

# 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
ВВЕДЕНИЕ .....	3
1. ПЛАН РАЗРАБОТКИ ПРИЛОЖЕНИЯ.....	4
2. ВЫЯВЛЕНИЕ СУЩЕСТВУЮЩИХ ТРЕБОВАНИЙ .....	4
3. ВЫБОР ИНСТРУМЕНТОВ.....	5
4. РАЗРАБОТКА .....	6
4.1 Авторизация пользователей, разграничение прав .....	6
4.2 Добавление учебных предметов и пользователей в систему.....	6
4.3 Фронтэнд .....	7
4.4 Добавление, редактирование и удаление тестов .....	8
4.5 Добавление, загрузка и удаление вопросов.....	9
4.6 Запуск и прохождение тестов .....	10
5. ТЕСТИРОВАНИЕ.....	11
6. ДЕПЛОЙ .....	12
ЗАКЛЮЧЕНИЕ .....	14

## **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.

## **1. 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.

## **2. 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.
5. Возможность запуска существующих тестов для их прохождения слушателями.
  6. Реализация выборки случайных вопросов для каждого слушателя для прохождении теста.
  7. Реализация прохождения тестов слушателями и записи результатов тестирования базу.

### **3. 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 веб 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.

## **4. Разработка**

### **4.1 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 возможностям 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.

### **4.2 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.

```
class Subject(models.Model):
    name = models.CharField('Название дисциплины', max_length=50)
    description = models.TextField('Описание дисциплины', default="")
```

Рисунок 4.2.1 – Модель учебного предмета

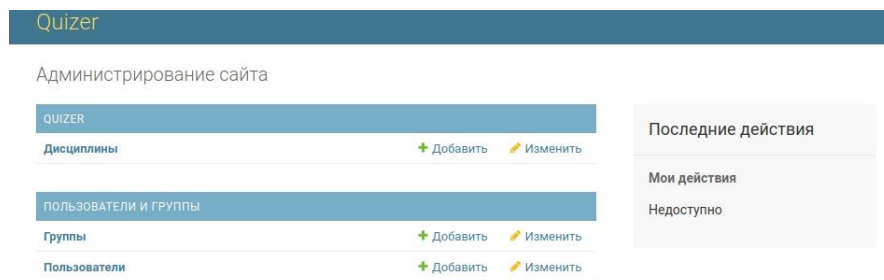


Рисунок 4.2.2 – Панель администрирования

### 4.3 Фронтэнд

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

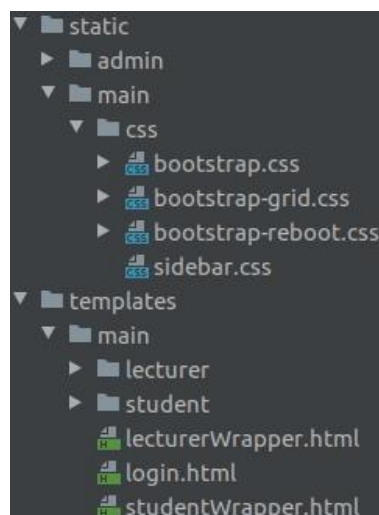


Рисунок 4.3.1 – Структура фронтэнда проекта

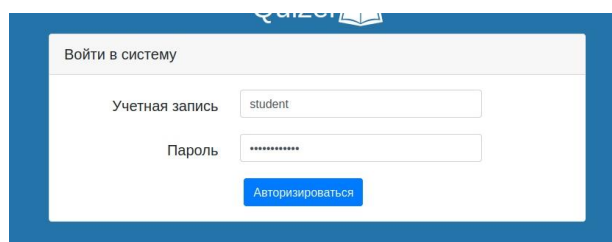


Рисунок 4.3.2 – Окно авторизации пользователей

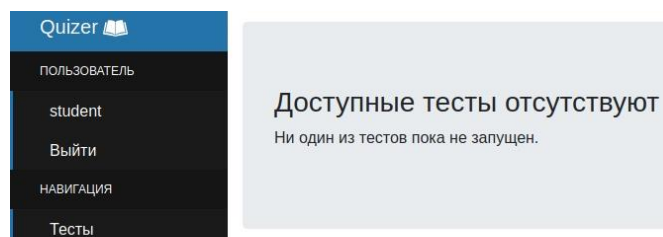


Рисунок 4.3.3 – Пользовательский интерфейс слушателя

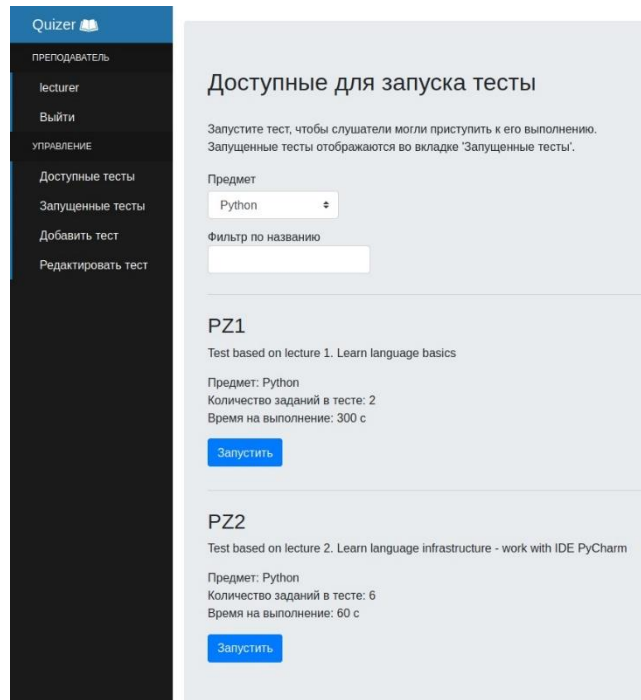


Рисунок 4.3.3 – Пользовательский интерфейс преподавателя

## 4.4 Добавление, редактирование и удаление тестов

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

```
class Test(models.Model):
    subject = models.ForeignKey(
        Subject,
        verbose_name='Предмет',
        on_delete=models.CASCADE)
    author = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        verbose_name='Составитель',
        on_delete=models.CASCADE,
        default=DEFAULT_AUTHOR_ID)
    name = models.CharField('Тема теста', max_length=200)
    description = models.TextField('Описание теста', default="")
    tasks_num = models.IntegerField('Количество заданий в тесте', default=0)
    duration = models.IntegerField('Длительность теста в секундах', default=300)
```

Рисунок 4.4.1 – Модель теста

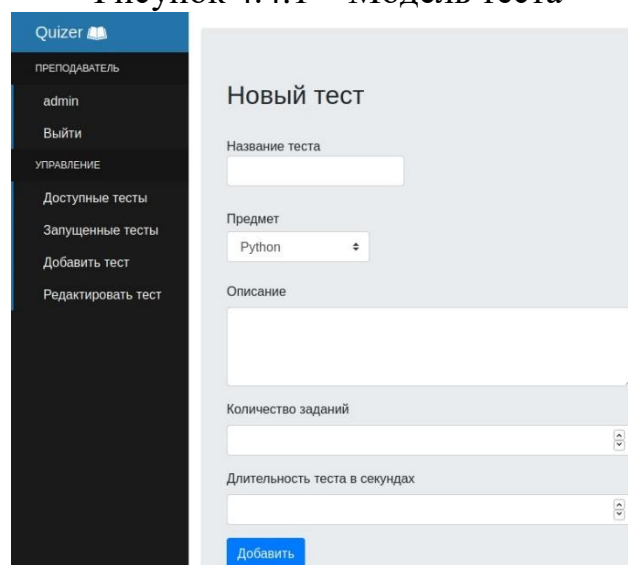


Рисунок 4.4.2 – Добавление нового теста с помощью веб-интерфейса



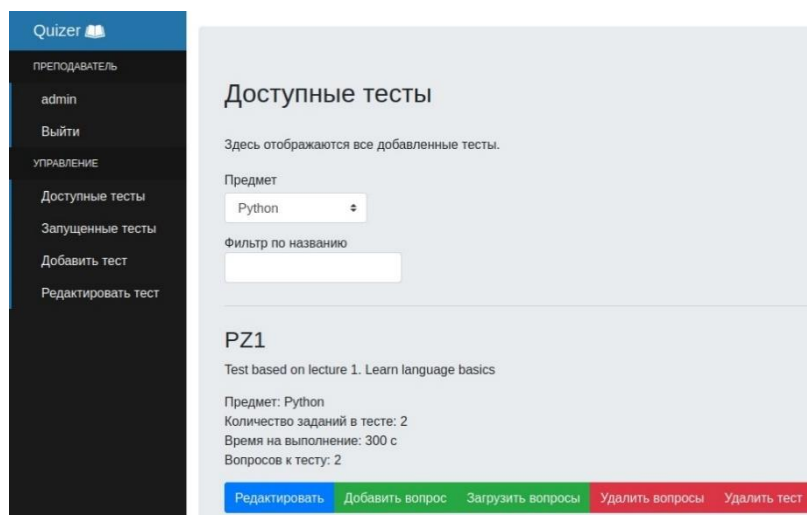


Рисунок 4.4.3 – Варианты редактирования тестов с помощью веб-интерфейса

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

## 4.5 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.

```
{
  "_id": {"$oid": "5e8a64a2b5633670bb89db2b"},
  "formulation": "Multiselect(1, 3)",
  "multiselect": true,
  "options": [
    {
      "option": "1",
      "is_true": true
    },
    {
      "option": "2",
      "is_true": false
    },
    {
      "option": "3",
      "is_true": true
    }
  ],
  "tasks_num": "3",
  "test_id": 1,
  "with_images": false
},
{
  "_id": {"$oid": "5e8b439e6a6bcfa266a4d7ff"},
  "formulation": "Question with images(1, 2)",
  "multiselect": true,
  "options": [
    {
      "option": "Python/П31/5e8b439e6a6bcfa266a4d7ff/0.jpg",
      "is_true": true
    },
    {
      "option": "Python/П31/5e8b439e6a6bcfa266a4d7ff/1.jpg",
      "is_true": true
    },
    {
      "option": "Python/П31/5e8b439e6a6bcfa266a4d7ff/2.jpg",
      "is_true": false
    }
  ],
  "tasks_num": "3",
  "test_id": 1,
  "with_images": true
},
```

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

Для вопросов с изображениями был выбран 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.

The screenshot shows the Quizer web application interface. On the left is a dark sidebar with the 'Quizer' logo and a menu for the 'ПРЕПОДАВАТЕЛЬ' (Teacher) role, including 'admin', 'Выйти' (Logout), and a 'УПРАВЛЕНИЕ' (Management) section with links for 'Доступные тесты' (Available tests), 'Запущенные тесты' (Running tests), 'Добавить тест' (Add test), and 'Редактировать тест' (Edit test). The main content area is titled 'Новый вопрос' (New question). It contains a text input for 'Формулировка вопроса' (Question formulation) with the placeholder 'Новый вопрос'. Below this are two radio buttons: 'Мультивывбор' (Multiselect) which is selected, and 'Фотографии' (Photos). A dropdown menu for 'Число вариантов' (Number of options) is set to '3'. There are three sections for options: 'Вариант 1' (Option 1) with text 'Верный вариант' (Correct option) and a selected 'Верный ответ' (Correct answer) checkbox; 'Вариант 2' (Option 2) with text 'Второй вариант' (Second option) and an unselected 'Верный ответ' checkbox; and 'Вариант 3' (Option 3) with text 'Третий вариант' (Third option) and a selected 'Верный ответ' checkbox. A blue 'Добавить' (Add) button is at the bottom.

Рисунок 4.5.2 – Добавление нового вопроса с помощью веб-интерфейса

## 4.6 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.

## 5. 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.

```
[run]
source = ./quizer/main
omit = ./venv/*, *tests*, *apps.py, *manage.py, *__init__.py, *migrations*, *asgi*, *wsgi*, *admin.py, *urls.py

[report]
omit = ./venv/*, *tests*, *apps.py, *manage.py, *__init__.py, *migrations*, *asgi*, *wsgi*, *admin.py, *urls.py

[html]
directory = ./htmlcov
```

Рисунок 5.1 – Конфигурация coverage – файл .coveragerc

Coverage report: 92%

Module \	statements	missing	excluded	coverage
quizer/main/decorators.py	27	1	0	96%
quizer/main/models.py	27	0	0	100%
quizer/main/mongo.py	94	6	0	94%
quizer/main/views.py	245	23	0	91%
<b>Total</b>	<b>393</b>	<b>30</b>	<b>0</b>	<b>92%</b>

Рисунок 5.2 – Покрытие кода приложения тестами

## 6. 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/web-testing-tool.git`
- `cd web-testing-tool`
- `./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 развертывания 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/web-testing-tool.git*
- *cd web-testing-tool*
- *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 применяться 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.