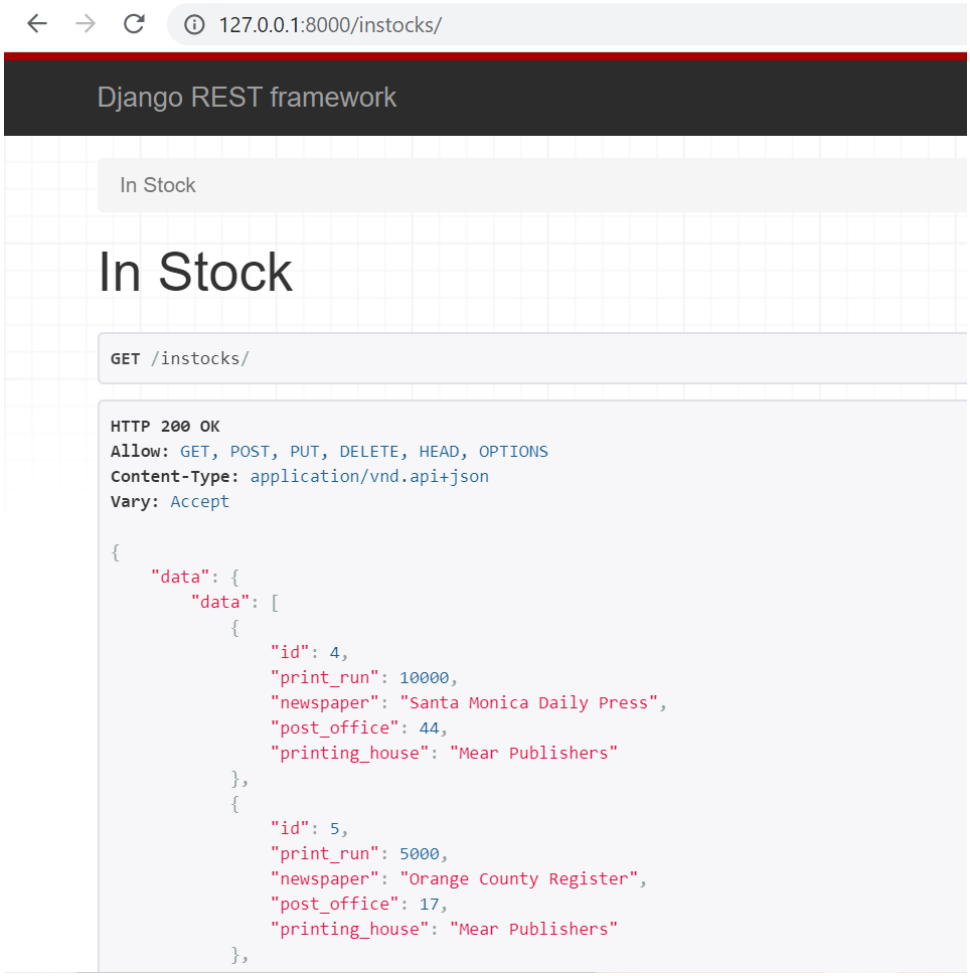

Рисунок 11 – Эндпоинт *Складов* с URL-адресом сервера и /instocks/, поддерживающий GET-, POST-, PUT-, DELETE-запросы.

3.2. Frontend

Для каждого интерфейса был разработан свой Vue.js компонент, состоящий из трех частей *template, script, style*. В части компонента *template* использовались Muse-UI элементы: панели, кнопки, таблицы, текстовые поля, вкладки, выпадающие списки. В части *script* содержится информация о данных компонента, хук создания (хук жизненного цикла) *created()*, а также функции компонента, описанные в методах системы. Некоторые функции выполняются автоматически при загрузке страницы, другие функции выполняются после нажатия на элемент интерфейса. Функции, выполняющиеся при загрузке страницы, позволяют отображать данные незамедлительно. Функции, выполняющиеся после нажатия на элемент интерфейса, подставлены в двух типах. Первые относятся к изменению какого-либо свойства

элемента или совершению над ним действия. Примером может являться раскрытие панелей. Вторые содержат *Ajax* запросы.

Ajax – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. В запросе указан URL, тип запроса, данные, которые будут переданы в вид, а также действия, которые необходимо выполнить, если запрос выполнен успешно, или если результатом запроса является ошибка.

Если запрос выполнен успешно, то Vue.js компонент получает данные в формате JSON, которые отображает как часть одного из элемента интерфейса, например, таблицы. Если же получена ошибка, то пользователь увидит окно, сообщающее об ошибке.

В части *style* описаны атрибуты, относящиеся к дизайну компонентов интерфейса.

3.3. Контейнеризация и оркестрация

Средствами docker и docker-compose проект контейнеризован и оркестрирован.

Все компоненты, необходимые для запуска приложения, упаковываются как один образ и могут быть использованы повторно. Приложение в контейнере работает в изолированной среде и не использует память, процессор или диск хостовой операционной системы. Это гарантирует изолированность процессов внутри контейнера.

3.4. Выводы

В результате шага программирования системы был создан каркас проекта, который был позднее доработан. По части backend были обновлены настройки и URL-адреса, добавлены представления, модели, сериализаторы. По части frontend были созданы Vue.js компоненты, которые послужили основой для клиентской части проекта. Ресурсы системы были виртуализированы и стали изолированы на уровне операционной системы.

ЗАКЛЮЧЕНИЕ

Рассмотрим достижение цели данного проекта.

Его непосредственной целью является создание программной системы, позволяющей отслеживать распределение по почтовым отделениям газет, печатающихся в типографиях города. Данная цель была достигнута в полной мере.

Созданный продукт прост в использовании, отличаться удобным и понятным для любого человека интерфейсом. Проект также способствует решению задач, с которыми сталкиваются сотрудники почтовых отделений в своей работе. Система работает быстро и не использует большой объем вычислительный ресурсов, выполняет указанные действия мгновенно.

Каждый требуемый компонент рекомендательной системы был реализован.

Система была протестирована на предмет выявления ее недостатков. Задача включила в себя тестирование различных компонентов системы, тестирование работы системы при различных данных. Также проект был протестирован на группе людей и был признан удобным в использовании, легким в понимании и отличающимся функциональным интерфейсом.

Данные, полученные в результате использования системы, являются адекватными и точными, так как система была разработана на принципе максимальной практичности для пользователя.

Дальнейшим шагом развития системы может являться расширение функционала на основе увеличения задач, требуемых к выполнению от сотрудников почты, доработка интерфейса относительно его комфорта в использовании, расширение базы данных за счет увеличения сущностей и атрибутов, которые может контролировать работник.

Таким образом, данный продукт отвечает поставленным на стадии его проектирования задачам и целям. Он имеет возможность быть востребованным на рынке, а также значительное время оставаться актуальным благодаря тому, что имеет множество перспектив развития.

СПИСОК ЛИТЕРАТУРЫ

1. Django documentation [Электронный ресурс] // Django documentation | Django documentation | Django [сайт]. 2005 – 2020. URL: https://docs.djangoproject.com/en/3.0/ (дата обращения: 25.05.2020).

2. Home - Django REST framework [сайт]. 2011 – present. URL: https://www.django-rest-framework.org/ (дата обращения 26.05.2020).

3. The Progressive JavaScript Framework [Электронный ресурс] // Vue.js [сайт]. 2014-2020. https://vuejs.org/ (дата обращения 26.05.2020).

4. Muse-UI [Электронный ресурс] // Muse-UI [сайт]. 2016. URL: https://muse-ui.org/#/en-US (дата обращения 28.05.2020).

5. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс] // PostgreSQL: The world's most advanced open source database [сайт]. 1996-2020. URL: https://www.postgresql.org/ (дата обращения 26.05.2020).

6. Get Started with Docker [Электронный ресурс] // Empowering App Development for Developers | Docker [сайт]. 2020. URL: https://www.docker.com/ (дата обращения: 21.06.2020).

7. REST API [Электронный ресурс] // REST API — Основы Веб-программирования [сайт] (дата обращения 27.05.2020).

Приложение 1. Использованные модули и программы

Версии использованных модулей и программ:

- Django 3.0.4;
- django-allauth 0.41.0;
- django-cors-headers 3.2.1;
- django-rest-swagger 2.2.0;
- django-templated-mail 1.1.1;
- djangorestframework 3.11.0;
- djangorestframework-jsonapi 3.1.0;
- djangoframework-jwt 1.11.0;
- djoser 2.0.3;
- Docker Toolbox for Windows 19.03.1
- Muse-UI 3.0.2;
- PostgreSQL 11.8;
- Python 3.7.7;
- Vue.js 2.6.11.