МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

# РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СКРИПТОВЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

КУРСОВАЯ РАБОТА

Студентки 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Мухиной Анастасии Алексеевны

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2020

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Целью настоящей работы является разработка веб-приложения с использованием скриптовых языков программирования, включающая

- написание серверной части,
- создание пользовательского интерфейса,
- использование базы данных,
- реализация взаимодействия пользователей и приложения.

# 1 Теоретическая часть разработки приложения

## 1.1 Скриптовые языки программирования

Скриптовый язык программирования — это набор лексических, семантических и синтаксических правил, разработанный для записи «сценариев», последовательностей операций, которые пользователь может выполнять на компьютере. Корректно также синонимичное название «язык сценариев». [1]

Сценарий (скрипт) — это отдельные последовательности действий, созданные для автоматического выполнения задачи, которую без сценария пользователь делал бы вручную, используя интерфейс программы, что влечет за собой соответствующие затраты времени и возможности появления ошибок. Для написания скриптов используются скриптовые языки программирования. Современные скриптовые языки разбирают скрипт во время запуска и преобразуют текст в промежуточный байт-код. Однажды скомпилированный, скрипт уже трудно отличить от программы.

Скриптовый язык удобен в следующих случаях: Если нужно обеспечить программируемость без риска дестабилизировать систему. Так как, в отличие от плагинов, скрипты интерпретируются, а не компилируются, неправильно написанный скрипт выведет диагностическое сообщение, а не приведёт к системному краху; Если важен выразительный код. Во-первых, чем сложнее система, тем больше кода приходится писать «потому, что это нужно». Во-вторых, в скриптовом языке может быть совсем другая концепция программирования, чем в основной программе — например, игра может быть монолитным однопоточным приложением, в то время как управляющие персонажами скрипты выполняются параллельно. В-третьих, скриптовый язык имеет собственный проблемно-ориентированный набор команд, и одна строка скрипта может делать то же, что несколько десятков строк на традиционном языке. Как следствие, на скриптовом языке может писать программист очень низкой квалификации — например, геймдизайнер своими руками, не полагаясь на программистов, может корректировать правила игры; Если требуется кроссплатформенность. [2]

Виды скриптов:

— SEO-скрипты (шаблоны) для продвижения сайтов. Обычно под их управ-

лением работают специализированные программы автоматизации этого процесса. Наиболее известные – ZennoPoster, Human Emulator; [3]

— системы для сбора статистики посещений (счетчики посещаемости). Эти скрипты чаще всего создаются с применением JavaScript;
— сценарии для обращения к базам данных. Здесь лидирует язык PHP;
— скрипты для работы гостевых книг и создания комментариев к записям. Чаще всего применяется комбинация PHP и JavaScript;
— скрипты для динамического отображения сайтов. В этом случае скриптовый язык определяется языком написания CMS;
— скрипты для изменения части страницы сайта без ее перезагрузки. При реализации используются технологии Ajax. В этом случае на первый план выходят асинхронный JavaScript и XML. Веб-приложения производят обмен данных с сервером в «фоне», изменения на страницах сайта происходят без их полной перезагрузки. Пользователи обычно не замечают таких изменений, и им не нужно понимать, что такое скриптовый язык программирования, чтобы отлично взаимодействовать с сайтом. [4]

Общие характеристики скриптовых языков программирования

— Динамическая типизация — переменная связывается с типом в момент присваивания значения, а не объявления переменной.
— Интерпретация — исходный код программы не преобразовывается в машинный код для непосредственного выполнения центральным процессором, а выполняется с помощью специальной программы-интерпретатора.
— Нет нужды в компиляции. Интерпретатор может уметь преобразовывать текст в промежуточный код, но этого не требуется делать в отдельной стадии. Такой код, если это предусмотрено конструкцией языка, будет сгенерирован при первом запуске скрипта.
— Возможности скриптового языка обычно представлены в виде более высокого уровня абстрагирования от системы, чем в компилируемом языке программирования.
— Команды в сценарии — это в большинстве своем команды, доступные оператору в командной строке операционной системы. Скрипты могут также содержать свои собственные команды.

— Безопасность. Скриптовый язык обеспечивает программируемость без риска дестабилизации системы. Скрипты не компилируются, а интерпретируются. Поэтому неправильно написанная программа выведет диагностическое сообщение, не вызывая падение системы;
— Наглядность. Язык сценариев используется, если необходим выразительный код. Концепция программирования в скриптовом языке может кардинально отличаться от основной программы;
— Простота. Код имеет собственный набор программ, поэтому одна строка может выполнять те же операции, что и десятки строк на обычном языке. Поэтому для написания кодов не требуется программист высокой квалификации;
— Кроссбраузерность. Скриптовые языки ориентированы на кроссбраузерность. Например, JavaScript может исполняться браузерами практически под всеми современными операционными системами. [5]

Основные плюсы и минусы

Плюсы:

— Их применение дает возможность вносить программные изменения без опасения разрушить всю систему. Если скрипт написан с ошибкой, то при его выполнении они будут выданы в результате. При этом сайт останется работоспособным.
— Использование скриптов дает возможность получать проблемно ориентированный набор команд. В этом случае одна строка сценария позволяет выполнять такой же объем действий, как программа из многих десятков строк на компилируемом языке.
— С использованием скриптов успешно реализуется кроссплатформенность выполнения задач.
— Увеличивают производительность разработчика
— Просты в изучении
— Позволяют быстро выполнять доработку кода

Минусы:

— Заметно большее время исполнения. Практически во всех случаях интерпретируемые сценарии требуют для выполнения гораздо больше времени и компьютерных ресурсов.
— До сих пор для таких языков не создана качественная среда разработки уровня IDE. В продвижение и рекламу этих языков вкладываются недостаточные средства. Как это ни парадоксально, относительная доступность и условная бесплатность сценарных языков приводят к тому, что у разработчиков просто не хватает средств на маркетинг и рекламу. Поэтому для многих крупных денежных проектов выбираются Java или C. [6]

### 1.2 Понятие веб-приложения

Веб-приложение — это решение, в основе которого лежит взаимодействие браузера и веб-сервера. Такие приложения являются кроссплатформенными сервисами, доступными с любого современного устройства, и не привязаны к архитектуре сети: вы можете получить доступ к ним с локального компьютера или со смартфона на другом конце света по удобному для вас протоколу, например, наиболее быстрому или зашифрованному.

Чаще всего веб-приложения состоят как минимум из трёх основных компонентов:

Клиентская часть веб-приложения — это графический интерфейс. Это то, что пользователь видит на странице. В первую очередь, она отвечает за интерфейс и непосредственное взаимодействие с пользователем. Графический интерфейс отображается в браузере.

Серверная часть веб-приложения — это программа или скрипт на сервере, обрабатывающая запросы пользователя (точнее, запросы браузера). Здесь производятся все сложные вычисления, хранятся объемные данные, координируется работа в целом. При каждом переходе пользователя по ссылке браузер отправляет запрос к серверу. Сервер обрабатывает этот запрос, вызывая некоторый скрипт, который формирует веб-страничку, описанную языком HTML, и отсылает клиенту по сети. Браузер тут же отображает полученный результат в виде очередной веб-страницы. [7]

База данных — программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. В случае форума или блога, хранимые в БД данные — это посты, комментарии, новости, и так далее. База

данных располагается на сервере. Серверная часть веб-приложения обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

Такая архитектура позволяет разделить зоны ответственности между двумя подсистемами и сделать их более независимыми. Схема взаимодействия фиксируется в API, которое может быть использовано многократно. Нет необходимости переписывать серверную часть под различные реализации клиента и, наоборот, при изменении внутренней логики сервера — все клиенты могут продолжать его использовать, пока соблюдается установленный API. [8]

Любое веб-приложение представляет собой набор статических и динамических веб-страниц. Статическая веб-страница — это страница, которая всегда отображается перед пользователем в неизменном виде. Веб-сервер отправляет страницу по запросу веб-браузера без каких-либо изменений. В противоположность этому, сервер вносит изменения в динамическую веб-страницу перед отправкой ее браузеру. По причине того что страница меняется, она называется динамической. [9]

Веб-сервер — это программное обеспечение, которое предоставляет веб-страницы в ответ на запросы веб-браузеров. Обычно запрос страницы создается при щелчке ссылки на веб-странице, выборе закладки в браузере либо вводе URL-адреса в адресной строке браузера. С точки зрения "железа «веб-сервер» — это компьютер, который хранит файлы сайта и доставляет их на устройство конечного пользователя. Он подключен к сети Интернет и может быть доступен через доменное имя. С точки зрения ПО, веб-сервер включает в себя несколько компонентов, которые контролируют доступ веб-пользователей к размещенным на сервере файлам, как минимум — это HTTP-сервер. HTTP-сервер — это часть ПО, которая понимает URL’ы и HTTP.

Окончательное содержимое статической веб-страницы определяется разработчиком и остается неизменным в процессе запроса страницы. Весь HTML-код создается разработчиком до того момента, когда страница будет размещена на сервере. Поскольку HTML-код не меняется после размещения страницы на сервере, данная страница называется статической.

Обработка статической веб-страницы

Веб-браузер запрашивает статическую страницу. Веб-сервер находит страницу. Веб-сервер отправляет страницу запросившему ее браузеру. В случае веб-приложений некоторые участки кода страницы отсутствуют до момента запроса страницы посетителем. Отсутствующий код формируется с помощью некоторого механизма, и только после этого страница может быть отправлена браузеру.

Обработка динамических страниц

Когда веб-сервер получает запрос на выдачу статической веб-страницы, он отправляет страницу непосредственно браузеру. Однако, когда запрашивается динамическая страница, действия веб-сервера не столь однозначны. Сервер передает страницу специальной программе, которая и формирует окончательную страницу. Такая программа называется сервером приложений.

Сервер приложений выполняет чтение кода, находящегося на странице, формирует окончательную страницу в соответствии с прочитанным кодом, а затем удаляет его из страницы. В результате всех этих операций получается статическая страница, которая передается веб-серверу, который в свою очередь отправляет ее клиентскому браузеру. Все страницы, которые получает браузер, содержат только HTML-код. [10]

Веб-браузер запрашивает динамическую страницу. Веб-сервер находит страницу и передает ее серверу приложений. Сервер приложений просматривает страницу на наличие инструкций и выполняет ее создание. Сервер приложений возвращает подготовленную страницу на веб-сервер. Веб-сервер отправляет подготовленную страницу запросившему ее браузеру.

Процесс разработки динамических страниц состоит из написания базового HTML-кода и последующего создания серверных сценариев или тегов HTML-страницы, с помощью которых страница становится динамической. Если взглянуть на окончательный код, видно, что язык сценариев встроен в HTML-код страницы. Соответственно, такие языки сценариев называют языками, встроенными в HTML.

Доступ к базе данных

Веб-браузер запрашивает динамическую страницу. Веб-сервер находит страницу и передает ее серверу приложений. Сервер приложений просматривает страницу на наличие инструкций и выполняет ее подготовку. Сервер приложений отправляет запрос драйверу базы данных. Драйвер выполняет запрос в базе данных. После того как драйвер установит соединение, выполняется запрос к базе, в результате чего формируется набор записей. Набор записей представляет собой множество данных, полученных из одной или нескольких таблиц базы данных. Набор записей возвращается серверу приложений, который использует полученные данные для формирования страницы. Драйвер передает набор записей серверу приложений. Сервер приложений вставляет данные в страницу и передает страницу веб-серверу. Веб-сервер отправляет подготовленную страницу запросившему ее браузеру. Для использования в веб-приложении пригодна любая база данных при условии, что на сервере установлен соответствующий драйвер базы данных.

Если база данных и веб-сервер располагаются на разных компьютерах, следует обеспечить скоростное подключение между системами, поскольку от этого будет зависеть эффективность и скорость работы всего веб-приложения. Драйвер базы данных — программный модуль, с помощью которого устанавливается взаимодействие между веб-приложением и базой данных. Данные в базе данных хранятся в специфическом для каждой базы собственном формате. С помощью драйвера базы данных веб-приложение получает возможность работать с данными, хранящимися в базе. СУБД, или система баз данных, представляет собой программное обеспечение, предназначенное для создания баз данных и управления ими. [11]

Pyhon-фреймворк Django

В разработке фреймворк — это набор готовых библиотек и инструментов, которые помогают создавать веб-приложения. Для примера опишу принцип работы фреймворка Django, написанного на языке программирования Python.

Первым этапом запрос от пользователя попадает в роутер, который решает какую функцию для обработки запроса надо вызвать. Решение принимается на основе списка правил, состоящих из регулярного выражения и названия функции: если такой-то урл, то вот такая функция.

Функция, которая вызывается роутером, называется вью. Внутри может содержаться любая бизнес-логика, но чаще всего это одно из двух: либо из базы берутся данные, подготавливаются и возвращаются на фронт; либо пришел запрос с данными из какой-то формы, эти данные проверяются и сохраняются в базу.

Данные приложения хранятся в базе данных. Чаще всего используются реляционные БД. Это когда есть таблицы с заранее заданными колонками и эти таблицы связаны между собой через одну из колонок.

Данные в БД можно создавать, читать, изменять и удалять. Иногда для обозначения этих действий можно встретить аббревиатуру CRUD. Для запроса к данным в БД используется специальный язык SQL.

В Джанго для работы с БД используются модели. Они позволяют описывать таблицы и делать запросы на привычном разработчику питоне, что гораздо удобнее. За это удобство приходится платить: такие запросы медленнее и ограничены в возможностях по сравнению с использованием чистого SQL.

Полученные из БД данные подготавливаются во вью к отправке на фронт. Они могут быть подставлены в шаблон (template) и отправлены в виде HTML-файла. Но в случае одностраничного приложения это происходит всего один раз, когда генерируется HTML-страница, на который подключаются все JS-скрипты. В остальных случаях данные сериализуются и отправляются в JSON-формате. [12]

Как клиент и сервер общаются между собой

Общение клиента с сервером происходит по протоколу HTTP. Основа этого протокола — это запрос от клиента к серверу и ответ сервера клиенту.

Для запросов обычно используют методы GET, если мы хотим получить данные, и POST, если мы хотим изменить данные. Еще в запросе указывается Host (домен сайта), тело запроса (если это POST-запрос) и много дополнительной технической информации.

Современные веб-приложения используют протокол HTTPS, расширенную версию HTTP с поддержкой шифрования SSL/TLS. Использование шифрованного канала передачи данных, независимо от важности этих данных,

стало хорошим тоном в интернете.

Есть еще один запрос, который делается перед HTTP. Это DNS (domain name system) запрос. Он нужен для получения ip-адреса, к которому привязан запрашиваемый домен. Эта информация сохраняется в браузере и мы больше не тратим на это время.

Когда запрос от браузера доходит до сервера, он не сразу попадает в Джанго. Сначала его обрабатывает веб-сервер Nginx. Если запрашивается статический файл (например, картинка), то сам Nginx его отправляет в ответ клиенту. Если запрос не к статике, то Nginx должен проксировать (передать) его в Джанго.

К сожалению, он этого не умеет. Поэтому используется еще одна программа-прослойка — сервер приложений. Например для приложений на питоне, это могут быть uWSGI или Gunicorn. И вот уже они передают запрос в Джанго.

После того как Джанго обработал запрос, он возвращает ответ с HTML-страницей или данными, и код ответа. Если все хорошо, то код ответа — 200; если страница не найдена, то — 404; если произошла ошибка и сервер не смог обработать запрос, то — 500. Это самые часто встречающиеся коды. [13]

### 1.3 HTML в разработке веб-приложения

Чтобы создавать веб-сайты, необходимо знать о HTML — фундаментальной технологии, которая используется для определения структуры веб-страницы. HTML применяется для того, чтобы определить как должен отображаться ваш контент: в виде абзаца, списка, заголовка, ссылки, изображения, мультимедийного проигрывателя, формы или же в виде одного из множества других доступных элементов, а также возможного нового элемента, который вы сами создадите. HTML- это язык разметки гипертекста. Он интерпретируется браузером и отображается в виде документа в удобной для человека форме. HTML позволяет нам наделять содержимое страницы определенным смыслом, а реализуется это с помощью так называемых тэгов. [14]

Тэги – это специальные маркеры, которые определенным образом интерпретируются браузером. Суть тэгов в том, что содержимое страницы, заключенное в разные тэги, по-разному обрабатывается браузером. Каждый из HTML-тэгов, предназначенных для разметки текстовой информации, придает этой информации некоторый смысл.

CSS – это язык описания внешнего вида документа, написанного с ис-

пользованием языка разметки. Язык CSS предназначен для того, чтобы придавать необходимый внешний вид HTML-документам.

Придание внешнего вида документам HTML – это хоть и самый популярный, однако лишь частный случай применения языка CSS, т.к. с его помощью можно придавать вид и документам других типов: XHTML, SVG и XUL. Про них мы отдельно говорить не будем, т.к. это выходит за рамки рассматриваемого вопроса.

Итак, целью создания CSS было отделение описания логической структуры веб-страницы от ее внешнего вида. Как вы уже знаете, для описания структуры используется HTML, для описания же того, как эта логическая структура будет выглядеть, отвечает как раз CSS.

Раздельное описание логической структуры и представления документа позволяет более гибко управлять внешним видом документа и минимизировать объем повторяющегося кода, который бы неизбежно возникал при использовании HTML для описания внешнего вида документа.

С помощью CSS веб-разработчик может задавать для страницы и отдельных ее элементов различные гарнитуры и размеры шрифта, цвета элементов, отступы элементов друг от друга, расположение отдельных блоков на странице и т.д.

Разумеется, для того, чтобы использовать CSS для придания внешнего вида HTML-документу, нужно этот документ как-то связать со стилями, т.е. «сообщить» HTML-документу, что он будет оформлен с помощью CSS.

Для этого существуют различные способы подключения CSS к документу, которые дают браузеру знать, что к странице в целом, либо к каким-то отдельным ее элементам должно быть применено стилевое оформление.

Таблицы стилей могут располагаться как непосредственно внутри того, документа, к которым они будут применяться, так и находиться в отдельном файле, имеющем расширение .css. [15]

Важно понимать, что CSS-файл – это обычный текстовый файл. В нем пишутся специальные инструкции, описывающие внешний вид элемента и его позиционирование на странице а также комментарии (произвольные пояснения относительно написанных инструкций).

В процессе своего развития язык CSS уже прошел достаточно длинный путь, и в настоящее время существует несколько его уровней: CSS1, CSS2,

CSS2.1, CSS3. С конца 2011 года разрабатывается уже CSS4.

Смысл разных уровней в том, что происходит исправление существующих ошибок, добавление новых свойств, расширение механизма селекторов и т.д.

Иными словами, каждый следующий уровень является не чем-то обособленным, а логическим развитием и продолжением предыдущего уровня, позволяя более тонко и гибко управлять внешним видом web-страниц.

В случае с HTML-страницей есть только один этап: браузер обрабатывает HTML-код, т.е. разметку страницы в соответствии с определенными правилами, в результате чего мы и видим веб-страницу в нормальном виде.

Фундаментом веб-разработки был и остается язык HTML. Без него все остальное практически лишено смысла, ведь именно HTML-разметка преобразуется браузером в ту итоговую картину, что мы видим на экране монитора. [16]

CSS является инструментом задания внешнего вида и позиционирования различных элементов веб-страницы, что позволяет нам гибко управлять обликом нашего веб-приложения.

## 2 Практическая часть разработки приложения

Как пример использования в разработке скриптового языка Python было написано веб-приложение My Travel, которое предоставляет пользователям информацию о странах мира, позволяет оставлять на форуме отзывы о своих поездках в какую-либо страну, видеть отзывы других людей, добавлять фотографии и видео из различных стран, отмечать места, где хотелось бы побывать, и добавлять их в свои записи. [17]

### 2.1 Серверная часть приложения

Серверная часть приложения представляет собой код на языке Python, который описывает основные разделы приложения, обеспечивает соединение с сервером и стабильность работы приложения.

Чтобы создать работоспособное веб-приложения на языке Python, мне потребовалось установить фреймворк Django, позволяющий быстро запустить сайт и обеспечить его работу.

Сначала выполняем базовую настройку приложения и базы данных.

Используем базу данных SQLite, которая уже встроена в проект.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),
    }
}
```

Настраиваем URL-адрес страницы.

URL соотношения хранятся в переменной urlpatterns, т.е. в списке функций path(). Каждая path() функция ассоциирует шаблон URL или с контроллером, или с другим таким списком. Добавляем новые элементы в список паттернов и соотношения, которые перенаправляют запросы с корневого URL на URL приложения.

```
from django.contrib import admin
from django.urls import include
from django.urls import path
from django.views.generic import RedirectView
from django.conf import settings
from django.conf.urls.static import static
```

```
7
8  urlpatterns = [
9      path('admin/', admin.site.urls),
10     path('countries/', include('countries.urls')),
11     path('', RedirectView.as_view(url='/countries/', permanent=True)), ] +
       ↪  static(settings.STATIC_URL,
```

Создаем основные модели веб-приложения. В проекте используем три модели — Страна, Отзывы, Пользователи. В Отзывах мы определяем поля: поле для текста отзыва, поле для названия страны, о которой мы оставляем отзыв, поле имени пользователя, который оставляет отзыв. Также определяем методы для работы с моделью. Создаем метод для возвращения удобочитаемой строки с информацией о каждого объекта. После создаем метод, который возвращает URL-адрес для отображения отдельных записей модели на веб-сайте. В приложении показан код оставшихся моделей.

```
1  from django.db import models
2  from django.urls import reverse  # Used to generate URLs by reversing the URL
   ↪  patterns
3  import uuid  # Required for unique book instances
4
5  # Create your models here.
6
7
8  class Review(models.Model):
9      """
10     Model representing reviews by the users.
11     """
12     text_of_review = models.CharField(max_length=2000, help_text="Enter a
       ↪  review about the country")
13     countryRev = models.CharField(max_length=30, help_text="Enter the name of
       ↪  the country")
14     usernameRev = models.CharField(max_length=30, help_text="Username")
15
16     def __str__(self):
17         """
18         String for representing the Model object (in Admin site etc.)
19         """
20         return self.text_of_review
21
22     def get_absolute_url(self):
```

```
        """
        Returns the url to access a particular book instance.
        """
        return reverse('reviews', args=[str(self.id)])
```

С помощью административной панели Django Admin мы имеем возможность использовать модели для создания части сайта, добавления и удаления данных, просмотра и обновления записей. Для работы с ней модели описаны в файле admin.py, который отвечает за их привязку к сайту.

```
from django.contrib import admin
from .models import Review, Country, User

# Register your models here.

# admin.site.register(Country)
# admin.site.register(User)
# admin.site.register(Review)


# Define the admin class
class UserAdmin(admin.ModelAdmin):
    list_display = ('username', 'origin_country', 'date_of_birth')


# Register the admin class with the associated model
admin.site.register(User, UserAdmin)


# Define the admin class
class CountryAdmin(admin.ModelAdmin):
    list_display = ('title', 'information', 'attractions')
```

Для того, чтобы начать работу с компонентами веб-приложения, заходим в админ-панель. Здесь администратор может взаимодействовать со странами, пользователями или отзывами. (см. рисунок 2.1).

Со стороны сервера можно увидеть информацию о всех странах, пользователях и все отзывы, как показано на рисунках 2.2, 2.3, 2.4.

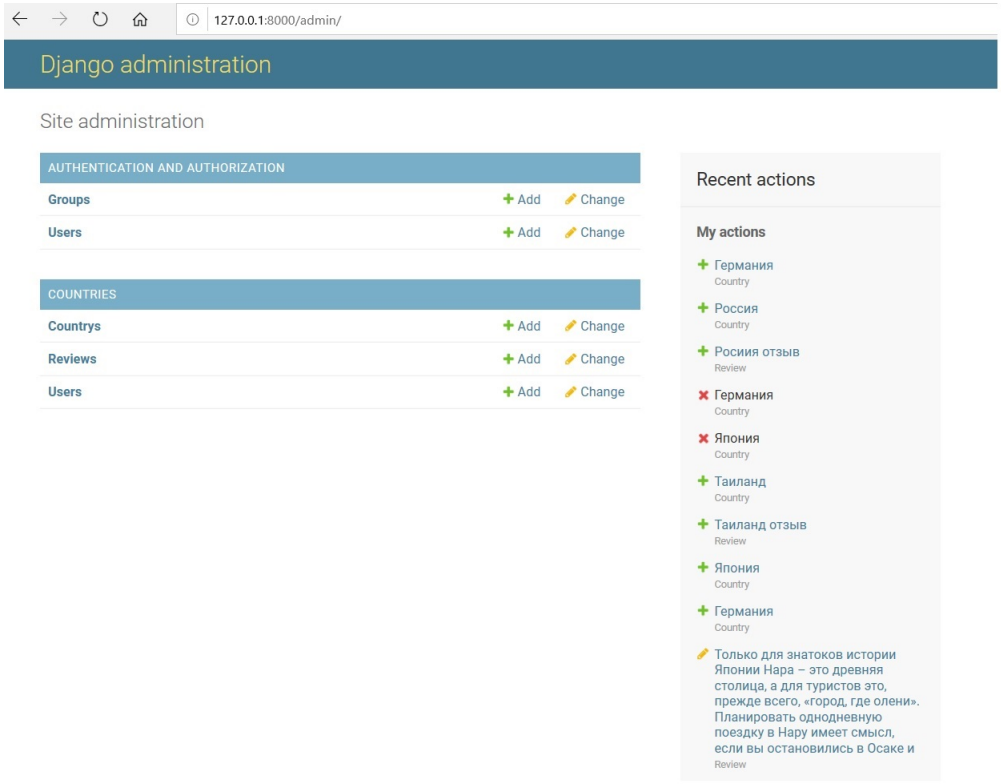


Рисунок 2.1 – Админ-панель для работы с приложением

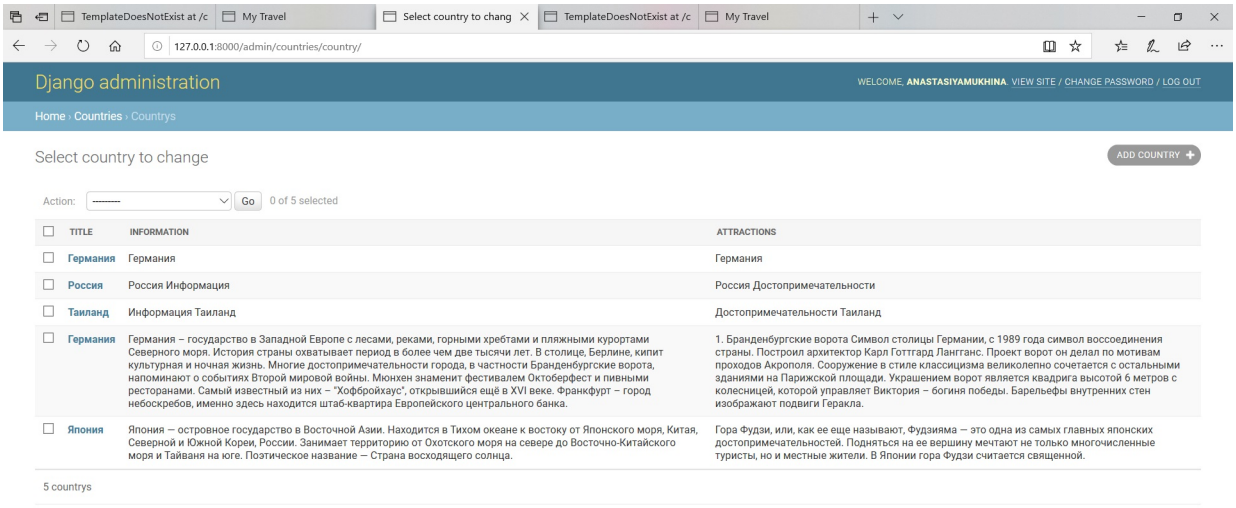


Рисунок 2.2 – Список всех стран

С серверной стороны добавлено поле сортировки отзывов по стране или по пользователю (см. рисунок 2.5).

Есть также возможность добавления и удаления записей на сервере. Добавление показано на рисунках 2.6, 2.7, 2.8.

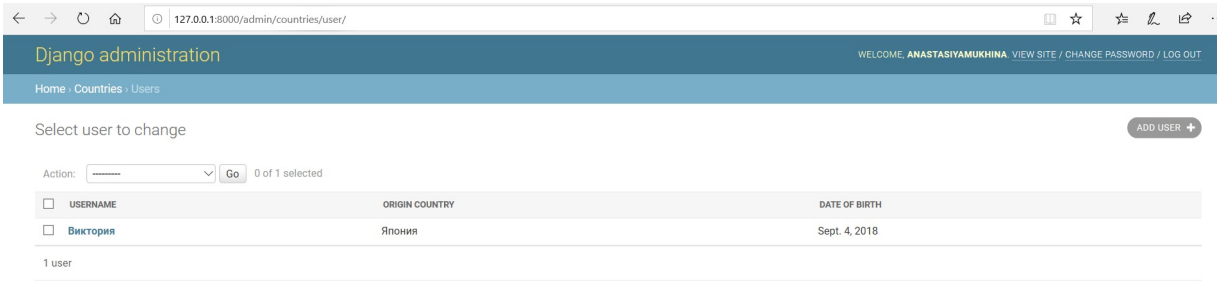


Рисунок 2.3 – Список всех пользователей

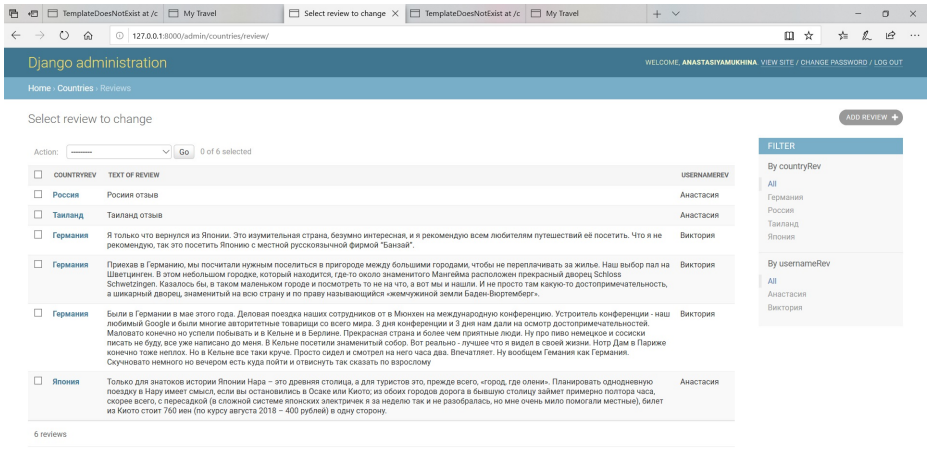


Рисунок 2.4 – Список отзывов от всех пользователей

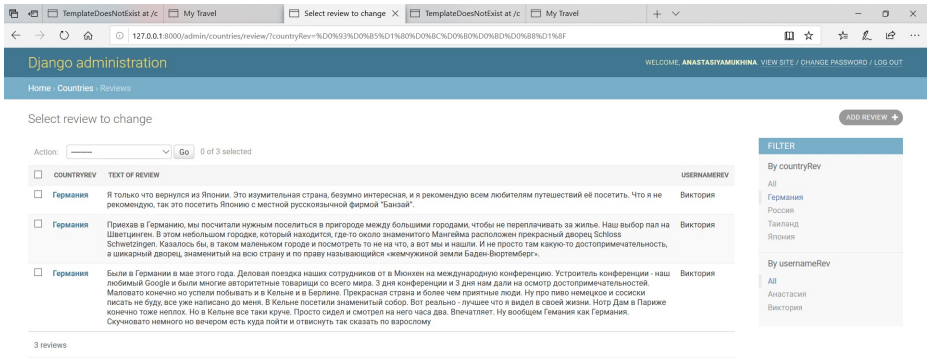


Рисунок 2.5 – Список отзывов о стране Германия

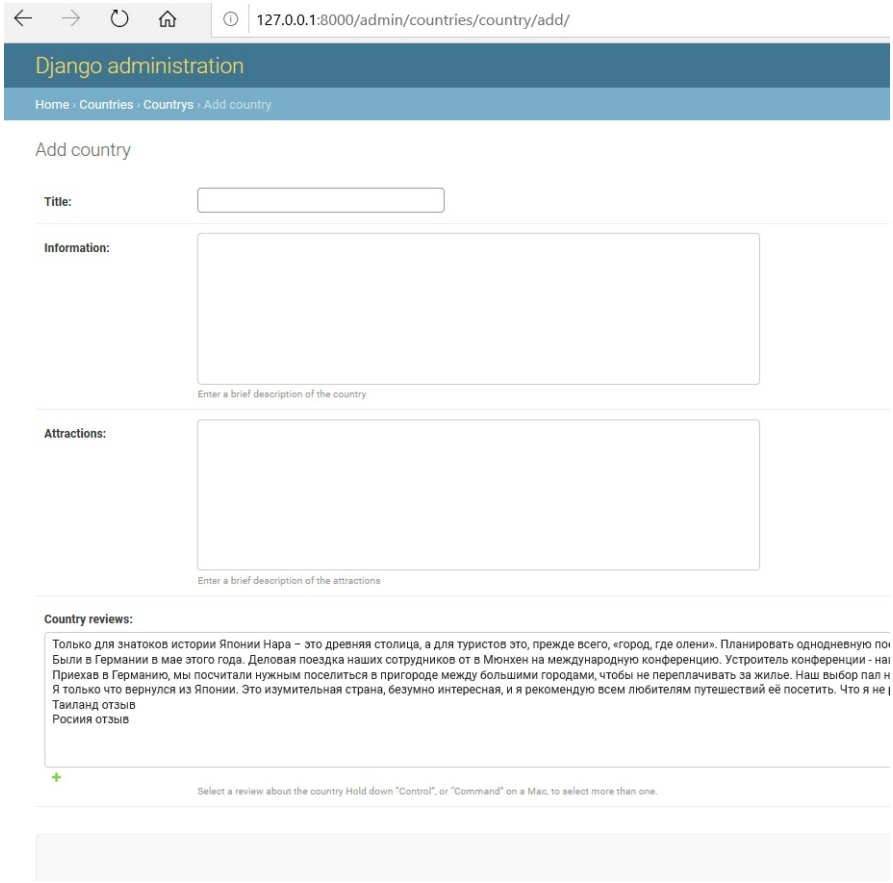


Рисунок 2.6 – Добавление страны

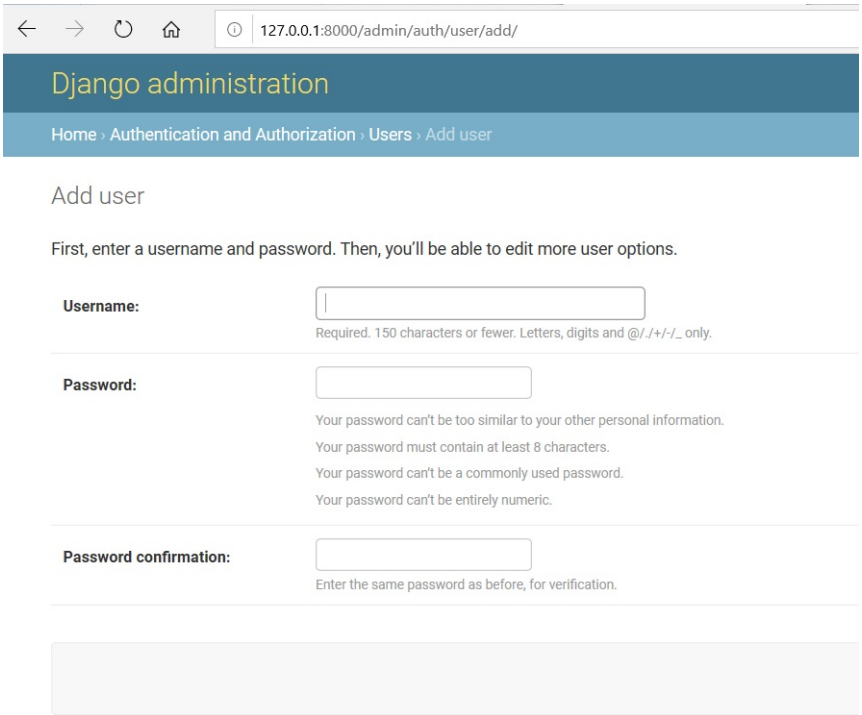


Рисунок 2.7 – Добавление пользователя

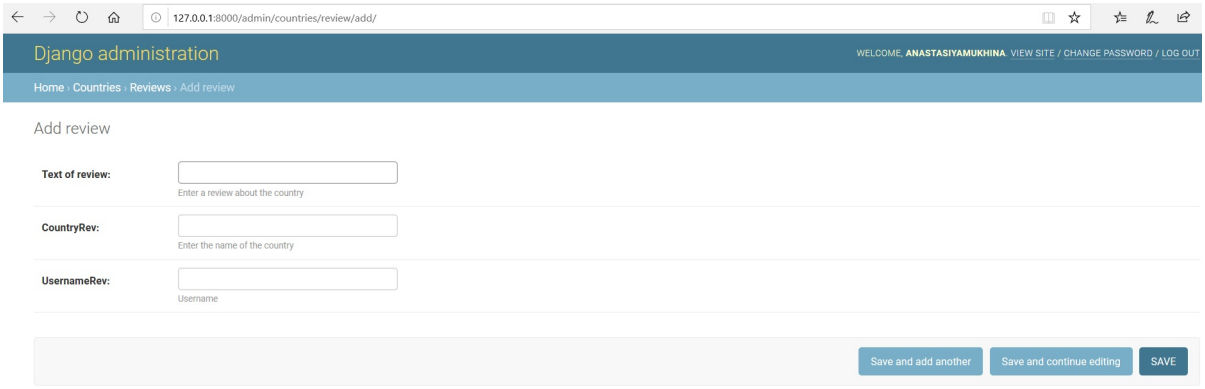


Рисунок 2.8 – Добавление отзыва

## 2.2 Взаимодействия пользователей и приложения

Пользовательский интерфейс разрабатывается на языке Python, а отображается на сайте с помощью языка разметки гипертекста HTML и CSS. Сначала опишем программную часть, которая организует взаимодействие компонентов приложения в клиентской части.

Создаем главную страницу сайта. Сначала прописываем URL-адреса всех страниц, с которыми может взаимодействовать пользователь, и прописываем связи.

```
from django.urls import path
from . import views
from django.conf.urls import url


urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^country/$', views.CountryListView.as_view(), name='country'),
    url(r'^country_details/(\d+)$', views.CountryDetailView.as_view(),
    ↪  name='country-details'),
    url(r'^reviews/$', views.ReviewsListView.as_view(), name='reviews'),
    url(r'^user/$', views.UsersListView.as_view(), name='user'),
]
```

Создаем отображение, которое будет обрабатывать HTTP-запрос от страницы и генерировать страницу HTML.

```
from django.shortcuts import render
from .models import Country, Review, User
from django.views import generic
```

```
4
5 # Create your views here.
6
7
8 def index(request):
9     num_countries=Country.objects.all().count()
10    num_reviews=Review.objects.all().count()
11    num_users=User.objects.all().count()
12
13    # Отрисовка HTML-шаблона index.html с данными внутри
14    # переменной контекста context
15    return render(
16        request,
17        'index.html',
18        context={'num_countries': num_countries, 'num_reviews': num_reviews,
          ↪  'num_users': num_users},
19    )
```

Здесь же описаны классы, которые получают все записи конкретной модели, которые нужно будет вывести в виде списка.

```
1 class CountryListView(generic.ListView):
2     model = Country
3     paginate_by = 10
4
5
6 class CountryDetailView(generic.DetailView):
7     model = Country
8
9
10 class ReviewsListView(generic.ListView):
11     model = Review
12
13
14 class UsersListView(generic.ListView):
15     model = User
```

### 2.3 Пользовательский интерфейс

Пользовательский интерфейс разрабатывается непосредственно на HTML, посредством добавления html-файлов в проект веб-сайта и привязки их к файлам Python.

Для отображения на сервере информации для пользователя создаем html- и css-файлы в проекте и описываем в них необходимую разметку страницы.

```
<!DOCTYPE html>
<html lang="en">
<head>

  {% block title %}<title>My Travel</title>{% endblock %}
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
   ↪  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script
   ↪  src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script
  <script
   ↪  src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></scri

  <!-- Добавление дополнительного статического CSS файла -->
  {% load static %}
  <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>

<body>
...
</body>
</html>
```

CSS-файл:

```
.sidebar-nav {
    margin-top: 20px;
    padding: 0;
    list-style: none;
}
```

Оформление домашней страницы:

```
{% extends "base_generic.html" %}

{% block content %}
<h1>My Travel Home</h1>

```

```
  <p>Welcome to <em>My Travel</em> Here you can read information and reviews
   ↪  about the countries of the world, and
  choose where you want to go</p>

<h2>Dynamic content</h2>

  <p>You can find here information about next things:</p>
  <ul>
    <li><strong>Countries:</strong> {{ num_countries }}</li>
    <li><strong>Reviews:</strong> {{ num_reviews }}</li>
    <li><strong>Users:</strong> {{ num_users }}</li>
  </ul>

{% endblock %}
```

В данном фрагменте мы объявили шаблонные переменные для информации, которую мы получаем из соответствующего отображения.

Также мы создали шаблоны для каждой из основных страниц, чтобы корректно выводить список объектов: шаблон страницы стран, шаблон страницы отзывов, шаблон страницы пользователей(см. в приложении).

Главная страница выглядит так(см. рисунок 2.9).

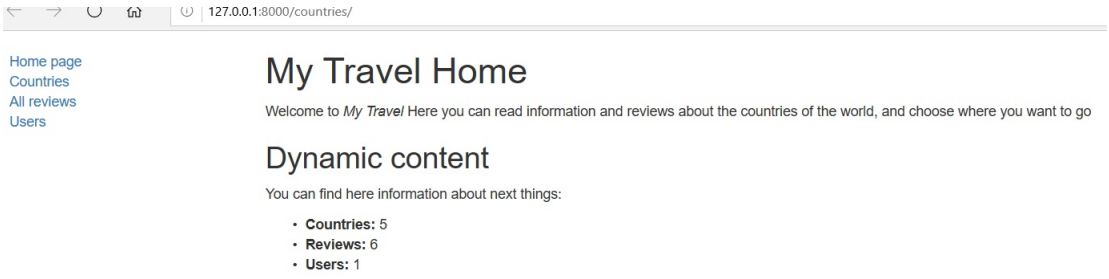


Рисунок 2.9 – Главная страница клиентской части

Страница списка пользователей выглядит так(см. рисунок 2.10).

Пользователи могут зайти на свой аккаунт или покинуть его, сменить пароль от аккаунта. Для этого реализованы шаблоны работы с данными пользователя(см. рисунки 2.11, 2.12, 2.13).

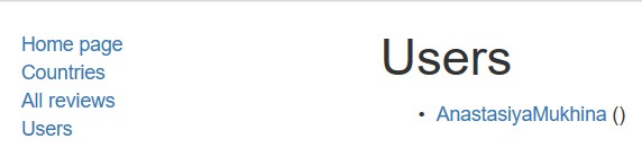


Рисунок 2.10 – Список пользователей

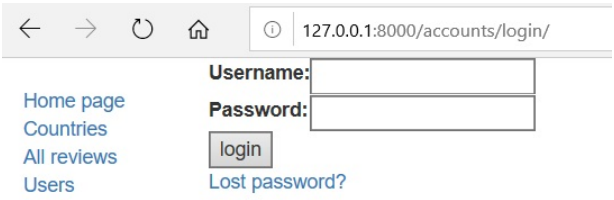


Рисунок 2.11 – Вход в систему

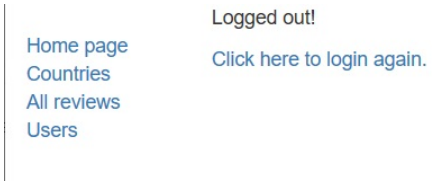


Рисунок 2.12 – Выход из системы


Рисунок 2.13 – Смена пароля

Django предоставляет систему аутентификации и авторизации пользователя на основе фреймворка работы с сессиями. Система аутентификации и авторизации позволяет проверять учетные данные пользователей и определять какие действия какой пользователь может выполнять. Данный фреймворк включает в себя встроенные модели для Пользователей и Групп, непосредственно саму систему прав доступа, флаги, которые определяют, может ли пользователь выполнить задачу, с какой формой и отображением для авторизованых пользователей, а также получить доступ к контенту с ограниченым доступом. Система аутентификации является очень гибкой и позволяет вам формировать свои собственные URL-адреса, формы, отображения, а также шаблоны страниц.

Шаблон входа в аккаунт показывает форму, в которую вводится имя

пользователя и пароль. Если значения полей формы будут недопустимыми, пользователю будет предложено ввести правильные значения, когда страница обновится. Если попытка войти в систему будет успешной, пользователь будет перенаправлен на домашнюю страницу. Код шаблона входа в аккаунт выглядит так:

```
{% extends "base_generic.html" %}

{% block content %}

{% if form.errors %}
  <p>Your username and password didn't match. Please try again.</p>
{% endif %}

{% if next %}
  {% if user.is_authenticated %}
    <p>Your account doesn't have access to this page. To proceed,
    please login with an account that has access.</p>
  {% else %}
    <p>Please login to see this page.</p>
  {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}
<table>

<tr>
  <td>{{ form.username.label_tag }}</td>
  <td>{{ form.username }}</td>
</tr>

<tr>
  <td>{{ form.password.label_tag }}</td>
  <td>{{ form.password }}</td>
</tr>
</table>

<input type="submit" value="login" />
<input type="hidden" name="next" value="{{ next }}" />
</form>
```

```
36
37 {# Assumes you setup the password_reset view in your URLconf #}
38 <p><a href="{% url 'password_reset' %}">Lost password?</a></p>
39
40 {% endblock %}
```

Код остальных шаблонов представлен в приложении.

# ЗАКЛЮЧЕНИЕ

В настоящей работе представлены компоненты разработки веб-приложения с использованием скриптового языка программирования Python, а именно:

- написание серверной части,
- создание пользовательского интерфейса,
- использование базы данных,
- реализация взаимодействия пользователей и приложения.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 *Лутц, М.* Изучаем Python / М. Лутц. — Москва: Диалектика, 2019.

2 Отличительные черты скриптовых языков [Электронный ресурс]. — URL: https://www.opennet.ru/docs/RUS/rpm_guide/111.html (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

3 Скрипт (script): что это такое, сценарные языки в программировании [Электронный ресурс]. — URL: wiki.rookee.ru/script/ (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

4 Скриптовые языки программирования. [Электронный ресурс]. — URL: http://bravit.rsu.ru/history/stud/Scripting%20languages.pdf (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

5 Скриптовый язык [Электронный ресурс]. — URL: https://dic.academic.ru/dic.nsf/ruwiki/42896 (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

6 3.6 Скриптовые языки программирования [Электронный ресурс]. — URL: https://studfile.net/preview/5440797/page:20/ (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

7 Как работают веб-приложения [Электронный ресурс]. — URL: https://habr.com/ru/post/450282/ (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

8 *Прохоренок, Н. А.* Python 3 и PyQt Разработка приложений / Н. А. Прохоренок. — Санкт-Петербург: БХВ-Петербург, 2012.

9 Общие сведения о веб-приложениях [Электронный ресурс]. — URL: https://helpx.adobe.com/ru/dreamweaver/using/web-applications.html (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

10 HTML, CSS, PHP, JavaScript, SQL – что и зачем? [Электронный ресурс]. — URL: http://codeharmony.ru/materials/125 (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

11 WEB-ПРИЛОЖЕНИЯ [Электронный ресурс]. — URL: http://www.council.ru/services/zakaznaya-razrabotka/web-prilozheniya/ (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

12 Веб-фреймворк Django (Python) [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

13 Структура веб-приложения [Электронный ресурс]. — URL: http://labaka.ru/likbez/struktura-veb-prilozheniya (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

14 Изучение HTML: руководства и уроки [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/Learn/HTML (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

15 HTML [Электронный ресурс]. — URL: https://ru.hexlet.io/courses/intro_to_web_development/lessons/html/theory_unit (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

16 Что такое веб-сервер [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/Learn/%D0%A7%D1%82%D0%BE_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5_%D0%B2%D0%B5%D0%B1_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80 (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

17 Создание сайта на Python/Django: подбор хостинга [Электронный ресурс]. — URL: https://igorosa.com/sozdanie-sajta-na-python-django-podbor-xostinga/ (Дата обращения 25.05.2020). Загл. с экр. Яз. рус.

## ПРИЛОЖЕНИЕ А

## CD-диск с отчетом о выполненной работе

На приложенном диске можно ознакомиться со следующими файлами:

**Папка** `Coursework` — LaTeX- вариант курсовой работы;

**Папка** `TravelCatalog` — папка проекта на языке Python;

`Coursework.pdf` — курсовая работа.

# ПРИЛОЖЕНИЕ Б

## Код программы

TravelCatalog countries admin.py

```python
from .models import Review, Country, User

# Register your models here.

# admin.site.register(Country)
# admin.site.register(User)
# admin.site.register(Review)


# Define the admin class
class UserAdmin(admin.ModelAdmin):
    list_display = ('username', 'origin_country', 'date_of_birth')


# Register the admin class with the associated model
admin.site.register(User, UserAdmin)


# Define the admin class
class CountryAdmin(admin.ModelAdmin):
    list_display = ('title', 'information', 'attractions')


# Register the admin class with the associated model
admin.site.register(Country, CountryAdmin)


# Define the admin class
class ReviewAdmin(admin.ModelAdmin):
    list_display = ('countryRev', 'text_of_review', 'usernameRev')
    list_filter = ('countryRev', 'usernameRev')


# Register the admin class with the associated model
admin.site.register(Review, ReviewAdmin)
```

TravelCatalog countries apps.py

```python
from django.apps import AppConfig


class CountriesConfig(AppConfig):
    name = 'countries'
```

TravelCatalog countries models.py

```python
class Country(models.Model):
    """
    Model representing a country.
    """
    title = models.CharField(max_length=30)
    information = models.TextField(max_length=1000, help_text="Enter a brief description of the country")
    attractions = models.TextField(max_length=1000, help_text='Enter a brief description of the attractions')
    country_reviews = models.ManyToManyField(Review, help_text="Select a review about the country")

    def __str__(self):
        """
        String for representing the Model object.
        """
        return self.title

    def get_absolute_url(self):
        """
        Returns the url to access a particular book instance.
        """
        return reverse('country-detail', args=[str(self.id)])


class User(models.Model):
    """
    Model representing an user.
    """
    username = models.CharField(max_length=100)
    origin_country = models.CharField(max_length=100)
    date_of_birth = models.DateField(null=True, blank=True)
    user_reviews = models.ManyToManyField(Review, help_text="Write your review about the country")
```

```
31
32      def __str__(self):
33          """
34          String for representing the Model object.
35          """
36          return self.username
37
38      def get_absolute_url(self):
39          """
40          Returns the url to access a particular author instance.
41          """
42          return reverse('user-detail', args=[str(self.id)])
```

TravelCatalog countries urls.py

```
1  from django.urls import path
2  from . import views
3  from django.conf.urls import url
4
5
6  urlpatterns = [
7      url(r'^$', views.index, name='index'),
8      url(r'^country/$', views.CountryListView.as_view(), name='country'),
9      url(r'^country_details/(\d+)$', views.CountryDetailView.as_view(),
       ↪  name='country-details'),
10     url(r'^reviews/$', views.ReviewsListView.as_view(), name='reviews'),
11     url(r'^user/$', views.UsersListView.as_view(), name='user'),
12 ]
```

TravelCatalog countries views.py

```
1  from django.shortcuts import render
2  from .models import Country, Review, User
3  from django.views import generic
4
5  # Create your views here.
6
7
8  def index(request):
9      num_countries=Country.objects.all().count()
10     num_reviews=Review.objects.all().count()
11     num_users=User.objects.all().count()
```

```
12
13      # Отрисовка HTML-шаблона index.html с данными внутри
14      # переменной контекста context
15      return render(
16          request,
17          'index.html',
18          context={'num_countries': num_countries, 'num_reviews': num_reviews,
            ↪  'num_users': num_users},
19      )
20
21
22  class CountryListView(generic.ListView):
23      model = Country
24      paginate_by = 10
25
26
27  class CountryDetailView(generic.DetailView):
28      model = Country
29
30
31  class ReviewsListView(generic.ListView):
32      model = Review
33
34
35  class UsersListView(generic.ListView):
36      model = User
```

TravelCatalog countries templates base_generic.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4
5    {% block title %}<title>My Travel</title>{% endblock %}
6    <meta charset="utf-8">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="stylesheet"
     ↪  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
9    <script
     ↪  src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script
10   <script
     ↪  src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></scri
```

```
11
12    <!-- Добавление дополнительного статического CSS файла -->
13    {% load static %}
14    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
15  </head>
16
17  <body>
18
19    <div class="container-fluid">
20
21      <div class="row">
22        <div class="col-sm-2">
23        {% block sidebar %}
24        <ul class="sidebar-nav">
25            <li><a href="{% url 'index' %}">Home page</a></li>
26            <li><a href="{% url 'country' %}">Countries</a></li>
27            <li><a href="{% url 'reviews' %}">All reviews</a></li>
28            <li><a href="{% url 'user' %}">Users</a></li>
29        </ul>
30       {% endblock %}
31        </div>
32        <div class="col-sm-10 ">
33        {% block content %}{% endblock %}
34        {% block pagination %}
35    {% if is_paginated %}
36        <div class="pagination">
37            <span class="page-links">
38                {% if page_obj.has_previous %}
39                    <a href="{{ request.path }}?page={{
                      ↪  page_obj.previous_page_number }}">previous</a>
40                {% endif %}
41                <span class="page-current">
42                    Page {{ page_obj.number }} of {{
                      ↪  page_obj.paginator.num_pages }}.
43                </span>
44                {% if page_obj.has_next %}
45                    <a href="{{ request.path }}?page={{
                      ↪  page_obj.next_page_number }}">next</a>
46                {% endif %}
47            </span>
48        </div>
```

```
  {% endif %}
{% endblock %}
      </div>
    </div>

  </div>
</body>
</html>
```

[TravelCatalog countries templates index.html

```
{% extends "base_generic.html" %}

{% block content %}
<h1>My Travel Home</h1>

  <p>Welcome to <em>My Travel</em> Here you can read information and reviews
  ↪  about the countries of the world, and
  choose where you want to go</p>

<h2>Dynamic content</h2>

  <p>You can find here information about next things:</p>
  <ul>
    <li><strong>Countries:</strong> {{ num_countries }}</li>
    <li><strong>Reviews:</strong> {{ num_reviews }}</li>
    <li><strong>Users:</strong> {{ num_users }}</li>
  </ul>

{% endblock %}
```

TravelCatalog countries templates countries country_details.html

```
{% extends "base_generic.html" %}

{% block content %}

<h1>Title: {{ country.title }}</h1>

<p><strong>Information:</strong> {{ country.information }}</p>
<p><strong>Attractions:</strong> {{ country.attractions }}</p>
<p><strong>Reviews:</strong> {% for reviews in country.country_reviews.all %}
  ↪  {{ country.country_reviews }}
```

```
    {% if not forloop.last %}, {% endif %}{% endfor %}</p>

{% endfor %}

{% endblock %}
```

TravelCatalog countries templates countries country_list.html

```
{% extends "base_generic.html" %}

{% block content %}
    <h1>Countries</h1>

    {% if country_list %}
    <ul>

      {% for countries in country_list %}
      <li>
        <a href="{{ country.get_absolute_url }}">{{ country.title }}</a>
        ↪ ({{country.information}})
      </li>
      {% endfor %}

    </ul>
    {% else %}
      <p>There are no countries.</p>
    {% endif %}
{% endblock %}
```

TravelCatalog countries templates countries review_list.html

```
{% extends "base_generic.html" %}

{% block content %}
    <h1>Reviews</h1>

    {% if object_list %}
    <ul>

      {% for reviews in object_list %}
      <li>
        <a href="{{ review.get_absolute_url }}">{{ review.text_of_review
        ↪ }}</a> ({{review.countryRev}})
```

```
      </li>
      {% endfor %}

    </ul>
    {% else %}
      <p>There are no reviews.</p>
    {% endif %}
{% endblock %}
```

TravelCatalog countries templates countries user_list.html

```
{% extends "base_generic.html" %}

{% block content %}
    <h1>Users</h1>

    {% if object_list %}
    <ul>

      {% for book in object_list %}
      <li>
        <a href="{{ user.get_absolute_url }}">{{ user.username }}</a>
        ↪ ({{user.origin_country}})
      </li>
      {% endfor %}

    </ul>
    {% else %}
      <p>There are no users.</p>
    {% endif %}
{% endblock %}
```

TravelCatalog countries static css styles.css

```
.sidebar-nav {
    margin-top: 20px;
    padding: 0;
    list-style: none;
}
```

TravelCatalog templates registration logged_out.html

```
{% extends "base_generic.html" %}

{% block content %}
<p>Logged out!</p>

<a href="{% url 'login'%}">Click here to login again.</a>
{% endblock %}
```

TravelCatalog templates registration login.html

```
{% extends "base_generic.html" %}

{% block content %}

{% if form.errors %}
  <p>Your username and password didn't match. Please try again.</p>
{% endif %}

{% if next %}
  {% if user.is_authenticated %}
    <p>Your account doesn't have access to this page. To proceed,
    please login with an account that has access.</p>
  {% else %}
    <p>Please login to see this page.</p>
  {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}
<table>

<tr>
  <td>{{ form.username.label_tag }}</td>
  <td>{{ form.username }}</td>
</tr>

<tr>
  <td>{{ form.password.label_tag }}</td>
  <td>{{ form.password }}</td>
</tr>
</table>

```

```
<input type="submit" value="login" />
<input type="hidden" name="next" value="{{ next }}" />
</form>

{# Assumes you setup the password_reset view in your URLconf #}
<p><a href="{% url 'password_reset' %}">Lost password?</a></p>

{% endblock %}
```

TravelCatalog templates registration password_reset_confirm.html

```
{% extends "base_generic.html" %}

{% block content %}

    {% if validlink %}
        <p>Please enter (and confirm) your new password.</p>
        <form action="" method="post">
            {% csrf_token %}
            <table>
                <tr>
                    <td>{{ form.new_password1.errors }}
                        <label for="id_new_password1">New
                        ↪  password:</label></td>
                    <td>{{ form.new_password1 }}</td>
                </tr>
                <tr>
                    <td>{{ form.new_password2.errors }}
                        <label for="id_new_password2">Confirm
                        ↪  password:</label></td>
                    <td>{{ form.new_password2 }}</td>
                </tr>
                <tr>
                    <td></td>
                    <td><input type="submit" value="Change my password"
                    ↪  /></td>
                </tr>
            </table>
        </form>
    {% else %}
        <h1>Password reset failed</h1>
        <p>The password reset link was invalid, possibly because it has
        ↪  already been used. Please request a new password reset.</p>
```

```
    {% endif %}

{% endblock %}
```

TravelCatalog templates registration password_reset_done.html

```
{% extends "base_generic.html" %}
{% block content %}
<p>We've emailed you instructions for setting your password. If they haven't
 ↪  arrived in a few minutes, check your spam folder.</p>
{% endblock %}
```

TravelCatalog templates registration password_reset_email.html

```
Someone asked for password reset for email {{ email }}. Follow the link below:
{{ protocol}}://{{ domain }}{% url 'password_reset_confirm' uidb64=uid
 ↪  token=token %}
```

TravelCatalog templates registration password_reset_form.html

```
{% extends "base_generic.html" %}
{% block content %}

<form action="" method="post">{% csrf_token %}
    {% if form.email.errors %} {{ form.email.errors }} {% endif %}
        <p>{{ form.email }}</p>
    <input type="submit" class="btn btn-default btn-lg" value="Reset password"
     ↪  />
</form>

{% endblock %}
```

TravelCatalog TravelCatalog asgi.py

```
"""
ASGI config for TravelCatalog project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/
"""
```

```
9
10 import os
11
12 from django.core.asgi import get_asgi_application
13
14 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'TravelCatalog.settings')
15
16 application = get_asgi_application()
```

TravelCatalog TravelCatalog settings.py

```
1  """
2  Django settings for TravelCatalog project.
3
4  Generated by 'django-admin startproject' using Django 3.0.6.
5
6  For more information on this file, see
7  https://docs.djangoproject.com/en/3.0/topics/settings/
8
9  For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/3.0/ref/settings/
11 """
12
13 import os
14
15 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
16 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
17
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = '87k%ae6mve5ydqo9b49j*9q+t!@b%u*snth_*9jy&8ahbe+!_8'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
```

```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'countries.apps.CountriesConfig',
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50     'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
52
53 ROOT_URLCONF = 'TravelCatalog.urls'
54
55 TEMPLATES = [
56     {
57         'BACKEND': 'django.template.backends.django.DjangoTemplates',
58         'DIRS': [os.path.join(BASE_DIR, 'templates')],
59         'APP_DIRS': True,
60         'OPTIONS': {
61             'context_processors': [
62                 'django.template.context_processors.debug',
63                 'django.template.context_processors.request',
64                 'django.contrib.auth.context_processors.auth',
65                 'django.contrib.messages.context_processors.messages',
66             ],
67         },
68     },
69 ]
70
71 WSGI_APPLICATION = 'TravelCatalog.wsgi.application'
72
```

```
73
74 # Database
75 # https://docs.djangoproject.com/en/3.0/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.sqlite3',
80         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
81     }
82 }
83
84
85 # Password validation
86 # https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators
87
88 AUTH_PASSWORD_VALIDATORS = [
89     {
90         'NAME':
          ↪  'django.contrib.auth.password_validation.UserAttributeSimilarityValidator
91     },
92     {
93         'NAME':
          ↪  'django.contrib.auth.password_validation.MinimumLengthValidator',
94     },
95     {
96         'NAME':
          ↪  'django.contrib.auth.password_validation.CommonPasswordValidator',
97     },
98     {
99         'NAME':
          ↪  'django.contrib.auth.password_validation.NumericPasswordValidator',
100     },
101 ]
102
103
104 # Internationalization
105 # https://docs.djangoproject.com/en/3.0/topics/i18n/
106
107 LANGUAGE_CODE = 'en-us'
108
109 TIME_ZONE = 'Europe/Samara'
```

```
110
111 USE_I18N = True
112
113 USE_L10N = True
114
115 USE_TZ = True
116
117
118 # Static files (CSS, JavaScript, Images)
119 # https://docs.djangoproject.com/en/3.0/howto/static-files/
120
121 STATIC_URL = '/static/'
122
123 # Redirect to home URL after login (Default redirects to /accounts/profile/)
124 LOGIN_REDIRECT_URL = '/'
```

TravelCatalog TravelCatalog urls.py

```
1  """TravelCatalog URL Configuration
2
3  The `urlpatterns` list routes URLs to views. For more information please see:
4      https://docs.djangoproject.com/en/3.0/topics/http/urls/
5  Examples:
6  Function views
7      1. Add an import:  from my_app import views
8      2. Add a URL to urlpatterns:  path('', views.home, name='home')
9  Class-based views
10     1. Add an import:  from other_app.views import Home
11     2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
12 Including another URLconf
13     1. Import the include() function: from django.urls import include, path
14     2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import include
18 from django.urls import path
19 from django.views.generic import RedirectView
20 from django.conf import settings
21 from django.conf.urls.static import static
22
23 urlpatterns = [
24     path('admin/', admin.site.urls),
```

```
    path('countries/', include('countries.urls')),
    path('', RedirectView.as_view(url='/countries/', permanent=True)), ] +
    ↪ static(settings.STATIC_URL,


# Add Django site authentication urls (for login, logout, password management)
urlpatterns += [
    path('accounts/', include('django.contrib.auth.urls')),
]
```

TravelCatalog TravelCatalog wsgi.py

```
"""
WSGI config for TravelCatalog project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'TravelCatalog.settings')

application = get_wsgi_application()
```

TravelCatalog manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'TravelCatalog.settings')
    try:
        from django.core.management import execute_from_command_line
```

```
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```