**ОТЧЕТ**

о выполнении практики по теме:

«Разработка системы тестирования слушателей»

Выполнил:

Коротаев И. Е.\_\_\_\_\_\_\_

(подпись)

Проверил:

Ляховец Д. С.\_\_\_\_\_\_\_\_

(подпись)

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва

2020 г.

**Оглавление**

[Оглавление 2](#_Toc40665157)

[**Введение 3**](#_Toc40665158)

[**1. План разработки приложения 4**](#_Toc40665159)

[**2. Выявление существующих требований 4**](#_Toc40665160)

[**3. Выбор инструментов 5**](#_Toc40665161)

[**4. Разработка 6**](#_Toc40665162)

[**4.1 Авторизация пользователей, разграничение прав 6**](#_Toc40665163)

[**4.2 Добавление учебных предметов и пользователей в систему 6**](#_Toc40665164)

[**4.3 Фронтэнд 7**](#_Toc40665166)

[**4.4 Добавление, редактирование и удаление тестов 8**](#_Toc40665167)

[**4.5 Добавление, загрузка и удаление вопросов 9**](#_Toc40665168)

[**4.6 Запуск и прохождение тестов 10**](#_Toc40665169)

[**5. Тестирование 11**](#_Toc40665170)

[**6. Деплой 12**](#_Toc40665171)

[**Заключение 14**](#_Toc40665172)

## Введение

При выполнении практики стоит задача разработать микросервис тестирования слушателей. Учитывая, что на кафедре отсутствуют внедренные системы тестирования, которые можно было бы обобщить на все предметы, работа является актуальной.

## План разработки приложения

Разработка приложения может быть разбита на 8 основных этапов:

1. Выявление существующих требований к разрабатываемой системе тестирования слушателей Quizer.
2. Определение формата хранения вопросов по различным дисциплинам с учётом вопросов и ответов в виде картинок с использованием СУБД MongoDB.
3. Реализация возможности авторизации слушателей и преподавателей в системе.
4. Реализация возможности добавления и редактирования тестов с помощью веб-интерфейса.
5. Реализация возможности добавления, редактирования и удаления вопросов к тестам с помощью веб-интерфейса.
6. Реализация функций запуска существующих тестов и сбора информации об их прохождении.
7. Тестирование приложения.
8. Деплой приложения.

## Выявление существующих требований

Требования к системе Quizer можно определить в следующих пунктах:

1. Возможности авторизации слушателей и преподавателей в системе с разграничением прав.
2. Возможность добавления учебных предметов и пользователей в систему.
3. Возможность добавления, редактирования и удаления тестов по различным предметам с помощью веб интерфейса.
4. Возможность добавления и удаления вопросов с помощью веб интерфейса, загрузки вопросов из файлов определенного формата. При этом реализовать:

* возможность добавлять вопросы различных типов - с мультивыбором, изображениями и переменным количеством вариантов ответов;
* разработать схему хранения вопросов с изображениями.

1. Возможность запуска существующих тестов для их прохождения слушателями.
2. Реализация выборки случайных вопросов для каждого слушателя для прохождении теста.
3. Реализация прохождения тестов слушателями и записи результатов тестирования базу.

## Выбор инструментов

Для написания вэб-приложения воспользуемся фреймворком Django, потому что:

1. Модульная структура обеспечит возможность дальнейшего стороннего совершенствования системы.
2. В нем «из коробки» доступны многие полезные функции, такие как панель администрирования, система авторизации пользователей, расширяемая система шаблонов с тегами и наследованием и тд.
3. Он удобный и простой в освоении.

Для реализации фронтэнда приложения воспользуемся фреймворком Bootstrap4, потому что:

1. Легко интегрируется в проект – достаточно просто скачать исходники с сайта и добавить их к статическим файлам проекта.
2. Богатый набор готовых решений, которые можно использовать в своем проекте.

В качестве БД выберем СУБД MongoDB, потому что:

1. Основные модели данных в проекте – вопросы и результаты тестирования – являются сильно вложенными данными, и потому реляционная модель представления для них не подходит.
2. MongoDB является документоориентированной СУБД, что делает ее идеальным решением для данного приложения.

Для тестирования приложения и измерения покрытия кода тестами будем использовать инструмент с открытым исходным кодом – coverage, так как:

1. Он имеет возможность работать поверх тестов, интегрированных в Django.
2. Легко конфигурируется.
3. Имеет опцию выдачи подробного отчета по покрытию кода тестами в удобочитаемом вэб формате.

Для статического анализа кода воспользуемся программным обеспечением pylint.

Для деплоя приложения на рабочем сервере воспользуемся системой контейнеризации Docker. Также реализуем 2 сборочных скрипта (под Линукс и Виндовс) для возможности развертывания системы на хостовой машине.

## Разработка

### Авторизация пользователей, разграничение прав

В системе существует 3 группы пользователей:

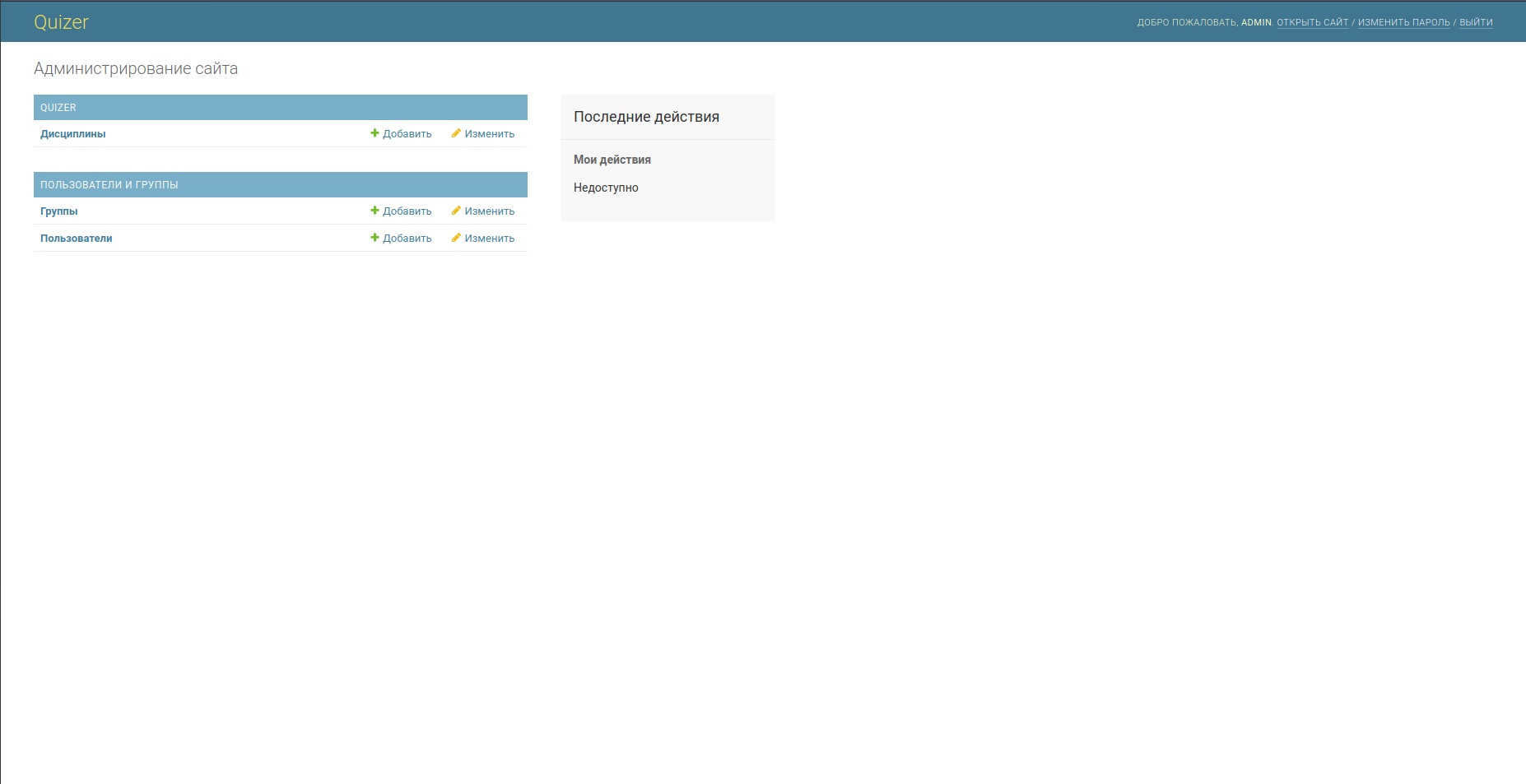
1. Студенты – имеют возможность запускать и проходить тесты.
2. Преподаватель – имеют возможность создавать, изменять, запускать и удалять тесты.
3. Суперпользователи – могут относиться к любой из 2 групп выше (но только одной) – в дополнение к возможностямй одной из вышеописанных групп могут заходить в панель администрирования приложения, где они могут добавлять новые предметы и новых пользователей.

Авторизация пользователей в системе производится встроенными инструментами Django.

### Добавление учебных предметов и пользователей в систему

Добавление новых учебных предметов и пользователей системы реализовано с помощью панели администрирования Django и доступно лишь суперпользователям. Создать суперпользователя можно двумя способами:

* с помощью графического интерфейса в панели администрирования;
* с помощью командной утилиты django-admin в корне проекта.



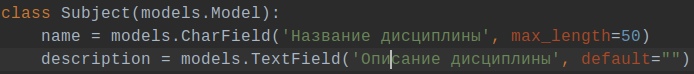


Рисунок 4.2.1 – Модель учебного предмета

## 

Рисунок 4.2.2 – Панель администрирования

### Фронтэнд

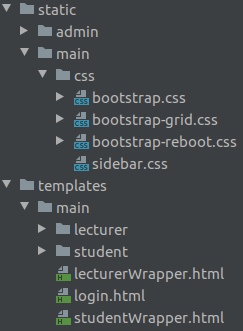
С учетом наличия в приложении групп пользователей 2 типов, была реализовано следующая структура, предоставляющая различный пользовательский интерфейс 2 группам пользователей.

Рисунок 4.3.1 – Структура фронтэнда проекта

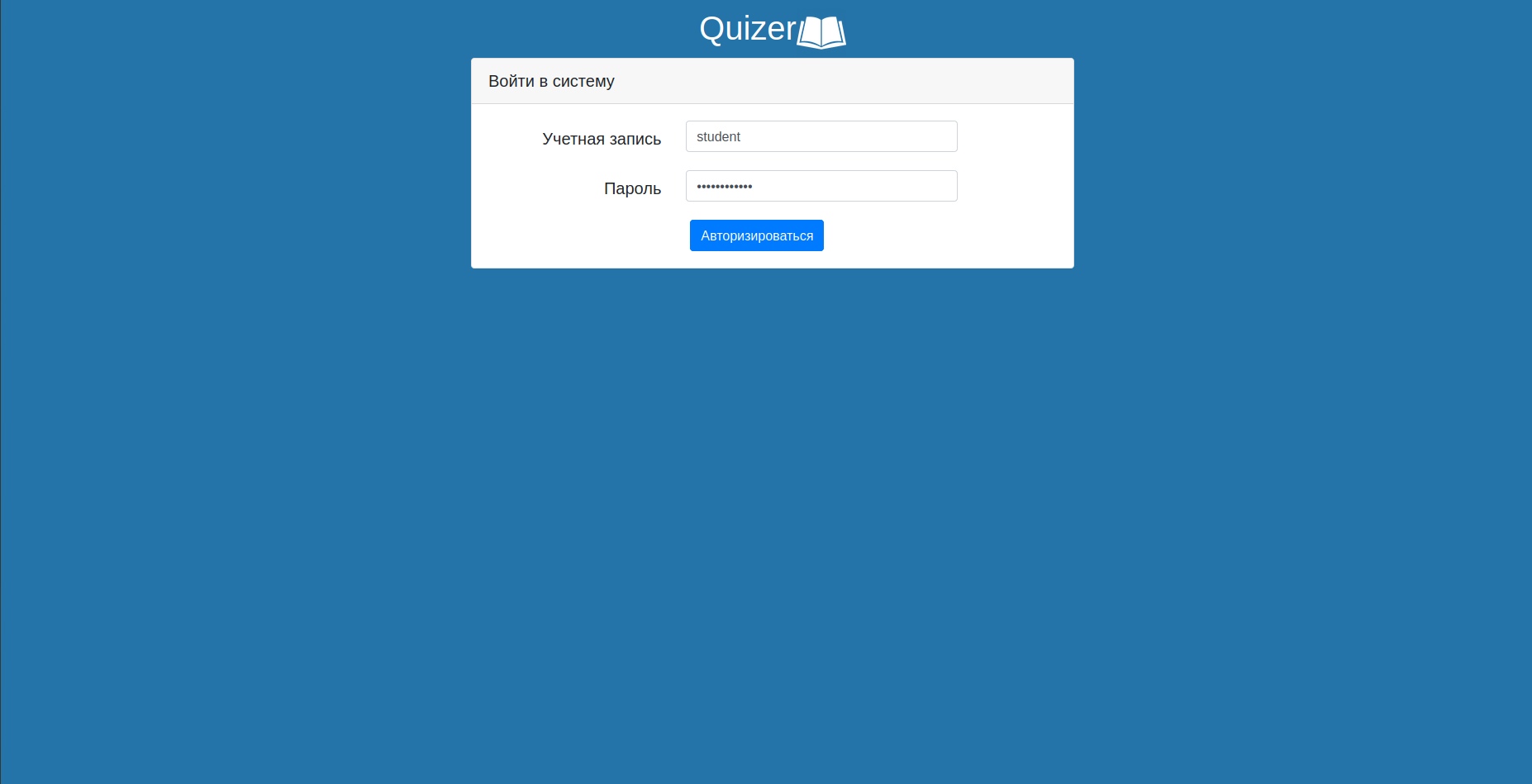


Рисунок 4.3.2 – Окно авторизации пользователей

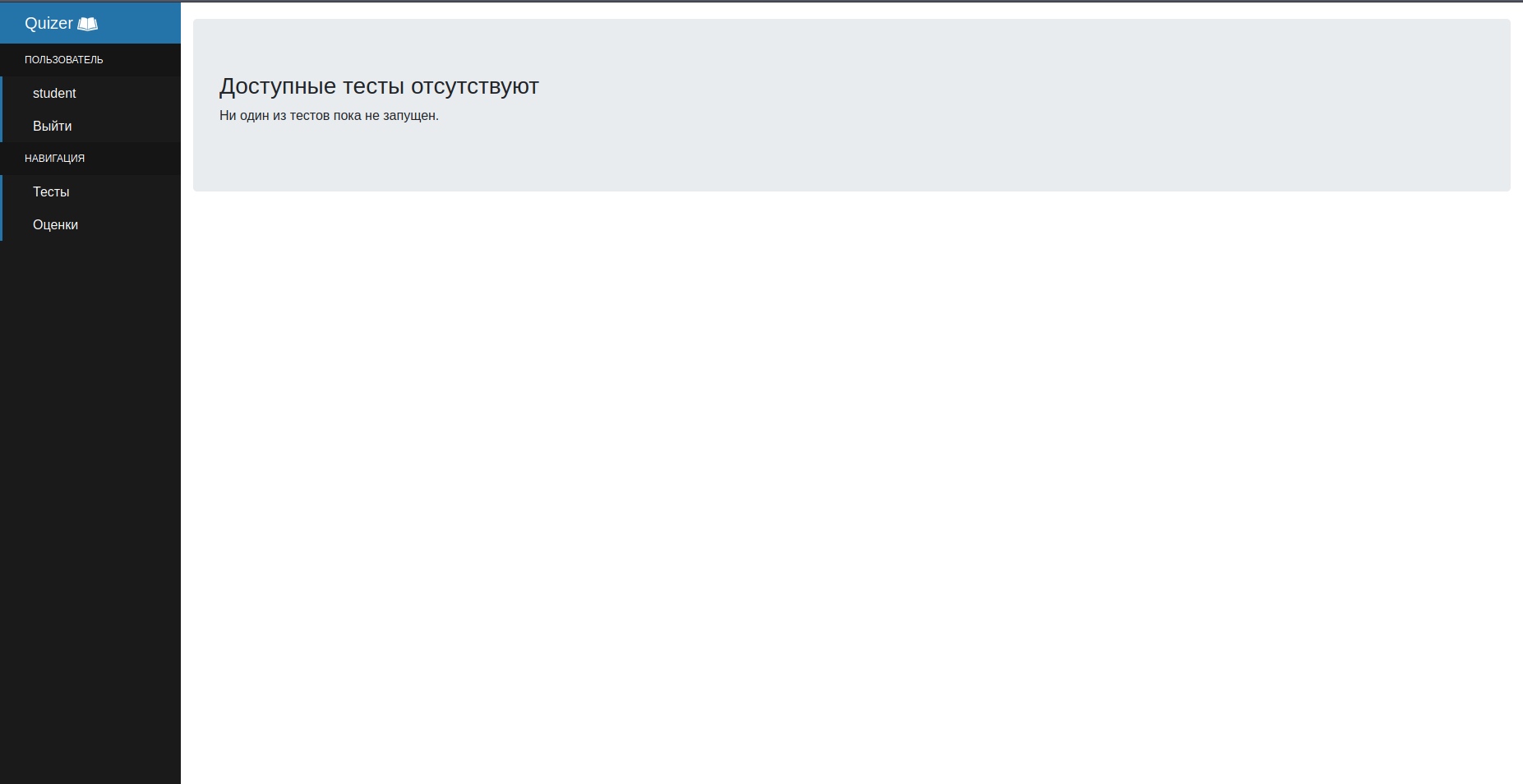


Рисунок 4.3.3 – Пользовательский интерфейс слушателя

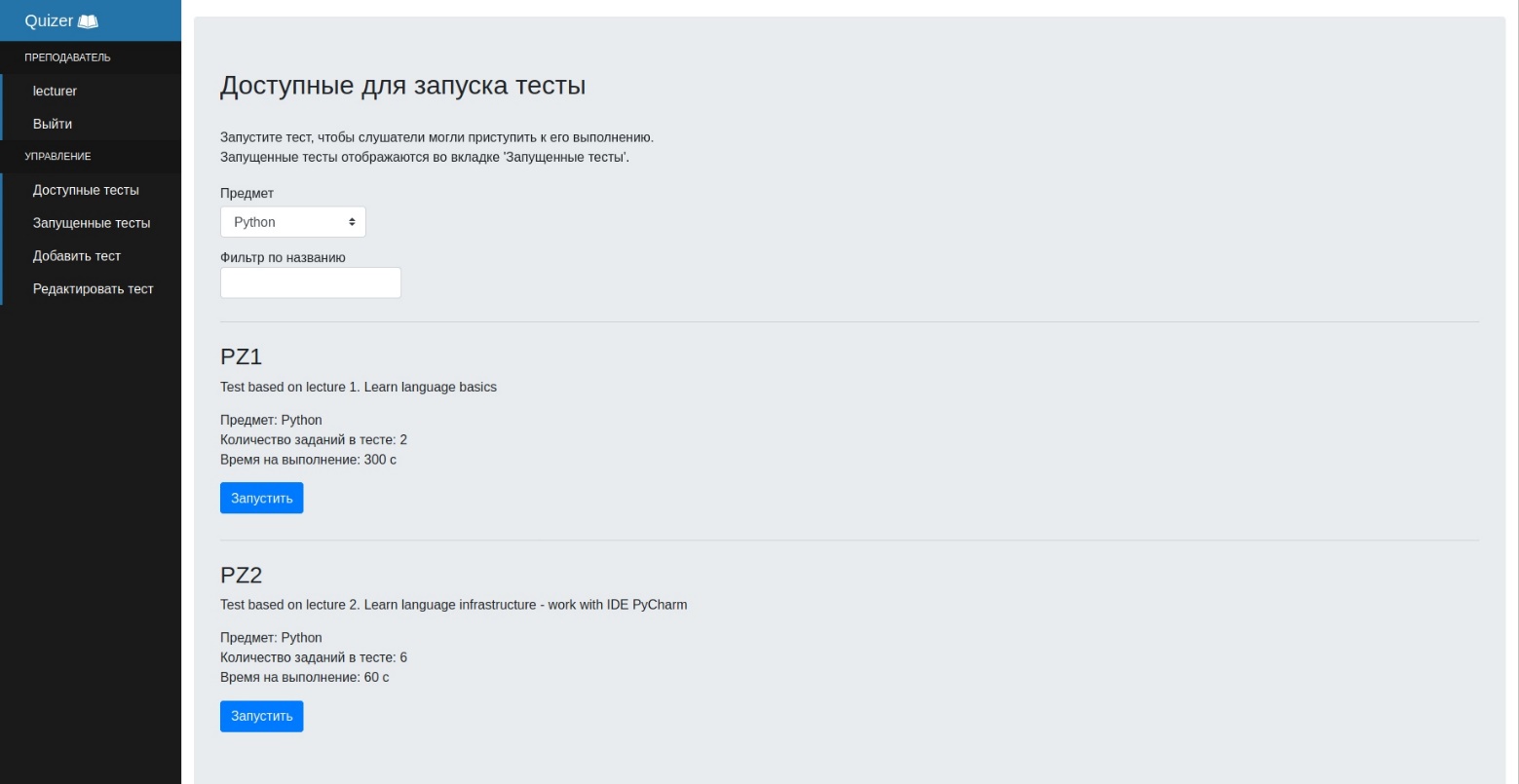


Рисунок 4.3.3 – Пользовательский интерфейс преподавателя

### Добавление, редактирование и удаление тестов

Была реализована возможность добавления, редактирования и удаления тестов с помощью веб-интерфейса.

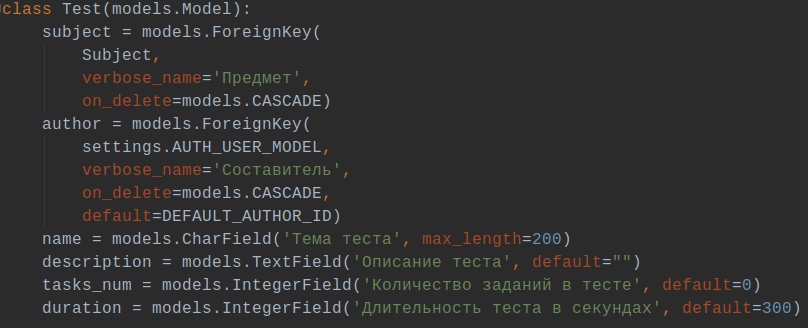


Рисунок 4.4.1 – Модель теста

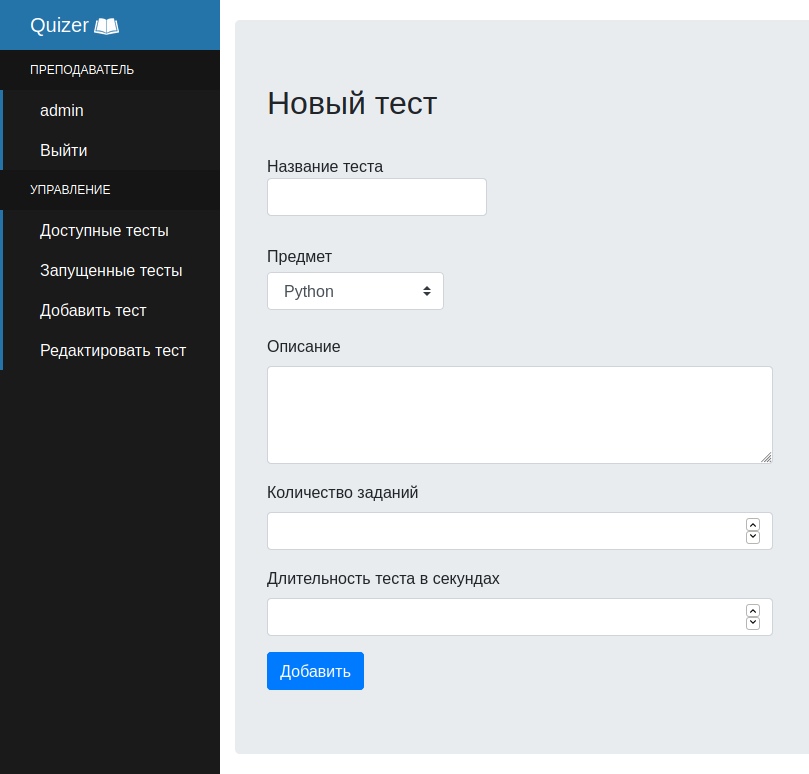


Рисунок 4.4.2 – Добавление нового теста с помощью веб-интерфейса

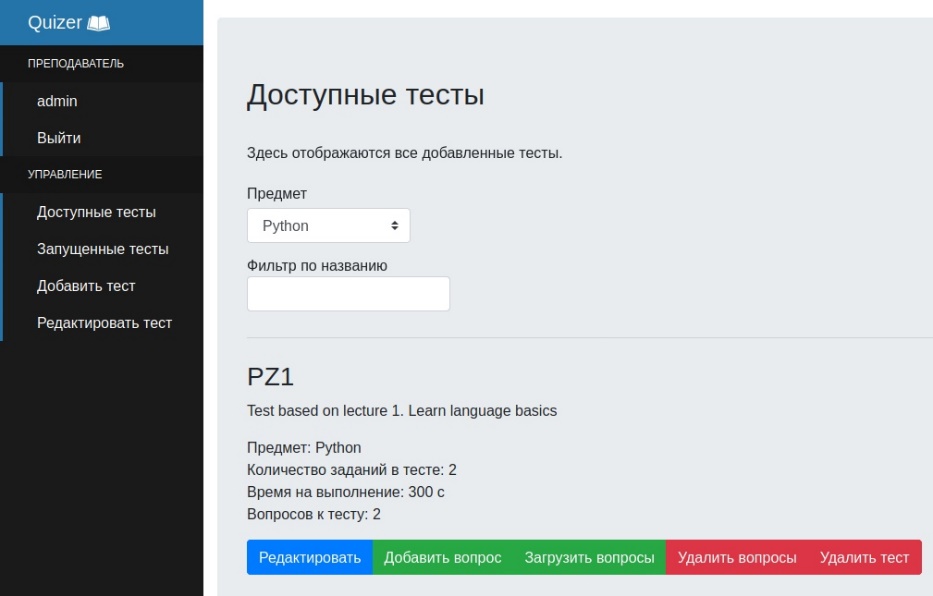


Рисунок 4.4.3 – Варианты редактирования тестов с помощью веб-интерфейса

Также с помощью вставок кода на JavaScript на странице редактирования всех доступных тестов была добавлена возможность выбора нужного теста по предмету и названию.

### Добавление, загрузка и удаление вопросов

Была продумана и реализована схема хранения вопросов в БД с учетом следующих требований:

* + вопросы могут быть с 1 ответом или с мультивыбором;
  + вопросы могут быть с изображениями и без.



Рисунок 4.5.1 – Пример вопросов 2 типов – с изображениями и без

Для вопросов с изображениями был выбрал следующий порядок хранения – для них в вариантах ответов хранятся расположения изображений на сервере. Например, изображения к вопросу {question} по тесту {test} по предмету {subject} с {options\_count} вариантами ответов хранятся по адресу quizer/media/{subject.name}/{test.name}/{str(question.\_id})/[0-{options\_count - 1}].jpg. Все папки создаются и удаляются автоматически.

Также были реализованы следующие возможности:

* + вопросы можно загружать из файлов определенного формата;
  + вопросы можно добавлять с помощью веб-интерфейса;
  + вопросы можно удалять с помощью веб-интерфейса.

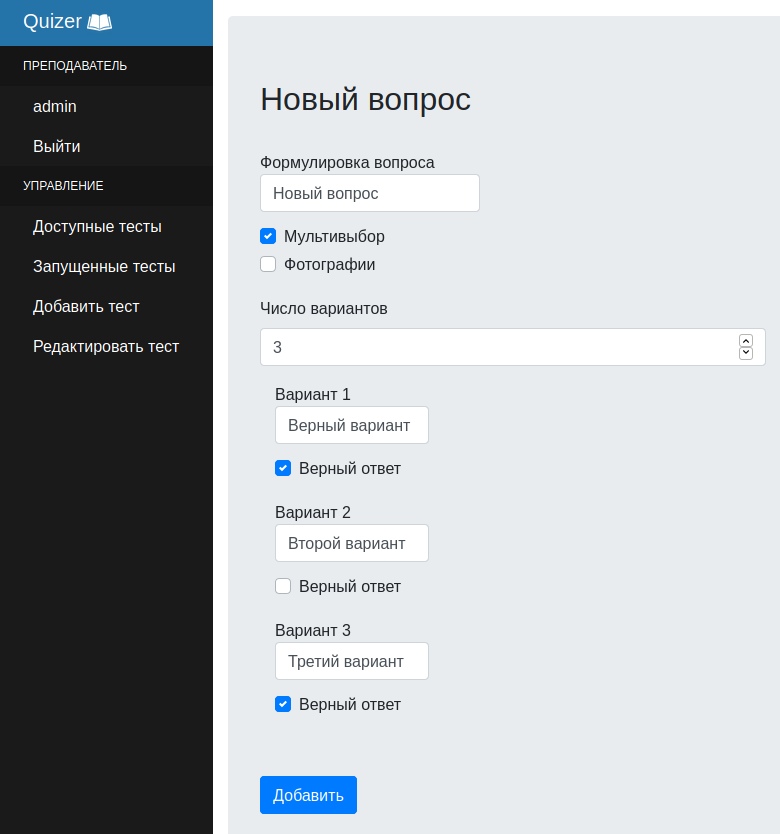


Рисунок 4.5.2 – Добавление нового вопроса с помощью веб-интерфейса

### Запуск и прохождение тестов

Реализована возможность запуска существующих тестов для их прохождения слушателями у преподавателя есть определенный список тестов, из которых он может запустить любой на выбор. После запуска теста он появится у слушателей во вкладке «Тесты», после чего последние смогут приступить к его выполнению. Преподаватель сможет отслеживать процесс прохождения теста на вкладке «Запущенные тесты».

Были реализованы следующие требования:

* + случайная выборка вопросов для каждого слушателя;
  + случайный порядок вариантов ответов;
  + запись результатов тестирования в базу.

Результаты прохождения теста записываются в базу, и после того, как преподаватель останавливает запущенный тест (с помощью веб-интерфейса), он может просмотреть результаты прохождения теста каждым слушателем (в том числе увидеть допущенные ими ошибки) с помощью веб-интерфейса.

## Тестирование

Тестировать приложение будем с использованием unit тестов. Django предоставляет фреймворк для создания тестов, построенного на основе иерархии классов, которые, в свою очередь, зависят от стандартной библиотеки Python unittest. Были реализованы следующие классы для тестирования приложения:

* + MainTest(TestCase) – базовый класс для всех тестов, в котором реализован метод setUp(), добавляющий данные в тестовую базу данных перед каждым тестом;
  + QuestionStorageTest(MainTest) – тесты хранилища вопросов;
  + AuthorizationTest(MainTest) – тесты авторизации пользователей;
  + AccessRightsTest(MainTest) – тесты разграничения доступа для 2 групп пользователей;
  + TestAddingTest(MainTest) – тесты добавления новых тестов;
  + TestEditingTest(MainTest) – тесты изменения существующих тестов;
  + LoadingQuestionsTest(MainTest) – тесты загрузки вопросов из файлов заданного формата;
  + AddQuestionTest(MainTest) – тесты добавления новых вопросов;
  + TestsResultsStorageTest(MainTest) – тесты хранилища результатов тестирования;
  + RunningTestsAnswersStorageTest(TestsResultsStorageTest) – тесты временного хранилища ответов на тесты слушателей (тестирование прохождения тестов слушателями).

Для оценки покрытия кода тестами был использован инструмент coverage.

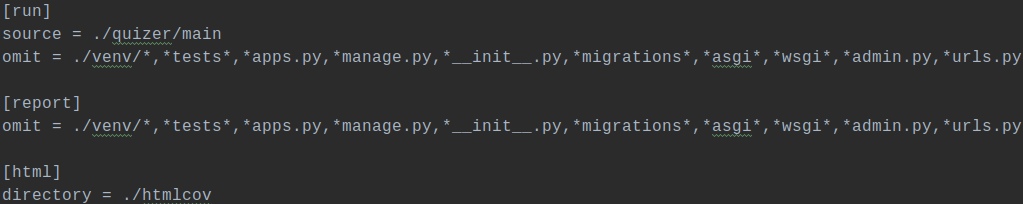


Рисунок 5.1 – Конфигурация coverage – файл . coveragerc

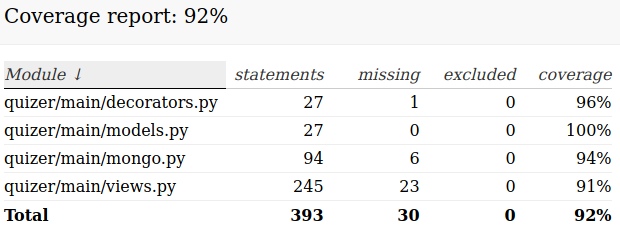


Рисунок 5.2 – Покрытие кода приложения тестами

## Деплой

Развернуть приложение на предоставленном сервере можно двумя способами:

* + как самостоятельное приложение;
  + в качестве докер контейнера.

Первый вариант не является предпочтительным, однако предоставляет возможность собрать работоспособное приложение с целью его доработки и усовершенствования. Собрать приложение можно с помощью сборочных скриптов – build\_for\_linux или build\_for\_win.ps1 в зависимости от системы. Из зависимостей требуется установленный python3 и MongoDB. Тогда процесс сборки приложения и его запуска уложится в 5 команд:

* + *git clone https://github.com/LeadNess/Quizer.git*
  + *cd Quizer*
  + *./deploy/build\_for\_linux* или *powershell .\deploy\build\_for\_win.ps1*
  + *source ./venv/bin/activate* или *.\venv\Scripts\activate*
  + *python ./quizer/manage.py runserver* или *python .\quizer\manage.py runserver*

Сборочный скрипт создаст виртуальное окружение, активирует его, сгенерирует новый секретный ключ приложения, запросит с клавиатуры конфигурации MongoDB (хост, порт и имя базы данных), применит миграции, добавит основные группы пользователей и создаст суперпользователя.

Вариант развертывая приложения в качестве докер-контейнера более предпочтителен. Тогда развернуть приложение на сервере можно в 2 этапа. Во-первых, необходимо собрать докер-образ приложения и создать контейнер на его базе:

* + *git clone https://github.com/LeadNess/Quizer.git*
  + *cd Quizer*
  + *docker build -t quizer .*
  + *docker run -p 80:80 --name testing-app quizer*

Dockerfile приведен в Листинге 6.1.

Листинг 6.1

FROM ubuntu:latest

MAINTAINER LeadNess

RUN apt-get update && apt-get install -y build-essential python3 \

&& apt-get install -y python3-setuptools \

&& apt-get install -y python3-pip

RUN apt-get update && apt-get install -y mongodb \

&& mkdir -p /data/db \

&& mkdir -p /data/code

COPY requirements.txt /app/requirements.txt

RUN pip3 install --no-cache-dir -r /app/requirements.txt

COPY quizer /app/quizer

# MongoDB port

EXPOSE 27017

# App port

EXPOSE 80

COPY deploy/entrypoint /entrypoint

ENTRYPOINT ["/entrypoint"]

Во-вторых, необходимо наладить автоматическое возобновление работы приложения после перезагрузки сервера. Ubuntu позволяет решить эту задачу силами системы инициализации systemd. Необходимо описать конфигурацию приложения в отдельном файле /etc/systemd/system/quizer.service. Затем юнит можно запустить, а также включить для него автозагрузку. Конфигурация приложения приведена в Листинге 6.2.

Листинг 6.2

[Unit]

Description=Quizer testing tool

Requires=docker.service

After=docker.service

[Service]

Type=oneshot

RemainAfterExit=true

ExecStart=/usr/bin/docker start testing-app

ExecStop=/usr/bin/docker stop testing-app

[Install]

WantedBy=multi-user.target

## Заключение

В ходе выполнения практики была разработана полноценная система тестирования, которая может применятся для организации тестирования слушателей по учебным предметам в течении учебного процесса. Также, учитывая специфику сбора информации о пройденных тестах данной системой, можно будет анализировать качество определенных тестов и вопросов, успеваемость определенных групп и слушателей.