

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра МКиИТ
Проектирование клиент-серверных приложений

Лабораторная работа №1
“Начало работы с Python”

Выполнил:
студент 3 курса,
группы БФИ2001
Лушин Е. А.

Москва 2023

Цель работы: изучить работу с Python 3.7.X через консоль, а также научиться создавать сервер через Django.

Задание:

- Скачать Python и создать директорию для лабораторных работ.
- Создать файл «mygroup.py», в котором будут находиться данные о студентах вашей группы (достаточно 5-7 человек) в формате списка. Обратите внимание, что весь список сохраняется в одну переменную (например, groupmates), для каждого студента должны быть заполнены поля: имя, фамилия, список экзаменов, оценки.
- Написать функцию фильтрации студентов, по средней оценке, (так чтобы функция возвращала всех студентов выше заданного в параметрах функции среднего балла). Примерная схема работы функции: создание пустого массива, куда будут добавляться все студенты, прошедшие фильтрацию; запуск цикла, в каждой итерации которого необходимо считать среднюю оценку текущего студента и сравнивать с тем значением, что передано в качестве параметра.
- Установить фреймворк Django, а также создать проект.
- Открыть файл **settings.py** через редактор исходного кода IDLE. Внимательно изучить все существующие там переменные и перевести описание каждой из них. Зубрить их нет необходимости, но примерно представлять совокупность всех настроек по умолчанию нужно.
- В файле **settings.py** настроить два параметра для базы данных. В переменной-словаре **DATABASES** есть ключ **default**, по которому содержится ещё один внутренний словарь с ключами **ENGINE** и **NAME**. Установить следующие значения этим параметрам:
 - **ENGINE:** *'django.db.backends.sqlite3'*
 - **NAME:** *'db_admin_learning'*
- Теперь для вашего проекта в качестве базы данных подключена **sqlite3**, а название новосозданной базы будет “**db_admin_learning**”.

- Изучить интерфейс административного приложения django.
- Через интерфейс административного приложения создать нового пользователя с правами суперюзера.
- Через интерфейс административного приложения создать нового пользователя без прав суперюзера.
- Через интерфейс административного приложения «забанить» одного из пользователей (сделать пользователя «неактивным»).

Краткая теория

Основы работы клиент-серверных приложений

Любое веб-приложение работает на клиент-серверной архитектуре — это значит, что у каждой программы (иными словами, у каждого сайта) есть две части — клиентская и серверная. Первая работает на компьютере непосредственно пользователя и реагирует на все пользовательские действия, вторая ждет от клиента запросов, по поступлению которых начинает формировать нужный ответ и отправлять его обратно клиенту. Обычно клиент нужен для обеспечения красивого интерфейса, который был бы удобен для конечного пользователя, а сервер для проведения основных вычислений и хранения информации. Чаще всего на множество клиентских приложений приходится всего несколько серверов (продемонстрировано на рисунке 1), что значительно облегчает обслуживание, ведь большинство ошибок будет, скорее всего, лишь на нескольких машинах, непосредственно принадлежащих производителю.



Рисунок 1 – Общая схема работы клиент-серверного приложения

У описанной архитектуры есть как преимущества, так и недостатки, которые описаны ниже.

Преимущества:

- Отсутствие дублирования кода программы-сервера программами-клиентами.
- Сделать приложения кроссплатформенным при такой архитектуре гораздо легче, ведь нужно оптимизировать лишь клиентский код.
- Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются.
- Все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Недостатки:

- Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть.
- Поддержка работы данной системы часто требует отдельного специалиста — системного администратора.
- Высокая стоимость оборудования.

Веб-сайты являются частным случаем такой архитектуры, когда общение между клиентом и сервером происходит через сеть Интернет по протоколу HTTP. Роль клиентской части выполняет браузер, который реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него.

Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP.

Ярким примером веб-приложения является система управления содержимым статей Википедии: множество её участников могут принимать

участие в создании сетевой энциклопедии, используя для этого браузеры своих операционных систем (будь то Microsoft Windows, GNU/Linux или любая другая операционная система) и не загружая дополнительных исполняемых модулей для работы с базой данных статей.

В настоящее время набирает популярность новый подход к разработке вебприложений, называемый Ajax. При использовании Ajax страницы веб-приложения не перезагружаются целиком, а лишь догружают необходимые данные с сервера, что делает их более интерактивными и производительными.

Для создания веб-приложений на стороне сервера используются разнообразные технологии и многие языки программирования, такие, как PHP, C++, ASP.NET, Perl, Ruby, Python или JavaScript (с помощью платформы NodeJS).

Клиентская часть обычно создается с помощью языка разметки HTML для определения того, что должно быть на странице, каскадных таблиц стилей для того, чтобы описать, как контент должен выглядеть, и языка сценариев JavaScript для того, чтобы создавать динамические изменения контента и стилей.

Как было описано, веб-приложения активно используют протокол HTTP для передачи данных. Говоря простыми словами, HTTP представляет из себя некий стандарт, которому должна соответствовать любая передаваемая информация. Сама структура каждого HTTP-сообщения довольно проста:

1. Стартовая строка (Starting line) — определяет тип сообщения.
2. Заголовки (Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения.
3. Тело сообщения (Message Body) — непосредственно данные сообщения.

Обязательно должно отделяться от заголовков пустой строкой.

Примером стартовой строки запроса является следующее:

```
GET /article/id35 HTTP/1.0
```

Здесь описывается метод, которым шлется запрос (всего есть несколько видов методов, но вам прежде всего нужно знать два: GET и POST). Метод

GET чаще всего используется, когда клиенту просто нужна какая-либо информация с сервера, например просто получить веб-страницу для просмотра, а метод POST нужен, если клиент не просто получает какую-либо информацию, но и передает для обработки и сохранения, например когда заполняет форму с контактными данными.

После метода через пробел идет интересующий пользователя адрес, так называемый URI, Uniform Resource Identifier, в данном случае скорее всего сервер по такому адресу вернет страницу статьи с id равным 35. Исторически здесь передавался путь до нужного файла, который хранится на сервере, но сейчас технологии позволяют создавать абстрактные URI (какой приведен в данном примере), которые привязаны не к хранящимся на сервере файлам, а к некоторому программному коду, выполняющемуся по запросу. В данном случае будет выполнен исходный код, который разыскивает в базе данных статью с id равным 35, формирует HTML-документ, который отображает содержимое этой статьи, и возвращает HTTP-ответ с этим документом.

Третьим параметром указывается версия используемого протокола.

В заголовках обычно указываются различные характеристики и параметры сообщения, например тип передаваемых данных, размер сообщения и тому подобное.

В теле же содержатся непосредственно данные сообщения, например если была запрошена веб-страница, то в теле будет находиться HTML-код этой страницы.

HTML по своей сути представляет из себя просто определенный формат данных. Для того, чтобы явно себе представить, что это такое, поставьте перед собой задачу передать только в текстовом формате документ, который содержит не только текст, но и заголовки различных уровней (главы, подглавы, разделы подглав и т. д.), таблицы, списки и подобные стандартные для более-менее сложной документации элементы. Скорее всего, вы стали бы придумывать некоторый стандарт, который бы с помощью ключевых символов говорил, что этот текст — заголовок второго уровня (подглава), а

тот текст — это список с пятью элементами. Собственно говоря, таким образом вы придумали бы аналог HTML! Чтобы понять на примере, как такую задачу решает именно данный язык разметки, внимательно изучите следующий HTML-код:

```
<h1>Глава 2. Права и свободы человека и гражданина</h1>
```

```
<h2>Статья 17</h2>
```

```
<ol>
```

```
<li>В Российской Федерации признаются и гарантируются права и свободы человека и гражданина согласно общепризнанным принципам и нормам международного права и в соответствии с настоящей Конституцией.</li>
```

```
<li>Основные права и свободы человека неотчуждаемы и принадлежат каждому от рождения.</li>
```

```
<li>Осуществление прав и свобод человека и гражданина не должно нарушать права и свободы других лиц.</li>
```

```
</ol>
```

```
<h2>Статья 18</h2>
```

```
<p>Права и свободы человека и гражданина являются непосредственно действующими. Они определяют смысл, содержание и применение законов, деятельность законодательной и исполнительной власти, местного самоуправления и обеспечиваются правосудием.</p>
```

А теперь прочтите описание основных тегов в языке HTML:

- 1) h1 — заголовок первого уровня;
- 2) h2 — заголовок второго уровня;
- 3) ol — нумерованный список, содержащий в себе несколько элементов;
- 4) li — один элемент нумерованного или маркированного списка;
- 5) p — абзац (от англ. paragraph).

Заметьте, что все вышеупомянутые теги оборачивают свои названия в угловые скобки «», а также имеют не только открывающие теги, но и закрывающие, которые отличаются тем, что после первой угловой скобки

содержат символ косой черты «/» (например, в отличие от открывающего тега `<h1>`, закрывающий будет выглядеть следующим образом: `</h1>`).

Основное описание веб-фреймворка Django

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Django является фреймворком, потому что именно с помощью него формируется каркас будущего приложения. Создатели Django уже продумали архитектуру будущего веб-приложения, которой нужно следовать, программируя с помощью фреймворка. Иными словами, Django — это довольно многофункциональный каркас, который предоставит почти все нужные возможности для всех самых распространенных в вебе задач.

Отличие фреймворков от библиотек состоит в следующем принципе: при работе с первым программист свое приложение встраивает в этот фреймворк, а при работе со второй, программист функции библиотеки встраивает в свое приложение.

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, New York Times, National Geographic, Pinterest и др.

В архитектуре всех проектов на Django должно прослеживаться четкое деление проекта на несколько как можно более независимых приложений, каждое из которых выполняет только свою узконаправленную задачу. Такой подход позволяет значительно упростить структуру проекта и вследствие этого избежать множества архитектурных ошибок. Также довольно характерной чертой Django является то, что каждое составное приложение проекта создается с помощью шаблона проектирования MVC, которое было немного изменено в связи со спецификой вебприложений.

В общих чертах для проекта, у которого три приложения (например, одно приложение для авторизации/регистрации пользователей, второе для создания/редактирования/удаления/чтения статей в блоге, третье для рекламы на сайте), схема работы представлена ниже на рисунке 2. Заметьте, что, в

идеале, приложения не должны пересекаться с собой слишком часто, а в каждом приложении реализована своя MVC-структура, где контроллер влияет и на модель, и на представление, и ещё вдобавок представление влияет тоже на модель (что видно из направления стрелок).

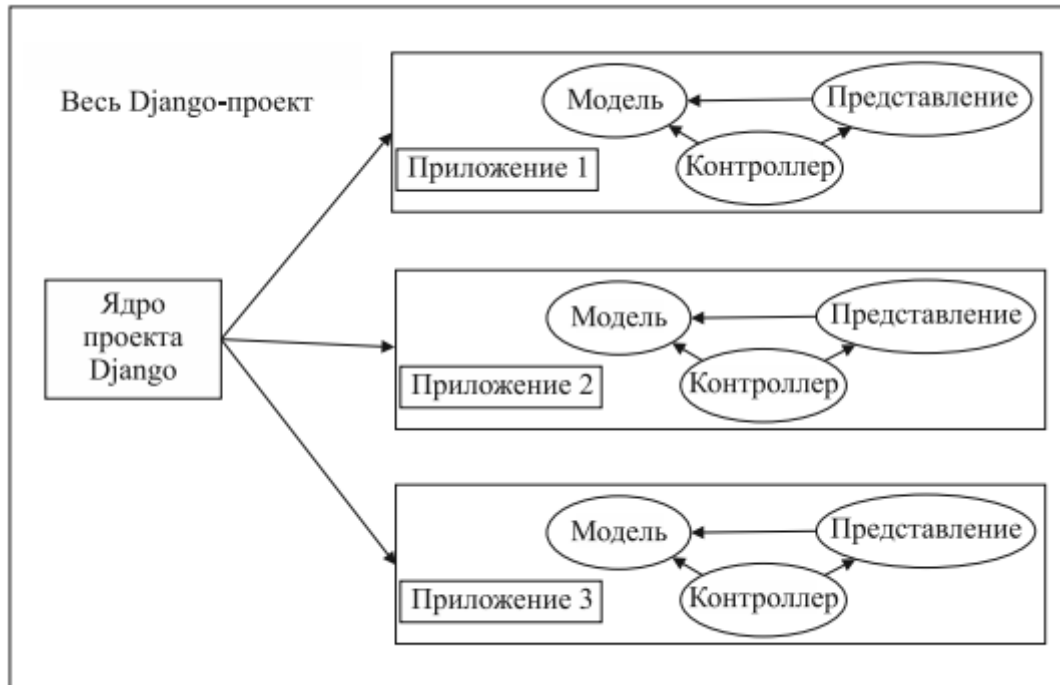


Рисунок 2 – Структура базового Django-проекта с тремя приложениями

Говоря простыми словами, MVC — это такой способ разработки программного обеспечения, при котором код определения и доступа к данным (модель) отделен от логики взаимодействия с приложением (контроллер), которая, в свою очередь, отделена от пользовательского интерфейса (представление).

Главное достоинство такого подхода состоит в том, что компоненты слабо связаны. У каждого компонента веб-приложения, созданного на базе Django, имеется единственное назначение, поэтому его можно изменять независимо от остальных компонентов. Например, разработчик может изменить URL некоторой части приложения, и это никак не скажется на ее реализации. Дизайнер может изменить HTML страницы, не трогая генерирующий ее код на языке Python. Администратор базы данных может переименовать таблицу базы данных и описать это изменение в одном-

единственном месте, а не заниматься контекстным поиском и заменой в десятках файлов.

В ходе дальнейшего развития платформы применение паттерна MVC было немного изменено, и эти изменения от стандартного подхода нужно тонко чувствовать. Суть модели остается той же самой: на уровне модели в Django реализуется только запись и получение данных из базы. Однако уровень «представления» в Django не является последним этапом в отображении данных — представления в платформе Django по своей сути ближе к «контроллерам» в архитектуре MVC. Они являются функциями на языке Python, которые связывают между собой уровень модели и уровень отображения (состоящий из разметки HTML и языка шаблонов платформы Django).

В итоге полученная модель получила название MTV — что значит Model-Template-View.

Модели. Основой любого приложения, будь то веб-приложение или любое другое, является информация, которую это приложение собирает, модифицирует и отображает.

С точки зрения многоуровневой архитектуры приложения модель является самым нижним уровнем, или фундаментом. Представления и шаблоны могут добавляться и удаляться, изменять порядок ввода/вывода данных и их представления, но модель остается практически неизменной. С точки зрения проектирования многоуровневого веб-приложения модель является, пожалуй, самой простой для понимания и самой сложной в реализации. Моделирование задач реального мира в объектно-ориентированной системе часто является относительно простой задачей, но с точки зрения веб-сайтов, работающих с высокой нагрузкой, самая реалистичная модель не всегда является самой эффективной.

Модель включает широкий диапазон потенциальных ловушек, одна из которых связана с изменением программного кода модели уже после развертывания приложения. Несмотря на то, что «изменяется всего лишь

программный код модели», тем не менее в действительности изменяется структура базы данных, а это часто отражается на данных, уже хранящихся в базе. В последующих главах мы рассмотрим множество подобных проблем, когда будем исследовать архитектуру некоторых примеров приложений.

Представления. Представления в значительной степени (иногда полностью) формируют логику приложений на платформе Django. Их определение выглядит обманчиво просто: функции на языке Python, связанные с одним или более адресами URL и возвращающие объекты ответов HTTP. Все, что будет происходить между этими двумя точками применения механизмов платформы Django, реализующих работу с протоколом HTTP, целиком и полностью зависит от вас. На практике на этом этапе обычно решается несколько похожих задач, таких как отображение объекта или списка объектов, полученных от модели, или добавление новых объектов, а также проверка аутентификации пользователя приложения и либо предоставление, либо отклонение попытки доступа к данным.

Платформа Django предоставляет множество вспомогательных функций для решения подобных задач, но вы можете реализовать всю логику работы самостоятельно, чтобы иметь полный контроль над процессом, широко использовать вспомогательные функции для быстрого создания прототипа и разработки самого приложения или комбинировать эти два подхода.

Шаблоны. Вы должны были заметить, что мы только что заявили, что представление отвечает за отображение объектов, полученных из модели. Это верно не на 100 процентов. Если подразумевать, что методы просто возвращают ответ HTTP, это достаточно верно — можно было бы реализовать на языке Python вывод строки и возвращать ее, и это было бы ничуть не хуже. Однако в подавляющем большинстве случаев такая реализация крайне неэффективна и, как уже упоминалось ранее, очень важно придерживаться деления на уровни.

Вместо этого для отображения результатов в разметку HTML, которая часто является конечным результатом работы веб-приложения, большинство разработчиков приложений на платформе Django используют язык шаблонов. Шаблоны, по сути, являются текстовыми документами в формате HTML, со специальным форматированием там, куда выводятся значения, получаемые динамически, поддерживающие возможность использовать простые логические конструкции, такие как циклы и другие. Когда от представления требуется вернуть документ HTML, оно обычно указывает шаблон, передает в него информацию для отображения и использует отображенный шаблон в своем ответе.

Хотя HTML является наиболее типичным форматом, в действительности шаблоны не обязательно должны содержать какую либо разметку HTML — шаблоны могут использоваться для воспроизведения любого текстового формата, такого как CSV (comma separated values — значения, разделенные запятыми), или даже текста сообщения электронной почты. Самое важное состоит в том, что шаблоны позволяют отделить отображение данных от программного кода представления, которое определяет, какие данные следует представить.

К настоящему моменту мы рассмотрели некоторые из наиболее крупных архитектурных компонентов, составляющих саму систему Django, а также вспомогательные компоненты, не входящие в ее границы.

Давайте теперь сложим их вместе, чтобы получить общее представление. На рисунке 3 можно видеть, что ближе всего к пользователю располагается протокол HTTP. С помощью адресов URL пользователи могут отправлять запросы веб-приложениям на платформе Django и принимать ответы посредством своих веб-клиентов, которые могут также выполнять программный код JavaScript и использовать технологию Ajax для взаимодействия с сервером.

На другом конце спектра (внизу рисунка) можно видеть базу данных — хранилище информации, которое управляется с помощью моделей и

механизма ORM платформы Django, взаимодействующих с базой данных посредством DB API языка Python и клиентских библиотек базы данных. Эти библиотеки используются в качестве промежуточного звена, они обычно написаны на языке C/C++ и предоставляют интерфейс для языка Python.

Наконец, в середине располагается основа приложения на платформе Django. Парадигма MVC на языке Django превращается в парадигму MTV (Model Template View — модель, шаблон, представление). Представления играют роль контроллера между моделью данных, создающей, изменяющей и удаляющей данные в базе посредством механизма ORM, и окончательным представлением данных пользователю с помощью шаблонов.

Соединив все вместе, получаем следующую картину: поступающие запросы HTTP передаются веб-сервером платформе Django, которая принимает их на промежуточном уровне обработки запросов. После этого, исходя из шаблонов адресов URL запросы передаются соответствующему представлению, которое выполняет основную часть работы, привлекая к работе при этом модель и/или шаблоны, необходимые для создания ответа. Затем ответ проходит один или более промежуточных уровней, где выполняется окончательная обработка перед передачей ответа HTTP обратно веб-серверу, который в свою очередь отправляет ответ пользователю.

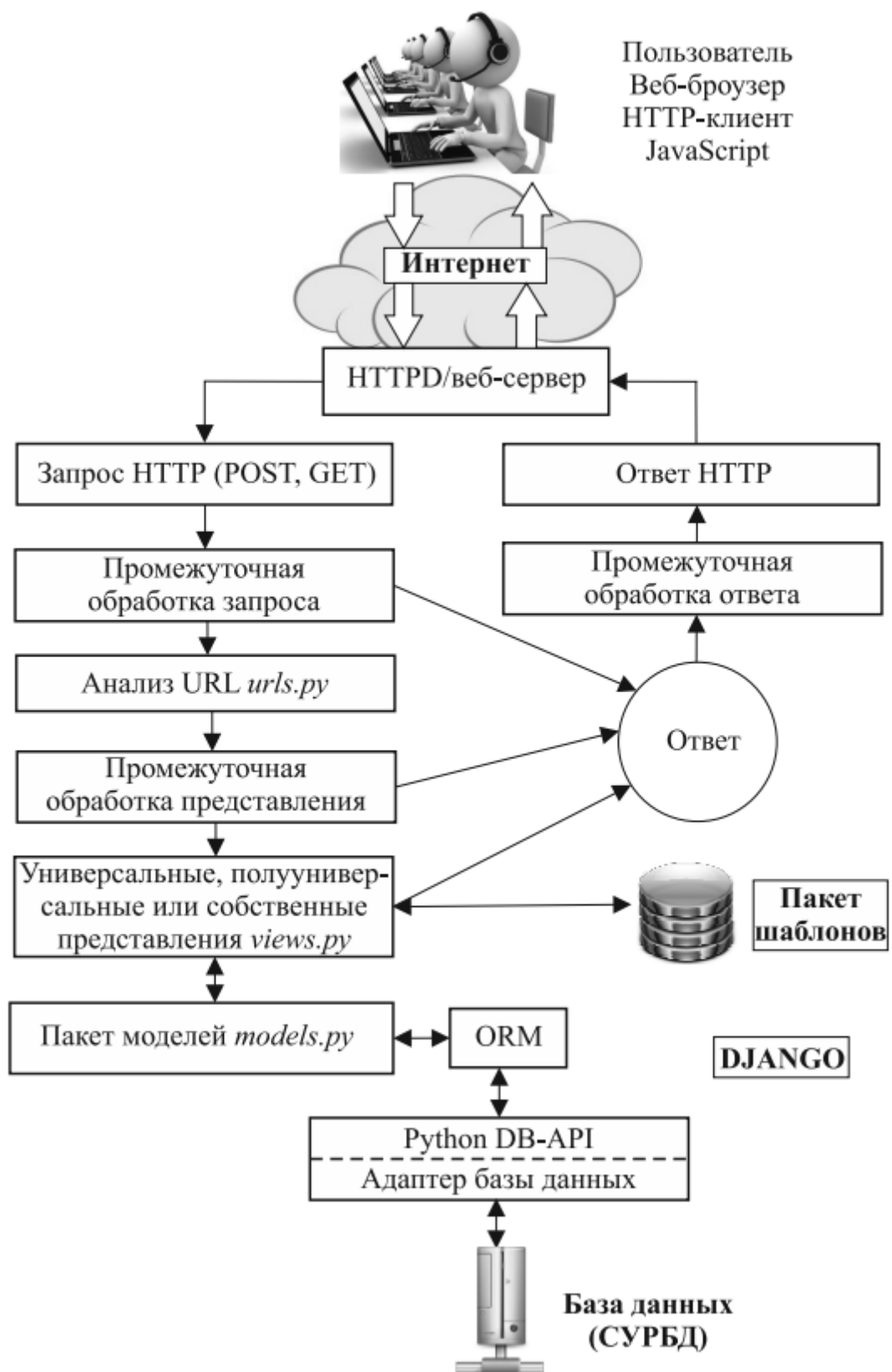


Рисунок 3 – Общая схема работы приложения на платформе Django

Выполнение

Сперва начнём с установки Python. Его можно скачать с сайта (<https://www.python.org/downloads/>).

Перед установкой проверим, есть ли на ПК уже установленный Python. Для этого необходимо ввести команду: **python** или **python --version**. Что продемонстрировано на рисунке 4.

```
PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений> python --version
Python 3.9.12
PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений> python
Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

Рисунок 4 – Проверка версии Python

После команд можно увидеть, что Python уже установлен, поэтому дополнительных действий делать не надо.

Теперь создадим директорию по адресу "C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab". Далее перейдём в эту директорию и создадим файл «mygroup.py», в котором будут находиться данные о студентах в формате списка.

Написал функцию фильтрации студентов, по средней оценке, так, чтобы на экран выводился список студентов, средний балл которых выше заданного. Средний балл, по которому будет проводиться фильтрация, вводится пользователем с клавиатуры. Листинг файла «mygroup.py» представлен ниже.

Листинг 1. Содержимое файла mygroup.py

```
mygroup.py > ...
1  # Создаем список словарей с информацией о студентах и их оценках
2  groupmates = [
3      {
4          "name": "Александр",
5          "surname": "Иванов",
6          "exams": ["Информатика", "ЭЭИС", "web"],
7          "marks": [4, 3, 5]
8      },
9      {
10         "name": "Иван",
11         "surname": "Петров",
12         "exams": ["История", "АиГ", "КТП"],
13         "marks": [4, 4, 4]
14     },
15     {
16         "name": "Вадим",
17         "surname": "Смирнов",
18         "exams": ["Философия", "ИС", "КТП"],
19         "marks": [3, 3, 3]
20     }
21 ]
```

```

21 {
22     "name": "Михаил",
23     "surname": "Кожанов",
24     "exams": ["РКПО", "СП", "ПКСП"],
25     "marks": [5, 5, 5]
26 },
27 {
28     "name": "Николай",
29     "surname": "Климов",
30     "exams": ["ОС", "Английский", "ПКСП"],
31     "marks": [5, 5, 5]
32 }
33 ]
34
35 # Определяем функцию для вывода информации о студентах
36 def print_students(students):
37     # Выводим заголовки таблицы
38     print(u"Имя".ljust(15), u"Фамилия".ljust(10), u"Экзамены".ljust(30), u"Оценки".ljust(20))
39     # Проходим по списку студентов и выводим их данные
40     for student in students:
41         print(student["name"].ljust(15), student["surname"].ljust(10), str(student["exams"]).ljust(30), str(student["marks"]).ljust(20))
42
43 # Вызываем функцию для вывода информации о студентах
44 print_students(groupmates)
45
46 # Получаем среднюю оценку от пользователя
47 mid = input("Средняя оценка: ")
48
49 # Определяем функцию для вывода информации о студентах с оценкой выше средней
50 def print_students_1(students):
51     # Выводим заголовки таблицы
52     print(u"Имя".ljust(15), u"Фамилия".ljust(10), u"Экзамены".ljust(30), u"Оценки".ljust(20))
53     # Проходим по списку студентов и выводим данные только для тех, у кого средняя оценка выше заданной
54     for student in students:
55         if sum(student["marks"]) / len(student["marks"]) > float(mid):
56             print(student["name"].ljust(15), student["surname"].ljust(10), str(student["exams"]).ljust(30), str(student["marks"]).ljust(20))
57
58 # Вызываем функцию для вывода информации о студентах с оценкой выше средней
59 print_students_1(groupmates)

```

Теперь запустив файл «mygroup.py» высвечивается список студентов. Далее требуется ввести среднюю оценку, введя которую высветятся все студенты, у которых средняя оценка выше заданной. Результат работы программы продемонстрирован на рисунок 5.

```

PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab> python mygroup.py
Имя          Фамилия      Экзамены      Оценки
Александр    Иванов       ['Информатика', 'ЭЭИС', 'Web'] [4, 3, 5]
Иван         Петров       ['История', 'АиГ', 'КТП'] [4, 4, 4]
Вадим        Смирнов      ['Философия', 'ИС', 'КТП'] [3, 3, 3]
Михаил       Кожанов      ['РКПО', 'СП', 'ПКСП'] [5, 5, 5]
Николай      Климов       ['ОС', 'Английский', 'ПКСП'] [5, 5, 5]
Средняя оценка: 4.1
Имя          Фамилия      Экзамены      Оценки
Михаил       Кожанов      ['РКПО', 'СП', 'ПКСП'] [5, 5, 5]
Николай      Климов       ['ОС', 'Английский', 'ПКСП'] [5, 5, 5]

```

Рисунок 5 – Результат работы программы

Теперь перейдём к установке Django. Установка фреймворка Django производится через командную строку командой: **pip install Django==4.2.5**.

Процесс установки показан на рисунке 6 на странице 17.


```

PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab> pip install Django==4.2.5
Collecting Django==4.2.5
  Obtaining dependency information for Django==4.2.5 from https://files.pythonhosted.org/packages/bf/8b/c38f2354b6093d9ba318a14b43a830fd776edd60c2625c7c5f4d23c243/Django-4.2.5-py3-none-any.whl.metadata
  Downloading Django-4.2.5-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.6.0 (from Django==4.2.5)
  Obtaining dependency information for asgiref<4,>=3.6.0 from https://files.pythonhosted.org/packages/9b/80/b9051a4a07ad231558fcd8ffcd89232711b4e618c15cb7a392a17384bbeef/asgiref-3.7.2-py3-none-any.whl.metadata
  Downloading asgiref-3.7.2-py3-none-any.whl.metadata (9.2 kB)
Collecting sqlparse>=0.3.1 (from Django==4.2.5)
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
  41.2/41.2 kB 2.1 MB/s eta 0:00:00
Collecting tzdata (from Django==4.2.5)
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
  341.8/341.8 kB 5.3 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in c:\users\lushi\appdata\local\programs\python\python39\lib\site-packages (from asgiref<4,>=3.6.0->Django==4.2.5) (4.7.1)
Downloading Django-4.2.5-py3-none-any.whl (8.0 MB)
  8.0/8.0 MB 10.6 MB/s eta 0:00:00
Downloading asgiref-3.7.2-py3-none-any.whl (24 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.2.5 asgiref-3.7.2 sqlparse-0.4.4 tzdata-2023.3

```

Рисунок 6 – Установка Django

После установки фреймворка создадим проект. Для этого с помощью командной строки перейдём в директорию, в которой находятся лабораторные работы, и выполним команду: **django-admin startproject admin_learning**.

Процесс создания показан на рисунке 7.

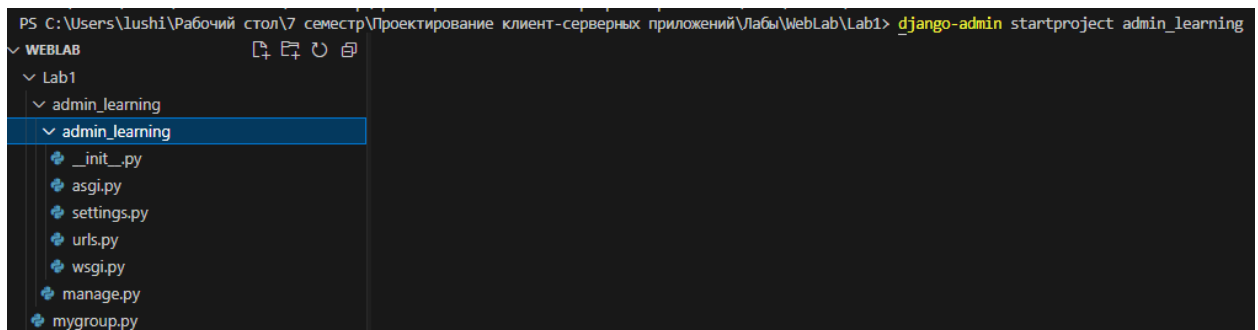


Рисунок 7 – Создание проекта Django

Для того, чтобы проверить, правильно ли был установлен фреймворк Django, необходимо запустить локальный web-сервер разработки. Для запуска сервера перейдём в директорию `admin_learning`, в которой находится файл `manage.py` и выполним команду, которая запускает локальный сервер на порту 8000. Результат выполнения команд показан на рисунке 8.

```

PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab1\admin_learning> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
September 11, 2023 - 19:21:04
Django version 4.2.5, using settings 'admin_learning.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

```

Рисунок 8 – Запуск локального web-сервера

Затем перейдём по адресу (<http://127.0.0.1:8000/>) для проверки работоспособности сервера. На нём выведено сообщение: **The install worked successfully! Congratulations!**, так как сервер был удачно запущен. На рисунке 9 на странице 18 продемонстрирована страница web-сервера.



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

Рисунок 9 – Страница web-сервера

Для того, чтобы попасть в административный интерфейс Django, необходимо обладать правами суперпользователя. Для этого в свою очередь необходимо создать нужные таблицы в базе данных, в которой будут храниться все данные о пользователях.

В файле `settings.py` изменим параметр базы данных. У переменной `DATABASES` есть внутренний словарь с ключами `ENGINE` и `NAME`. Для параметра `NAME` установим значение, продемонстрированное на рисунке 10.

```
74 # Database
75 # https://docs.djangoproject.com/en/4.2/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.sqlite3',
80         'NAME': os.path.join(BASE_DIR, 'db_admin_learning')
81     }
82 }
```

Рисунок 10 – Параметры переменной DATABASES

Чтобы создать таблицы базы данных, перейдём в директорию `admin_learning` и через командную строку выполним команду: **`python manage.py migrate`**. На рисунке 11 на странице 19 продемонстрирован результат работы команды.

```

PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab1\admin_learning> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

```

Рисунок 11 – Создание таблицы базы данных

Затем создадим суперпользователя при помощи команды: **python manage.py createsuperuser**. На рисунке 12 продемонстрирован результат работы команды.

```

PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab1\admin_learning> python manage.py createsuperuser
Username (leave blank to use 'lushi'): admin
Email address: admin@test.ru
Password:
Password (again):
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

```

Рисунок 12 – Создание суперпользователя

Теперь перейдя по адресу (<http://127.0.0.1:8000/admin/>) можно войти в личный кабинет. На рисунке 13 продемонстрировано окно авторизации.

Рисунок 13 – Окно авторизации

Теперь через интерфейс административного приложения создадим нового пользователя без прав суперпользователя, нажав на кнопку

«Добавить», напротив «Пользователи». Процесс создания показан на рисунках 14-16.

The screenshot shows the Django Admin interface for adding a new user. The breadcrumb trail is: Начало > Пользователи и группы > Пользователи > Добавить пользователя. The left sidebar has a search bar and a menu with 'Пользователи и группы' (selected), 'Группы' (+ Добавить), and 'Пользователи' (+ Добавить). The main content area is titled 'Добавить пользователь' and contains a form with the following fields: 'Имя пользователя:' with the value 'guest' and a note 'Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/-/_.'; 'Пароль:' with a masked input and three notes: 'Пароль не должен быть слишком похож на другую вашу личную информацию.', 'Ваш пароль должен содержать как минимум 8 символов.', and 'Пароль не должен быть слишком простым и распространенным.'; 'Подтверждение пароля:' with a masked input and a note 'Для подтверждения введите, пожалуйста, пароль ещё раз.' At the bottom are three buttons: 'СОХРАНИТЬ', 'Сохранить и добавить другой объект', and 'Сохранить и продолжить редактирование'.

Рисунок 14 – Создание пользователя через web-интерфейс

The screenshot shows the 'Права доступа' (Permissions) section for a user. It contains three checkboxes: 'Активный' (checked), 'Статус персонала' (unchecked), and 'Статус суперпользователя' (unchecked). Below each checkbox is a descriptive text: 'Отметьте, если пользователь должен считаться активным. Уберите эту отметку вместо удаления учётной записи.' for 'Активный'; 'Отметьте, если пользователь может входить в административную часть сайта.' for 'Статус персонала'; and 'Указывает, что пользователь имеет все права без явного их назначения.' for 'Статус суперпользователя'.

Рисунок 15 – Создание пользователя через web-интерфейс

<input type="checkbox"/>	ИМЯ ПОЛЬЗОВАТЕЛЯ	▲	АДРЕС ЭЛЕКТРОННОЙ ПОЧТЫ	ИМЯ	ФАМИЛИЯ	СТАТУС ПЕРСОНАЛА
<input type="checkbox"/>	admin		admin@test.ru			✓
<input type="checkbox"/>	guest					✗

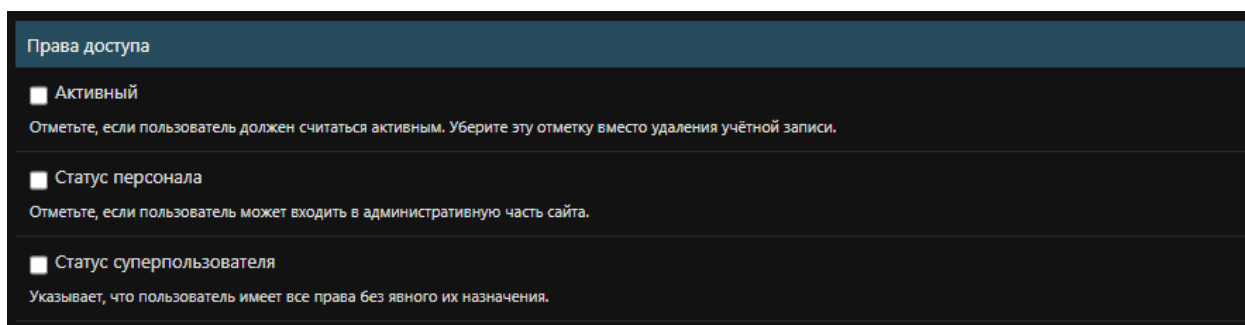
Рисунок 16 – Создание пользователя через web-интерфейс

Теперь для создания нового пользователя с правами суперпользователя нужно поставить соответствующую галочку как показано на рисунке 17.

The screenshot shows the 'Права доступа' (Permissions) section for a user, similar to Figure 15, but with the 'Статус суперпользователя' checkbox checked. The other checkboxes and their descriptions remain the same.

Рисунок 17 – Создание суперпользователя через web-интерфейс

Через интерфейс административного приложения «забаним» одного из пользователей (сделаем пользователя «неактивным»). Это продемонстрировано на рисунке 18.



The screenshot shows a dark-themed Django admin interface. At the top, there is a header bar with the text "Права доступа" (Access rights). Below this, there are three sections, each with a checkbox and a label:

- ☐ Активный
Отметьте, если пользователь должен считаться активным. Уберите эту отметку вместо удаления учётной записи.
- ☐ Статус персонала
Отметьте, если пользователь может входить в административную часть сайта.
- ☐ Статус суперпользователя
Указывает, что пользователь имеет все права без явного их назначения.

Рисунок 18 – Изменение статуса пользователя на неактивный

Вывод: В данной лабораторной работе я научился работать с Python через консоль, а также научился создавать сервер через Django и управлять им через интерфейс административного приложения.