Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

**КУРСОВАЯ-ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2-3**

**«РЕАЛИЗАЦИЯ ВЕБ-СЕРВИСА НА DJANGO»**

**Факультет: ИКТ**

**Направление: ИТГС**

**Группа: К3342**

**Выполнил:**

**Кочешкова Ксения**

**Преподаватель:**

**Говоров А.И.**

Санкт-Петербург

2020

Оглавление

[Введение: 3](#_Toc55442566)

[1. Предметная область 4](#_Toc55442567)

[1.1. Описание предметной области: 4](#_Toc55442568)

[1.2. Схема базы данных 5](#_Toc55442569)

[2. Реализация 6](#_Toc55442570)

[2.1. Теория по Django 6](#_Toc55442571)

[2.2. Теории о Vue.js 9](#_Toc55442572)

[2.3. Составляющие проекта и приложения 10](#_Toc55442573)

[2.4. Docker file 14](#_Toc55442574)

[2.5. Страницы vue 16](#_Toc55442575)

[3. Полученный сайт 18](#_Toc55442576)

[3.1. Меню 18](#_Toc55442577)

[3.2. Регистрация 18](#_Toc55442578)

[3.3. Раздел «Книга» 20](#_Toc55442579)

[3.4. Разделы «Посетитель» и «Библиотекарь» 21](#_Toc55442580)

[3.5. Страница администратора 23](#_Toc55442581)

[Заключение: 25](#_Toc55442582)

# Введение:

Целью выполнения курсовой работы является овладение практическими навыками и умениями реализации web-сервисов средствами Django 2.2. Компьютерный класс используется как оборудование, а в качестве программного обеспечения — **Python 3.6, Django 2.2, PostgreSQL \*.**

* В ходе выполнения курсовой работы необходимо решить следующие задачи:
* Разбор предметной области;
* Разбор теории, касающейся разработки веб-приложения на **Django;**
* **Разработка схемы базы данных, а также модели на Django, в которой были бы отмечены все связи;**
* **Реализация веб-приложения.**

**В качестве варианта был выбран второй, который предлагает** создать программную систему, предназначенную для работников библиотеки. Такая система должна обеспечивать хранение сведений об имеющихся в библиотеке книгах, о читателях библиотеки и читальных залах.

Для каждой книги в БД должны храниться следующие сведения: название книги, автор (ы), издательство, год издания, раздел, число экземпляров этой книги, а также шифр книги и дата закрепления книги за читателем. Сведения о читателях библиотеки должны включать номер читательского билета, ФИО читателя, номер паспорта, дату рождения, адрес, номер телефона, образование, наличие ученой степени.

Читатели закрепляются за определенным залом и могут записываться и выписываться из библиотеки. Библиотека имеет несколько читальных залов, которые характеризуются номером, названием и вместимостью, то есть количеством людей, которые могут одновременно работать в зале. Библиотека может получать новые книги и списывать старые. Шифр книги может измениться в результате переклассификации, а номер читательского билета в результате перерегистрации.

# Предметная область

## Описание предметной области:

Библиотека представляет собой хранилище книг. Основные ее функции — выдавать читателям книги и принимать их. Остальные функции относятся к обслуживанию процесса выдачи и приема. Экземпляр каждой книги, хранящейся в библиотеке, должен иметь индивидуальный номер. Каждый читатель должен иметь свой абонемент, который хранит как историю заказов, так и информацию о самом читателе.

Книги выдаются читателям на руки, на определенный срок. Индивидуальный номер книги, дата выдачи и дата возврата записываются в читательский билет. В пределах указанного срока читатель должен вернуть книгу в библиотеку. При необходимости срок использования книги может быть продлен читателем. Если читатель не вернул книгу в срок и не сообщил о продлении, то у него возникает задолженность. Продлить книгу можно с помощью администратора (библиотекаря), достаточно сообщить об этом сотруднику. Наличие задолженностей влияет на дальнейшие отношения библиотеки с читателем.

Каждый читатель имеет индивидуальный номер и читательский билет. По индивидуальному номеру работник библиотеки может определить ФИО читателя, дату рождения и его контактные данные Придя в библиотеку, читатель, делает заказ на книгу. Сотрудник библиотеки может принять заказ в устной или письменной форме. Сотрудник принимает заказ, проверяет наличие ассортимента и по возможности выдает необходимые книги.

Информация о выданной книге заносится библиотекарем в читательский билет, где указывается дата выдачи книги и предполагаемая дата возврата. Это позволяет регулировать возврат книг и определять должников. В случае невозврата книги в указанные сроки, сотрудник библиотеки может, воспользовавшись абонементом читателя, определить его личные данные и обратится к нему с уведомлением или же напомнить ему посредством телефонного звонка об имеющемся долге. В случае утери читатель обязуется выплатить полную стоимость экземпляра или же возместить ущерб покупкой такой же самой книги для библиотеки.

Кроме того, есть отдельные функции, которые библиотекарь может выполнять. К ним можно отнести организацию мероприятий. Достаточно часто библиотека используется в качестве площадки, на которой реализуют встречи читателей с авторами, проводят мероприятия, посвященные тем или иным праздникам («Библионочь»), самостоятельные собрания фанклубов. У каждого подобного мероприятия есть лимит на количество людей, которые могут принять участие.

## Схема базы данных

Для быстрой реализации веб-приложения достаточно будет выбрать несколько главных функций библиотеке, а также выделить несколько главных сущностей. В данном случае можно оставить «читателя», «библиотекаря», «книги», «мероприятия» (оно же «аренда», так как библиотекари, по сути, арендуют тот или иной зал для проведения мероприятия), «залы». Кроме того, буду добавлены промежуточные элементы «владение» (выдача какой-либо книги читателю на руки), «принадлежность» (принадлежность читателя к тому или иному залу).

Все элементы, а также связи отражены на схеме 1.

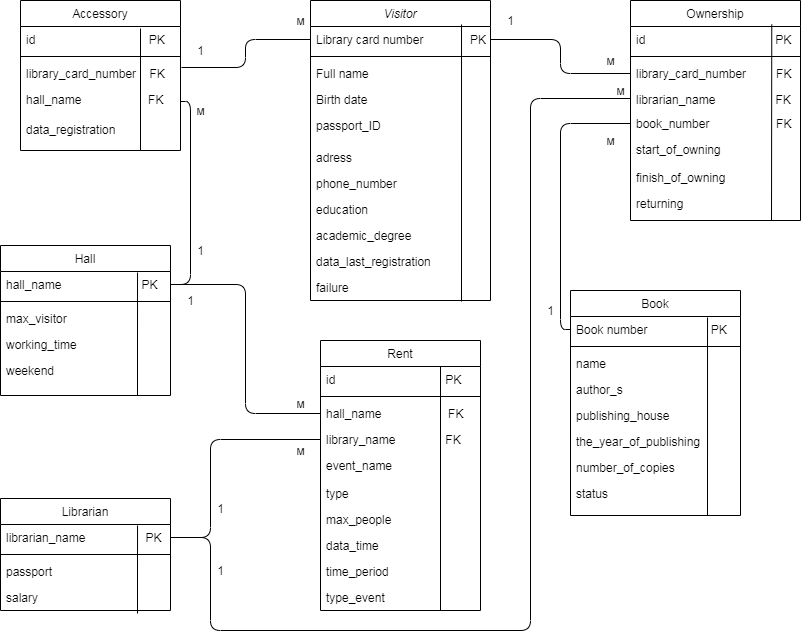


Схема 1. База данных «библиотеки»

# Реализация

## Теория по Django

Django — это Open Source[[1]](#footnote-1) фреймворк для создания веб-приложений различной сложности. Одним из основных преимуществ Django является то, что необходимо позаботиться только о логике будущего приложения, остальное сделает Django. Django считается лучшим веб-фреймворком, написанным на Python (по мнению народа он обощел Flask, Bottle, Zope и CherryPy). Этот инструмент удобно использовать для разработки сайтов, работающих с базами данных.

В Django реализован принцип DRY (don’t repeat yourself). Благодаря этому сокращается время создания сайтов. То есть при использовании Django не нужно несколько раз переписывать один и тот же код. Фреймворк позволяет создавать сайт из компонентов.

Фреймворк Django справляется с большим количеством задач и повышенными нагрузками. Его применяют для создания:

* CRM-систем (Customer Relationship Management или Управление отношениями с клиентами — это — прикладное программное обеспечение для организаций, предназначенное для автоматизации стратегий взаимодействия с заказчиками (клиентами), в частности, для повышения уровня продаж, оптимизации маркетинга и улучшения обслуживания клиентов путем сохранения информации о клиентах и истории взаимоотношений с ними, установления и улучшения бизнес-процессов и последующего анализа результатов);
* CMS (**Система управления содержимым**  — информационная система или компьютерная программа, используемая для обеспечения и организации совместного процесса создания, редактирования и управления содержимым, иначе — контентом)
* Коммуникационных платформ.
* Сервисов бронирования номеров.
* Платформ управления документооборотом.

Также Django подходит для создания алгоритмических генераторов, платформ для электронных рассылок, систем верификации, систем фильтрации с динамическими правилами и сложными параметрами, платформ для анализа данных и сложных вычислений, машинного обучения.

Основные преимущества фреймворка, благодаря которым он стал популярным.

1) Развитая экосистема:

Опытные разработчики рекомендуют воспринимать Django как систему. Это значит, что фреймворк обычно используется с большим количеством сторонних приложений. Их можно выбирать в зависимости от потребностей конкретного проекта.

2) Зрелость:

Django был представлен в 2005 году. За 14 лет существования он сильно изменился и усовершенствовался. В фреймворке постоянно появляются новые возможности, а старые совершенствуются.

3) Административная панель:

Административная панель Django автоматически генерируется при создании приложения. Это избавляет разработчика от необходимости создавать админку вручную.

С помощью сторонних приложений дефолтную консоль управления Django можно усовершенствовать и адаптировать под нужды своего проекта. Кроме того, фреймворк позволяет настраивать интерфейс дефолтной административной панели.

4) SEO-дружественность:

Написанный на Python код получается читабельным и понятным даже неподготовленным людям. Это один из факторов, благодаря которым веб-приложения на Python считаются SEO-дружественными. Django генерирует семантические URL. Их также называют человеко-понятными URL или ЧПУ. В приложениях на Django легко реализуются другие функции, необходимые для поисковой оптимизации.

5) Расширяемость

Функциональность Django расширяется с помощью плагинов. Это программные модули, которые позволяют быстро добавить на сайт нужную функцию. В официальном каталоге есть сотни плагинов, которые позволяют легко реализовать на сайте sitemap.xml, управлять доступами, подключить платёжную систему Stripe и так далее. При необходимости можно отключать или заменять плагины, чтобы адаптировать приложение к текущим нуждам проекта.

6) Библиотеки:

В популярных языках программирования есть библиотеки, с помощью которых удобно решать специальные задачи. В библиотеках можно найти готовые решения: функции, классы, конфигурации и так далее. Благодаря таким решениям расширяются возможности языка, а также упрощается создание приложений.

Django поддерживает использование библиотек при разработке веб-приложений. В число популярных библиотек входят:

* Django REST Framework, который упрощает работу с API.
* Django CMS — удобный инструмент для управления контентом.
* Django-allauth — с его помощью реализуются функции регистрации, авторизации, управления учётными записями.

7) ORM:

В Django реализовано объектно-реляционное отображение (ORM), которое обеспечивает взаимодействие приложения с базами данных (БД). ORM автоматически передаёт данные из БД, например, PostgreeSQL или MySQL, в объекты, которые используются в коде приложения.

ORM ускоряет разработку прототипов и готовых веб-приложений на Django. Также ORM позволяет быстро переключаться между базами данных с минимальными изменениями кода. Например, вы можете использовать SQLite на локальном сервере, а потом переключиться на MySQL на production-сервере.

У Django есть недостатки, как у любого инструмента:

1) Не подходит для небольших проектов:

Возможности Django могут быть избыточными для небольшого проекта.

2) Нет поддержки WebSocket по умолчанию:

Протокол WebSocket обеспечивает обмен данными между сервером и браузером в режиме реального времени. Django не поддерживает эту функциональность из коробки. Но проблему можно решить с помощью других инструментов, например, aiohttp.

3) Монолитность:

Некоторые модули Django, например, ORM или формы, сложно заменить. Разработчикам приходится тратить много времени, чтобы изменить внутреннюю структуру фреймворка и использовать альтернативные инструменты.

4) Непредсказуемость поведения некоторых компонентов:

Поведением некоторых компонентов Django не всегда просто управлять. Например, к таким компонентам относится административная консоль. Добавление чего-то, не входящего в экосистему Django, может потребовать серьёзных усилий и много времени.

В целом преимущества Django с запасом перекрывают недостатки этого фреймворка.

## Теории о Vue.js

Vue.js представляет современный прогрессивный фреймворк, написанный на языке JavaScript и предназначенный для создания веб-приложений клиентского уровня. Основная сфера применения данного фреймворка — создание и организация пользовательского интерфейса.

Официальный сайт фреймворка - <https://vuejs.org/>,где можно найти непосредственно файлы фреймворка, а также сопроводительные материалы и документацию. Кроме того, сам проект доступен на github по адресу <https://github.com/vuejs/vue>.

Vue.js имеет довольно небольшой размер - не более 20 кБ, и при этом обладает хорошей производительностью по сравнению с такими фреймворками как Angular или React. Поэтому неудивительно, что данный фреймворк в последнее время набирает обороты и становится все более популярным.

Одним из ключевых моментов в работе Vue.js является виртуальный DOM. Структура веб-страницы, как правило, описывается с помощью DOM (Document Object Model), которая представляет организацию элементов html на странице. Для взаимодействия с DOM (добавления, изменения, удаления html-элементов) применяется JavaScript. Но когда мы пытаемся манипулировать html-элементами с помощью JavaScript, то мы можем столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте. Однако если бы мы работали из кода js с объектами JavaScript, то операции производились бы быстрее.

Для этого Vue.js использует виртуальный DOM. Виртуальный DOM представляет легковесную копию обычного DOM. Если приложению нужно узнать информацию о состоянии элементов, то происходит обращение к виртуальному DOM. Если данные, которые используются в приложении Vue.js, изменяются, то изменения вначале вносятся в виртуальный DOM. Потом Vue выбирает минимальный набор компонентов, для которых надо выполнить изменения на веб-странице, чтобы реальный DOM соответствовал виртуальному. Благодаря виртуальному DOM повышается производительность приложения.

Vue.js поддерживается всеми браузерами, которые совместимы с ECMAScript 5. На данный момент это все современные браузеры, в том числе IE11.

## Составляющие проекта и приложения

Документация Django определяет модель (model) как «источник информации о данных, в которых содержатся ключевые поля и поведение данных». Обычно одна модель указывает на одну таблицу в базе данных. Django поддерживает базы данных PostgreSQL, MySQL, SQLite и Oracle.

Модели содержат информацию о данных. Эти данные представлены атрибутами или полями. Поскольку модель представляет собой простой класс, она ничего не знает о других уровнях Django. Взаимодействие между уровнями происходит через API.

Модель отвечает за бизнес-логику, методы, свойства и другие элементы, связанные с манипуляцией данными. Также модели позволяют разработчикам создавать, читать, обновлять и удалять объекты в базе данных.

Модель в соответствии с показанной выше схемой:

from django.db import models  
  
  
class Visitor(models.Model):  
 EDUCATION\_CHOICES = (  
 ('GP', 'General\_preschool'),  
 ('GI', 'General\_initial'),  
 ('GtM', 'General\_the\_main'),  
 ('GtA', 'General\_the\_average'),  
 ('PI', 'Professional\_initial'),  
 ('PtA', 'Professional\_the\_average'),  
 ('PH', 'Professional\_higher'),  
 ('PP', 'Professional\_postgraduate'),  
 ('AG', 'Additional\_general'),  
 ('AP', 'Additional\_professional'),  
 )  
 library\_card\_number = models.CharField(max\_length=20)  
 full\_name = models.CharField(max\_length=150)  
 birth\_date = models.DateField()  
 passport\_ID = models.IntegerField()  
 adress = models.CharField(max\_length=50)  
 phone\_number = models.CharField(max\_length=12)  
 education = models.CharField(max\_length=50, choices=EDUCATION\_CHOICES)  
 academic\_degree = models.BooleanField()  
 data\_last\_registration = models.DateField()  
 failure = models.BooleanField()  
   
 def \_\_str\_\_(self):  
 return self.library\_card\_number  
  
  
class Book(models.Model):  
 book\_number = models.CharField(max\_length=20)  
 name = models.CharField(max\_length=150)  
 author\_s = models.CharField(max\_length=150)  
 publishing\_house = models.CharField(max\_length=50)  
 the\_year\_of\_publishing = models.CharField(max\_length=50)  
 number\_of\_copies = models.CharField(max\_length=50)  
 status = models.CharField(max\_length=20)  
   
 def \_\_str\_\_(self):  
 return self.book\_number  
  
  
class Hall(models.Model):  
 hall\_name = models.CharField(max\_length=20)  
 max\_visitor = models.CharField(max\_length=50)  
 start\_time\_working = models.CharField(max\_length=40)  
 finish\_time\_working = models.CharField(max\_length=40)  
 weekend = models.CharField(max\_length=50)  
   
 def \_\_str\_\_(self):  
 return self.hall\_name  
  
  
class Librarian(models.Model):  
 passport = models.CharField(max\_length=12)  
 librarian\_name = models.CharField(max\_length=150)  
 salary = models.IntegerField(default=0)  
  
 def \_\_str\_\_(self):  
 return self.librarian\_name  
  
  
class Ownership(models.Model):  
 visitor = models.ForeignKey('Visitor', on\_delete=models.CASCADE)  
 book = models.ForeignKey('Book', on\_delete=models.CASCADE)  
 librarian = models.ForeignKey(Librarian, on\_delete=models.CASCADE)  
 start\_of\_owning = models.CharField(max\_length=40)  
 finish\_of\_owning = models.CharField(max\_length=40)  
 returning = models.CharField(max\_length=40)  
  
  
class Accessory(models.Model):  
 visitor = models.ForeignKey('Visitor', on\_delete=models.CASCADE)  
 hall = models.ForeignKey('Hall', on\_delete=models.CASCADE)  
 data\_registration = models.DateField()  
  
  
class Rent(models.Model):  
 hall = models.ForeignKey('Hall', on\_delete=models.CASCADE)  
 librarian = models.ForeignKey('Librarian', on\_delete=models.CASCADE)  
 event\_name = models.CharField(max\_length=50)  
 type\_event = models.CharField(max\_length=50)  
 max\_people = models.IntegerField()  
 data\_time = models.DateTimeField()  
 time\_period = models.TimeField()

Представление (view) решает три задачи: принимает HTTP-запросы, реализует бизнес-логику, определённую методами и свойствами, отправляет HTTP-ответ в ответ на запросы. То есть представление получает данные от модели и предоставляет шаблонам (templates) доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам.

В Django реализован мощный движок шаблонов и собственный язык разметки. Шаблоны представляют собой файлы с HTML-кодом, с помощью которого отображаются данные. Содержимое файлов может быть статическим или динамическим. Шаблоны не содержат бизнес-логики. Поэтому они только отображают данные.

View.py:

class LibrarianListView(generics.ListAPIView):  
 queryset = Librarian.objects.all()  
 serializer\_class = LibrarianSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class VisitorListView(generics.ListAPIView):  
 queryset = Visitor.objects.all()  
 serializer\_class = VisitorSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class HallListView(generics.ListAPIView):  
 queryset = Hall.objects.all()  
 serializer\_class = HallSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class BookListView(generics.ListAPIView):  
 queryset = Book.objects.all()  
 serializer\_class = BookSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class AddBookView(generics.CreateAPIView):  
 queryset = Book.objects.all()  
 serializer\_class = BookSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class OwnershipListView(generics.ListAPIView):  
 queryset = Ownership.objects.all()  
 serializer\_class = OwnershipSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class AccessoryListView(generics.ListAPIView):  
 queryset = Accessory.objects.all()  
 serializer\_class = AccessorySerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class RentListView(generics.ListAPIView):  
 queryset = Rent.objects.all()  
 serializer\_class = RentSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class AddRentView(generics.CreateAPIView):  
 queryset = Rent.objects.all()  
 serializer\_class = RentSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly  
  
  
class AddOwnershipView(generics.CreateAPIView):  
 queryset = Ownership.objects.all()  
 serializer\_class = OwnershipSerializer  
 permission\_class = permissions.IsAuthenticatedOrReadOnly

Для преобразования сложных данных (например, экземпляров моделей) в нативные типы данных Python используют сериализаторы. В последствии с их помощью эти данные можно легко срендерить в Json, XML или другие типы контента. Кроме того, сериализаторы обеспечивают десереилизацию, которая позволяет преобразовать данные обратно в сложные типы.

Serializers.py:

class LibrarianSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Librarian  
 fields = '\_\_all\_\_'  
  
  
class VisitorSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Visitor  
 fields = '\_\_all\_\_'  
  
  
class HallSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Hall  
 fields = '\_\_all\_\_'  
  
  
class BookSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Book  
 fields = '\_\_all\_\_'  
  
  
class OwnershipSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Ownership  
 fields = '\_\_all\_\_'  
  
  
class AccessorySerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Accessory  
 fields = '\_\_all\_\_'  
  
  
class RentSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Rent  
 fields = '\_\_all\_\_'

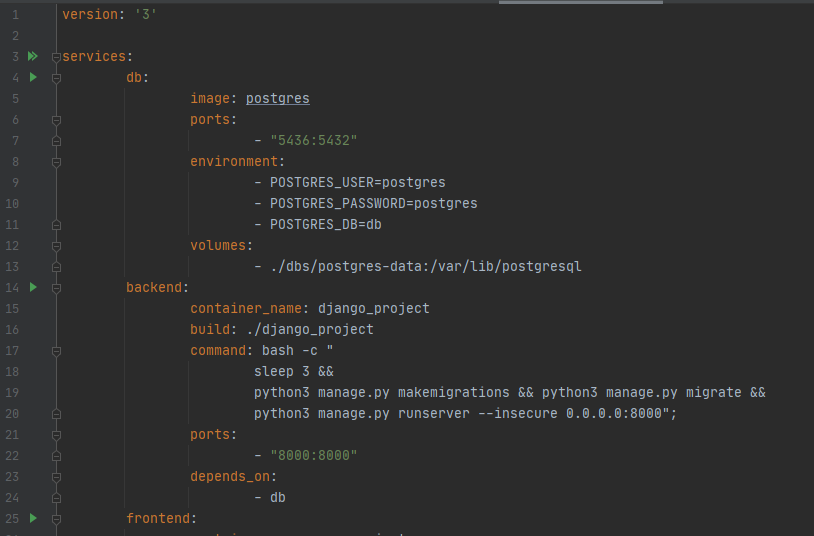
## Docker file

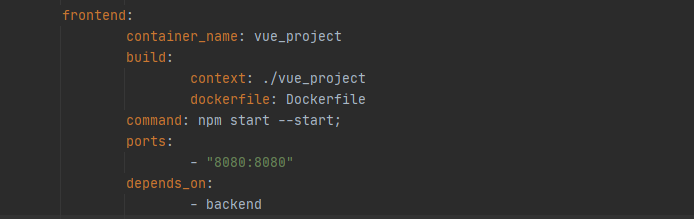
Можно запускать PostgreSQL локально, используя такой инструмент, как Postgres.app, однако сегодня среди многих разработчиков предпочтительным является использование Docker, инструмента для создания изолированных операционных систем. Проще всего представлять это как виртуальную среду, которая содержит все необходимое для нашего проекта Django: зависимости, базы данных, службы кэширования и любые другие необходимые инструменты.

Основная причина использования Docker заключается в том, что он полностью устраняет любые проблемы, связанные с локальной разработкой. Вместо того, чтобы беспокоиться о том, какие программные пакеты установлены или работают с локальной базой данных вместе с проектом, вы просто запускаете образ Docker всего проекта. Лучше всего то, что этим можно поделиться в группах и значительно упростить разработку команды.

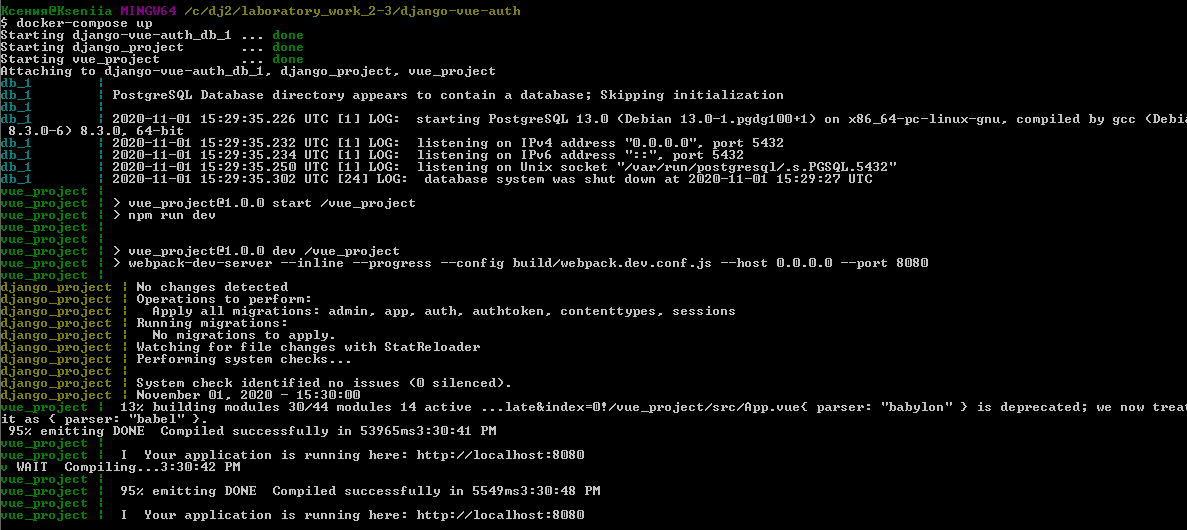
Кроме того, в файле docker реализуется миграция модели в базу данных.

Файл docker, через который реализуется запуск сайта (а также небольшие функции вроде «миграции»):





Запуск файла через «Docker Quickstart Terminal» (несмотря на указанный адрес, сайт будет иметь адрес http://192.168.99.100:8080/)



## Страницы vue

В формате vue прописываются станицы, которые будут отображаться в веб-приложении. Учитывая составленную ранее схему, необходимо прописать следующий страницы: book, librarian, visitor, rent, ownership, auth, registr, addbook, homeи т.д. Кроме того для отображения главной страницы необходимо прописать файл App.vue.

Одна из страниц веб-приложения – book.vue:

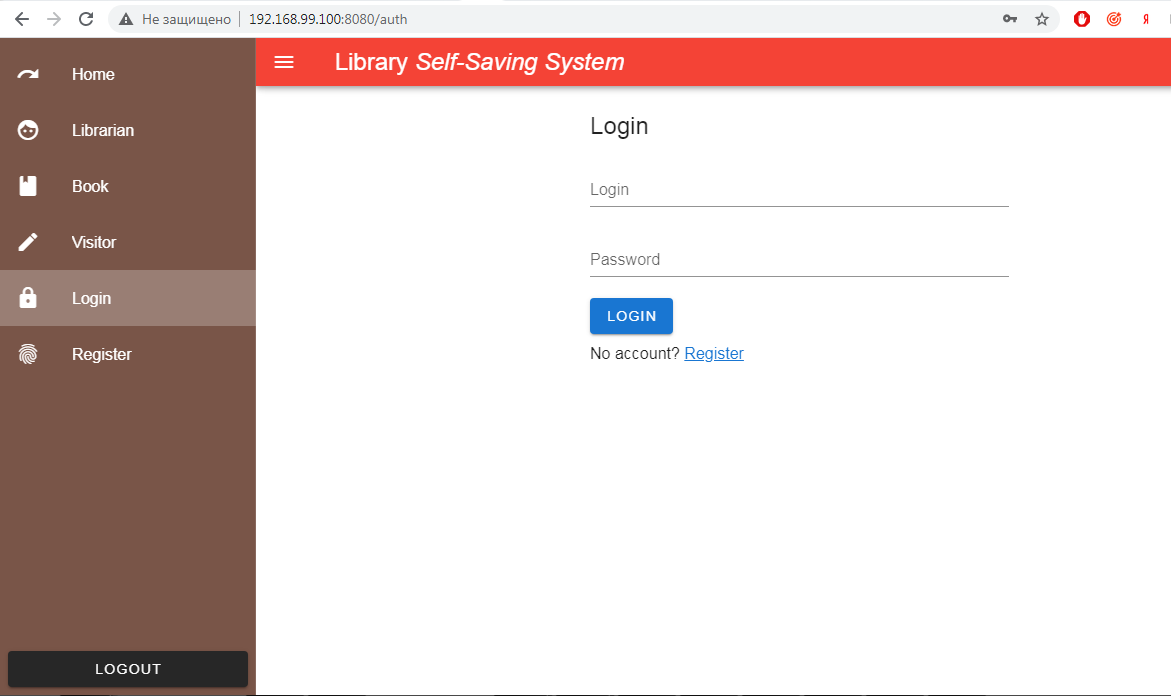
<template>  
 <main>  
 <section class="content">  
 <h2>Books list: {{ books.length }}</h2>  
 <v-container fluid>  
 <v-row>  
 <v-card class="my-1 mx-1" width=320 color="brown" dark v-for="book in books" :key="book.id">  
 <v-card-title class="headline"> Book number: {{ book.book\_number }} </v-card-title>  
 <v-card-text class="title">Name: {{ book.name }} </v-card-text>  
 <v-card-text class="title">Author\_s: {{ book.author\_s }} </v-card-text>  
 <v-card-text>Published by {{ book.publishing\_house }} in {{ book.the\_year\_of\_publishing }} </v-card-text>  
 <v-card-text>Number of copies: {{ book.number\_of\_copies}} </v-card-text>  
 <v-card-text> Status: {{ book.status }} </v-card-text>  
 </v-card>  
 </v-row>  
 </v-container>  
 <v-btn href="/books/add">New book</v-btn>  
 </section>  
 </main>  
</template>  
<script>  
 export default {  
 name: 'Books',  
 data () {  
 return {  
 books: []  
 }  
 },  
 mounted() {  
 this.axios  
 .get(`http://192.168.99.100:8000/api/books`)  
 .then(response => { this.showBooks(response.data) })  
 .catch(err => { ***console***.error(err) })  
 },  
 methods: {  
 showBooks(data) {  
 this.books = []  
 for (let i = 0; i < data.length; i++) {  
 let book = {}  
 book.book\_number = data[i].book\_number  
 book.name = data[i].name  
 book.author\_s = data[i].author\_s  
 book.publishing\_house = data[i].publishing\_house  
 book.the\_year\_of\_publishing = data[i].the\_year\_of\_publishing  
 book.number\_of\_copies = data[i].number\_of\_copies  
 book.status = data[i].status  
 ***console***.log(book)  
 this.books.push(book)  
 }  
 },  
 }  
 }  
</script>  
<style></style>

# Полученный сайт

## Меню

В меню представлено несколько разделов, среди них:

* Home – домашняя страница, на которой отображается имя аккаунта (если человек пользователь зашел на этот аккаунт);
* Librarian – библиотекарь, страница со списком сотрудников;
* Book – список книг;
* Visitor – раздел, где можно посмотреть всех читателей;
* Login – вкладка, на которой можно войти в аккаунт;
* Register – вкладка, где можно зарегистрироваться



## Регистрация

На сайте можно зарегистрироваться (при этом сайт может запомнить пользователя), так же можно зайти на уже существующий аккаунт

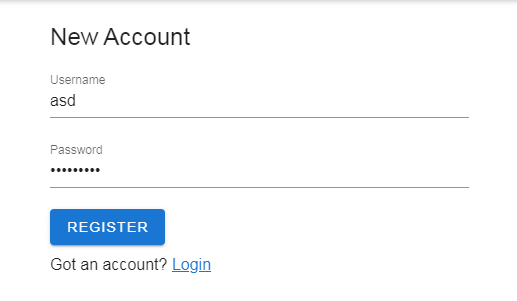


Рис 1 – регистрация нового пользователя

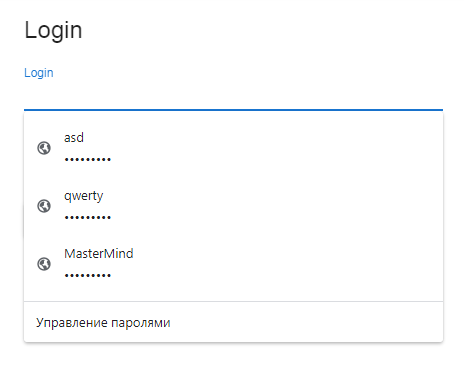


Рис.2 – сохраненные на сайте аккаунты

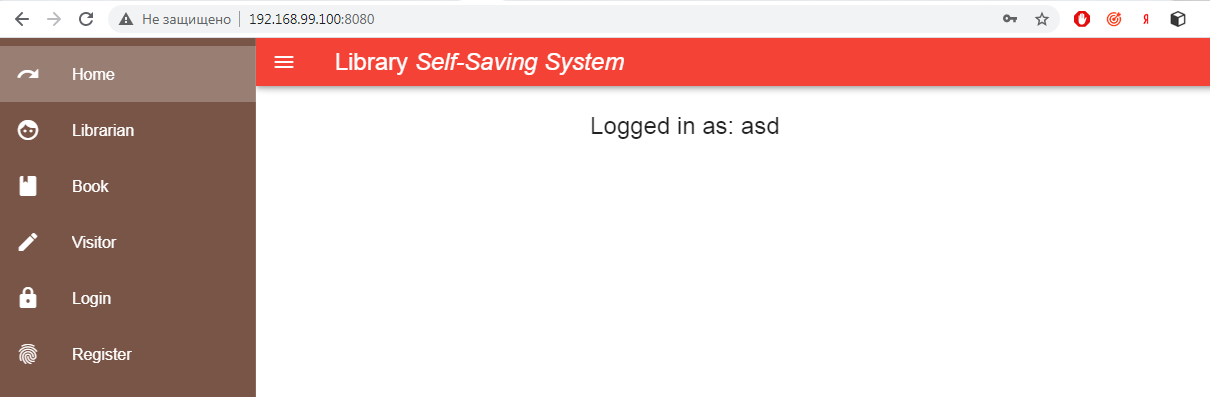


Рис.3 – вход за созданного пользователя

## Раздел «Книга»

В данном разделе выведен список книг (а также их количество), которые находятся в библиотеке. В карточке книги указаны ее номер, автор, название, год издания и издательство, количество экземпляров, а также ее статус — в качестве статуса выставляются буквы A (доступна), L (потеряна), D (списана).

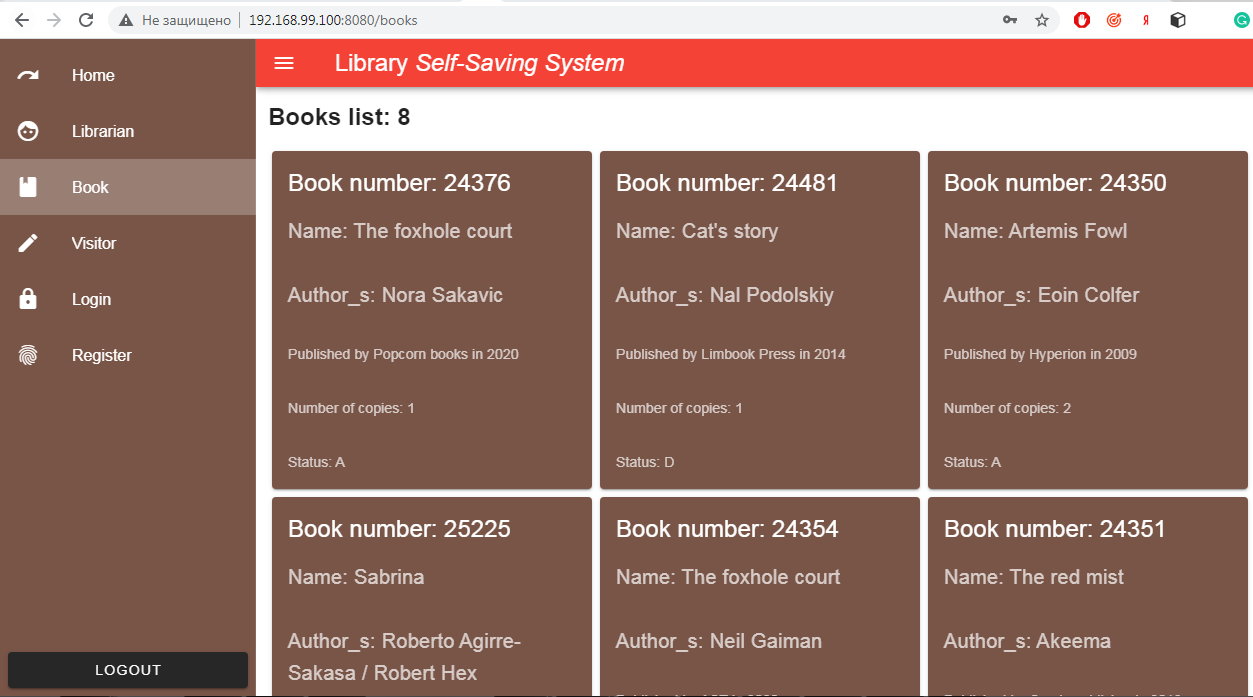


Рис. 4 – скриншот со страницы «/books»

В этом же разделе есть возможность добавить новую книгу.

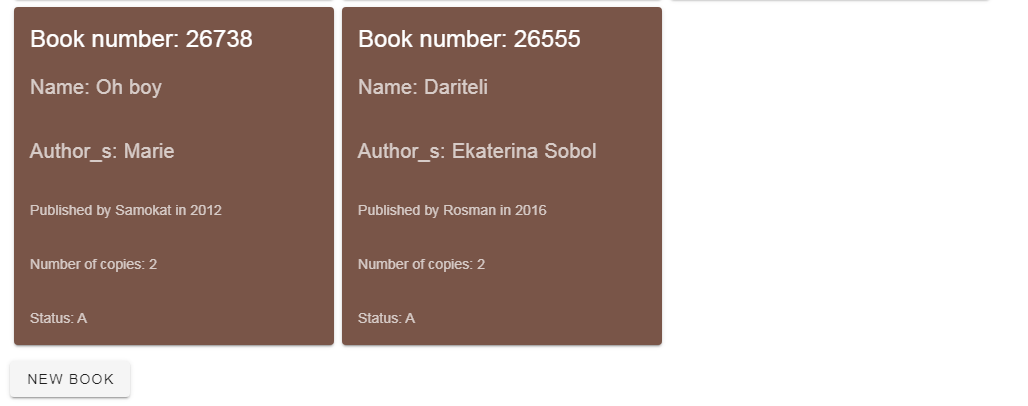


Рис. 5 – в конце списка книг расположена кнопка «новая книга»

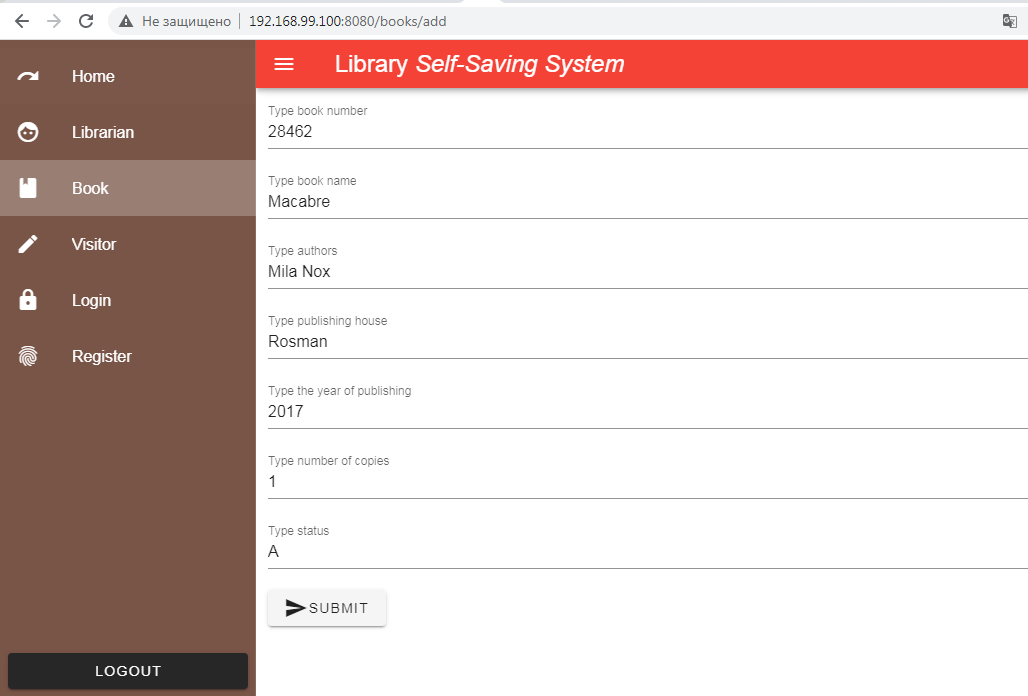


Рис. 6 – ввод данных новой книги



Рис. 7 – новая книга добавилась в конце списка книг

## Разделы «Посетитель» и «Библиотекарь»

В данных разделах выводятся списки читателей и библиотекарей, а также их данные.

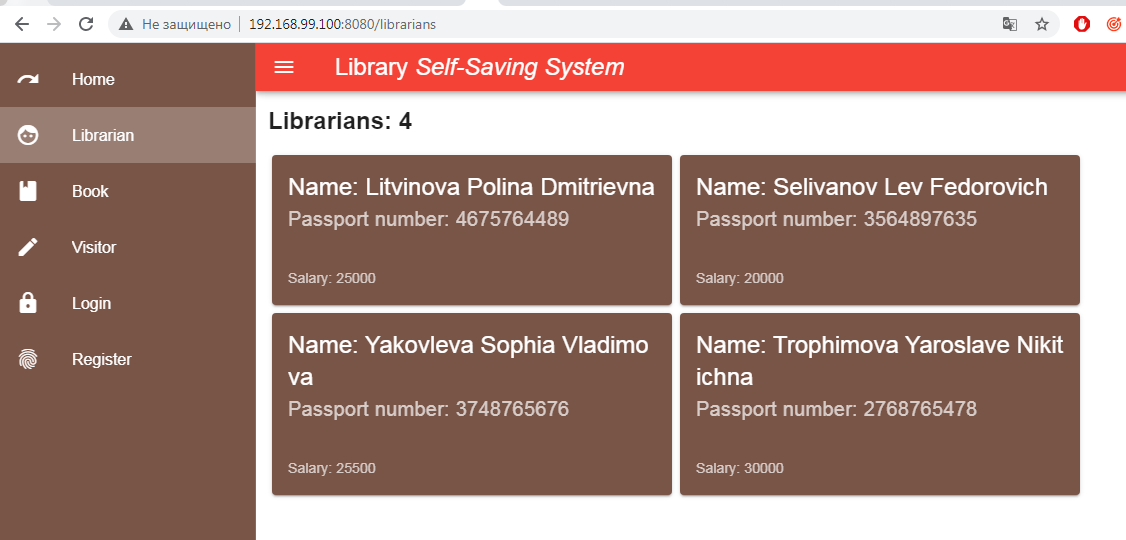


Рис. 8 – страница /lidrarians

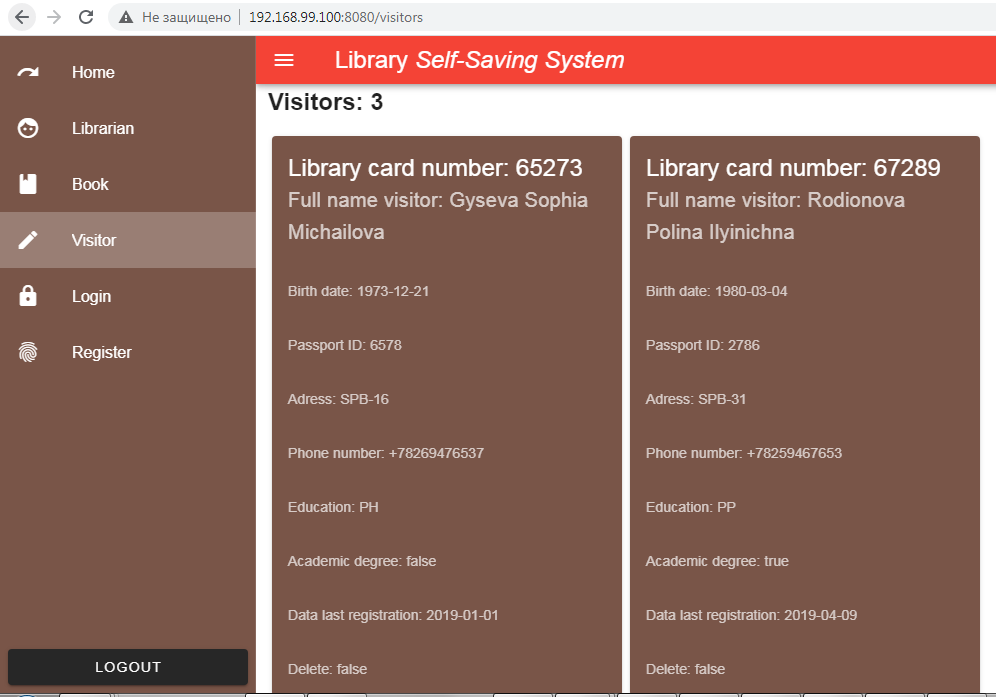


Рис. 9 – страница /visitors

## Страница администратора

После создания базы данных был добавлен администратор (супервользователь) с доступом к странице <http://192.168.99.100:8000/admin/>. На данной странице пользователь может добавлять в объекты в любой отдел, а также влиять на зарегистрировавшихся пользователей.

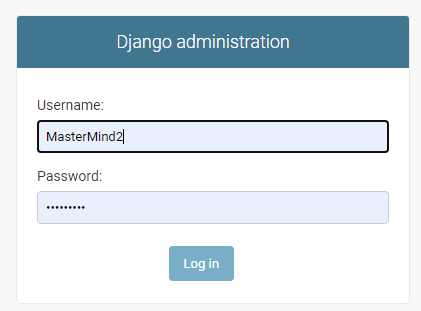


Рис. 10 – вход за администратора

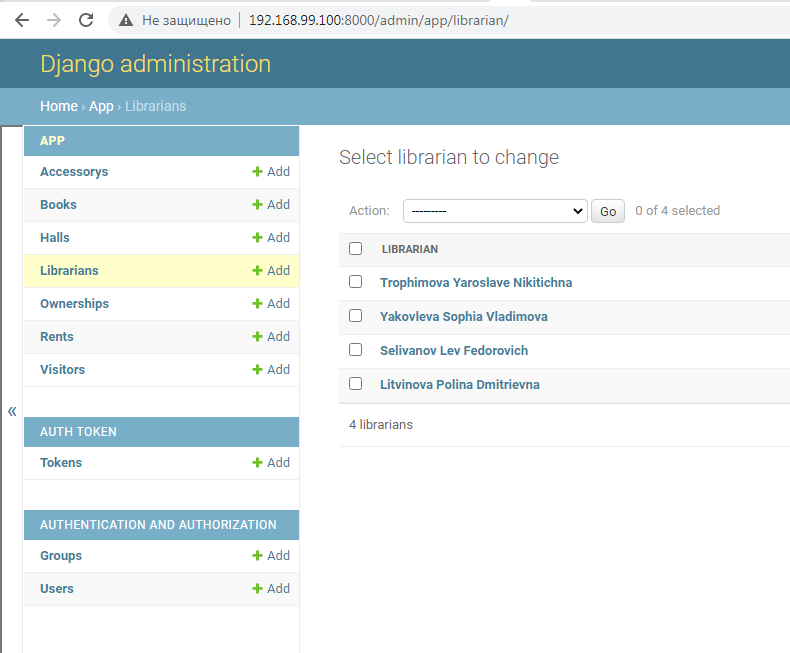


Рис. 11 – раздел библиотекарей

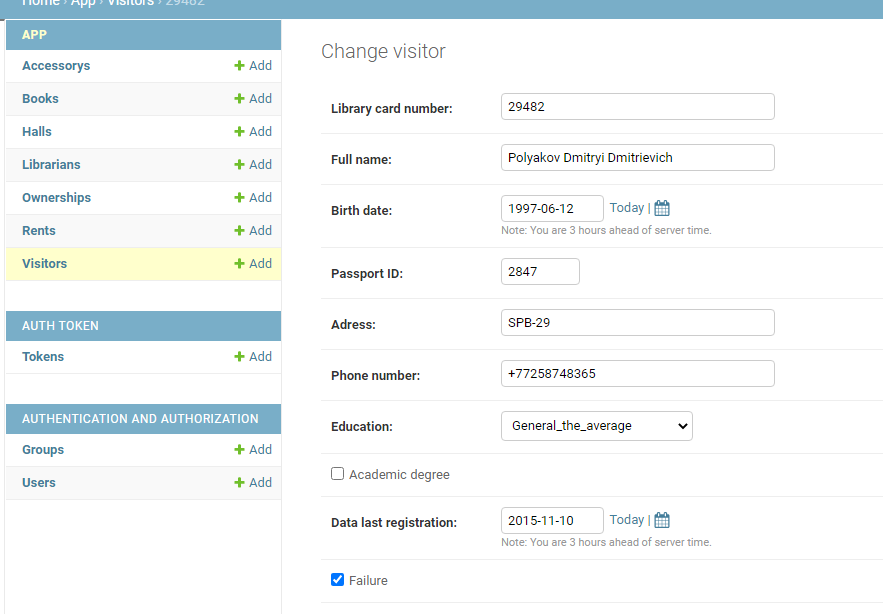


Рис. 12 – ввод/редактура одного из читателей (посетителей)

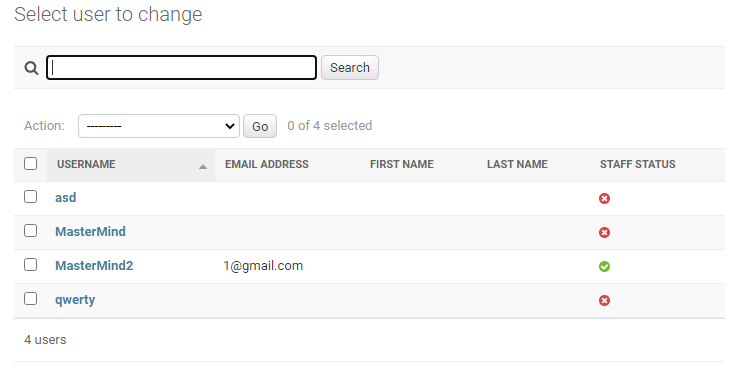


Рис. 13 - список всех зарегистрированных пользователей

# Заключение:

В ходе выполнения данной работы была изучена такая предметная область как «библиотека» и выявлены основные функции, которые можно было реализовать в веб-приложении. За время создания этого приложения были изучены основные знания, необходимые для написания проекта и приложения на Django с использованием Vue.js и Docker.

С помощью этих средств было создано веб-приложение с базой данных формата db. В ней же были реализованы функции вывода информации в виде списка, а также добавление новых со стороны пользователя. Также была реализована страница админа, где установленный за время выполнения работы суперпользователь мог создавать новые записи, а также редактировать и удалять уже созданные.

Кроме того, по итогу были получены навыки создания простого web-сервиса средствами языка программирования Python с помощью фреймворка Django.

1. Открытое программное обеспечение — программное обеспечение с открытым исходным кодом. Исходный код таких программ доступен для просмотра, изучения и изменения, что позволяет пользователю принять участие в доработке самой открытой программы, использовать код для создания новых программ и исправления в них ошибок — через заимствование исходного кода, если это позволяет совместимость лицензий, или через изучение использованных алгоритмов, структур данных, технологий, методик и интерфейсов (поскольку исходный код может существенно дополнять документацию, а при отсутствии таковой сам служит документацией). [↑](#footnote-ref-1)