**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”  
(УНИВЕРСИТЕТ ИТМО)**

**ЦЕНТР АВТОРИЗОВАННОГО ОБУЧЕНИЯ ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ**

**ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА**

«Музыкальная социальная сеть»

Автор Вихров Максим Вячеславович Вихров

(Фамилия Имя Отчество) (Подпись)

**Центр авторизованного обучения информационным технологиям**

Наименование программы **«Python-разработчик»**

Руководитель Кузьмин Константин Михайлович \_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

**К защите допустить**

|  |  |  |
| --- | --- | --- |
| *Заместитель директора ЦАО ИТ,*  *к.т.н.* |  | / *Т.В. Зудилова*/ |

Санкт-Петербург, 2023г.

Обучающийся\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа 124/29

(Фамилия, И. О.)

Работа принята «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.

Работа выполнена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.

Секретарь ИАК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, И. О.) (подпись)

Листов хранения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Демонстрационных материалов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОГЛАВЛЕНИЕ**

[Введение 4](#_Toc130222770)

[Цель 5](#_Toc130222771)

[Описание системы 7](#_Toc130222772)

[Функции системы 17](#_Toc130222773)

[Структура данных 31](#_Toc130222774)

[Заключение 34](#_Toc130222775)

[Литература 35](#_Toc130222776)

# Введение

Разработанный проект музыкальной социальный сети обладает высокой актуальностью в сфере цифровых технологий. В течение всей истории развития человечества музыка являлась средством привлечения и объединения масс. Своей многогранностью она охватывала миллионы человек, живущих на земле. Каждому находилось что-то по душе и что привлекало не только к прослушиванию вновь и вновь, но и к обсуждению, к дискуссиям и поиску новых музыкальных произведений подобного жанра. Благодаря к способности музыки вызывать чувства и своей открытости, в мире все больше появлялось объединений, посвященных музыкальным группам, жанрам, исполнителям и любителей музыкального досуга.

За все свое существование накопились миллионы музыкальных произведений, которые впоследствии стали, подобно эволюции человечества за счет расширения генотипов, содержать невероятно широкий спектр присущих качеств, формирующих уникальный вкус от прослушивания. Эксперименты, история, мода, технологии – все это факторы влияния на развитие музыки.

В современном мире трудно представить человека, у которого нет предпочтений к прослушиванию музыкальных произведений. Напротив считается, что музыкальный вкус эволюционирует с развитием человека и формируется совместно с прохождением им определенных жизненных этапов. То, что он предпочитает слушать в настоящий момент времени непосредственно связано с чувствами и эмоциями, которые ему присущи, или в которых он нуждается. Музыкальное собрание определенного человека в абстрактном смысле раскрывает его личность, о которой может сложить впечатление.

Приложение “Музыкальная социальная сеть” призвано обеспечить такую цифровую площадку, в которой основой для пользовательских взаимосвязей будут именно музыкальные вкусы.

Как средство для реализации подобной площадки мною было выбрано веб-приложение, основанное на использовании фреймворка Django. Для хранения пользовательских данных я выбрал MySQL. А с целью упрощения развертывания использовался Docker.

# Цель

Целью дипломной работы являлось изучить и использовать следующие технологии применимые к языку программирования Python.

Django Framework: это высокоуровневый веб-фреймворк на языке python, который позволяет создавать сложные веб-приложения. Он предоставляет возможности для решения готовых задач, таких как обработка HTTP-запросов, управление базами данных, аутентификация и авторизация пользователей. Благодаря принципам DRY и архитектурного шаблона MVC и различным компонентам Django, возможно решать лаконично решать типичные задачи и разделять код на отдельные слои.

MySQL: это открытая реляционная система управления базами данных (СУБД), которая позволяет хранить и управлять данными в организованном виде. Выбор в пользу MySQL основывался на том, что система предоставляет высокую производительность и масштабируемость и используется в обширном количестве различных отраслей.

Docker: это платформа для разработки, доставки и запуска приложений в контейнерах. С помощью инструмента приложение оказывается в изолированной виртуальной среде (контейнер), что делает его легковесным и мобильным для разработки, тестирования и развертывания. Собирая сервисы в контейнеры и используя инструмент Docker Сompose, становится проще управлять развертыванием приложения.

HTML/CSS: это языки для написания веб-страниц. HTML (HyperText Markup Language) используется для описания структуры и содержания веб-страницы, располагая текст, изображения, ссылки и другие элементы. CSS (Cascading Style Sheets) используется для определения внешнего вида и оформления веб-страницы, таких как цвета, шрифты, размеры и расположение элементов.

С помощью вышеописанных инструментов главной задачей дипломной работы являлась разработка веб-приложения, позволяющее пользователям осуществлять поиск музыкальных произведений, исполнителей и взаимодействовать с ними, искать и находить пользователей с похожими музыкальными интересами и предоставить возможность для обмена мнениями.

Основные функциональные возможности:

1. Регистрация и аутентификация пользователей, возможность создания личного кабинета и управление им.
2. Поиск музыкальных произведений, исполнителей и пользовательских профилей по соответствующему названию, просмотр их детальной информации.
3. Возможность добавления и удаления музыкальных произведений в личный профиль.
4. Создание плейлистов и возможность оставлять комментарии.
5. Подборка трендовых произведений и исполнителей.
6. Персональные рекомендации для пользователей.

Дополнительные возможности:

1. Прослушивание треков
2. Просмотр в новостной ленте действий пользователей из персональных подписок.

# Описание системы

Технические средства

С целью реализации проекта веб-сервиса потребовалось использование следующих программных библиотек языка Python:

* Django (v4.1.7)
* pyhton-dotenv (v0.21.1)
* mysql-connector-python (v8.0.32)
* requests (v.2.28.2)

Библиотека «Django» являлась основной библиотекой нашего веб-сервиса. Выбранный фреймворк содержит внутри множество вспомогательных библиотек и обеспечил эффективный процесс разработки. Основные технические возможности фреймворка заключаются в предоставлении следующих инструментов:

1. Административный интерфейс. С помощью данного инструмента была возможность управления контентом приложения и данными, содержащихся в базе данных. Дополнительного написания кода не требовалось.
2. ORM (Object Relative Mapping). Инструмент, позволяющий взаимодействовать с базой данных, используя объектно-ориентированный подход.
3. URL – адресация. С помощью данного инструмента была упрощена работа по маршрутизации и обработки запросов.

Помимо выделенных инструментов разработка с использованием фреймворка Django обладала следующими преимуществами: встроенные элементы защиты, обеспечивающих защиту от CSRF - атак и SQL инъекций, гибкость при масштабировании и доступность интеграции с другими технологиями.

Библиотека «python-dotenv» обеспечивала работу с переменными окружения. Работа с некоторыми переменными окружения требовала их сокрытия при размещении версии в публичном доступе и сборке контейнеров, а также являлись константами. Данная библиотека позволяет разместить переменные подобного рода в файле с названием «.env», скрытого от инструмента контроля версий git, и загружать их по мере необходимости в модулях и исполнительный файлах из локального виртуального окружения.

Библиотека «mysql-connector-python» была необходимой в использовании нашего веб-сервиса совместно с базой данных MySQl. В конфигурационном модуле «project/settings.py» прописываются параметры для использования данной библиотеки в ядре приложения Django, который в свою очередь будет использовать это для предоставления ORM.

Библиотека «requests» использовалась при API запросах к музыкальному сервису «musixmatch» с целью получения данных о чартах, музыкальных произведениях, исполнителях.

Во время разработки приложения использовалась открытая реляционная база данных MySQL. Использование данной базы обладало рядом преимуществ: удобное взаимодействие с базой с помощью MySQL workbench, написание SQL запросов напрямую, проверка состояния миграций моделей, графический редактор таблиц моделей.

Docker использовался для развертывания нашего приложения сперва на локальном машине, а в последствии на виртуальной. Использование данной программы позволило осуществить сборку нашего приложения в контейнер с изолированной виртуальной средой и объединить при помощи docker compose в единую виртуальную сеть совместно с подобным контейнером, содержащим исключительно базу данных MySQL. Благодаря данному действию, появилась изолированная сеть среди контейнеров, которая ограничена от внешних факторов и очень легко поддается управлению.

Структура системы

Структура нашей системы выстраивается таким образом, что корневой путь нашего проекта содержит следующие файлы и директории.

venv – директория, в которой содержится созданное изолированное виртуальное окружение, хранящее в себе необходимые версии установленных библиотек.

app – директория нашего приложения, входящее в состав проекта веб-сервиса:

* app/migrations/ – пакет, содержащий в себе модули с данными о миграциях моделей таблиц в базу данных. Первоначальная инициализация и последующие корректировки моделей формируются в модули, с которых происходит перенос непосредственно в таблицы БД.
* app/static/ – директория, предназначенная для хранения статических файлов, таких как изображения, файлы стилей CSS, скрипты JavaScript и другие файлы, которые не требуют обработки сервером.
* app/templates/ – директория, используемая для хранения шаблонов, которые используются для генерации динамического контента на веб-страницах и содержащие HTML-код, который может быть дополнен различными фрагментами кода на языке шаблонов Django, такими как переменные, циклы, условия и другие конструкции.
* app/admin.py – модуль используется для настройки административного интерфейса приложения, позволяющего управлять данными приложения через веб-интерфейс, без необходимости написания кода.
* app/apps.py - модуль является частью структуры приложения в Django и предназначен для настройки конфигурации приложения.
* app/forms.py - модуль отвечает за определение форм в виде классов, которые используются в приложении для отправления данные на сервер.
* app/mixins.py – файл, содержащий в себе классы-миксины, используемые во вспомогательных целях с целью избежания повторения кода и обеспечивающие определенную функциональность тем объектам, где они используются.
* app/models.py – модуль, отвечающий за определение моделей данных, которые определяют структуру данных и правила их взаимодействия с базой данных. Каждая модель обычно соответствует таблице в базе данных и содержит поля, которые соответствуют столбцам в этой таблице.
* app/tests.py – файл, предназначенный для написания автоматических тестов для приложения.
* app/urls.py - модуль содержит определения маршрутов (URL-адресов), которые используются в приложении. Маршруты определяют соответствие между URL-адресом и представлением обработчика, который должен быть вызван при обращении по данному адресу.
* app/views.py - файл содержит определения представлений, которые обрабатывают запросы от пользователей и возвращают ответы в виде HTTP-ответов. Представления являются основными компонентами веб-приложения и определяют, какие данные должны быть отображены на странице, как пользователь может взаимодействовать с этими данными и какие действия должны быть выполнены в ответ на запрос пользователя.

project – директория, содержит основные файлы и конфигурации проекта:

* project/asgi.py - файл, используемый для настройки ASGI-совместимого сервера, который может быть использован для развертывания асинхронных Django-приложений на сервере.
* project/wsgi.py - файл, используемый для настройки WSGI-совместимого сервера, который может быть использован для развертывания Django-приложения на сервере.
* project/urls.py - файл, содержащий определения маршрутов (URL-адресов), которые используются в проекте, а также включения маршрутов из других приложений.
* project/settings.py - файл, содержащий настройки проекта, такие как база данных, статические файлы, шаблоны и другие компоненты. Этот файл также содержит настройки безопасности, такие как секретный ключ и список разрешенных хостов.

manage.py - файл, используемый для управления проектом, такой как запуск локального сервера разработки, создание миграций базы данных и другие команды.

.dockerignore – файл, в который записываются файлы и директории игнорируемые докером при сборке записи приложения.

.gitignore – файл, в который записываются файлы и директории игнорируемые инструментом контроля версий git

.env – файл, в который записываются переменные окружения, константы, токены, которые впоследствии оказываются в виртуальной среде.

requirements.txt – файл, в котором записываются установленные библиотеки и указываются их версии.

db.py – скрипт, который подключается к БД и создает в ней необходимую базу.

docker-compose.yml – файл, описывающий процесс создания docker compose контейнера, управляющим контейнером с БД и контейнером нашего приложения.

Dockerfile – файл, в котором записан алгоритм сборки контейнера приложения.

Makefile – файл, который содержит набор правил и инструкций для автоматической сборки, тестирования и установки программного обеспечения.

Руководство пользователя

Пользователям программы доступен обширный список вариантов взаимодействия с музыкальным контентом и контентом других пользователей. С помощью веб-браузера при подключении по HTTP протоколу к IP адресу сервера (в стадии разработки дополнительно указывается порт), хранящего приложение, пользователю предоставится веб-страница с приветствием, и это будет являться началом пользования приложением.

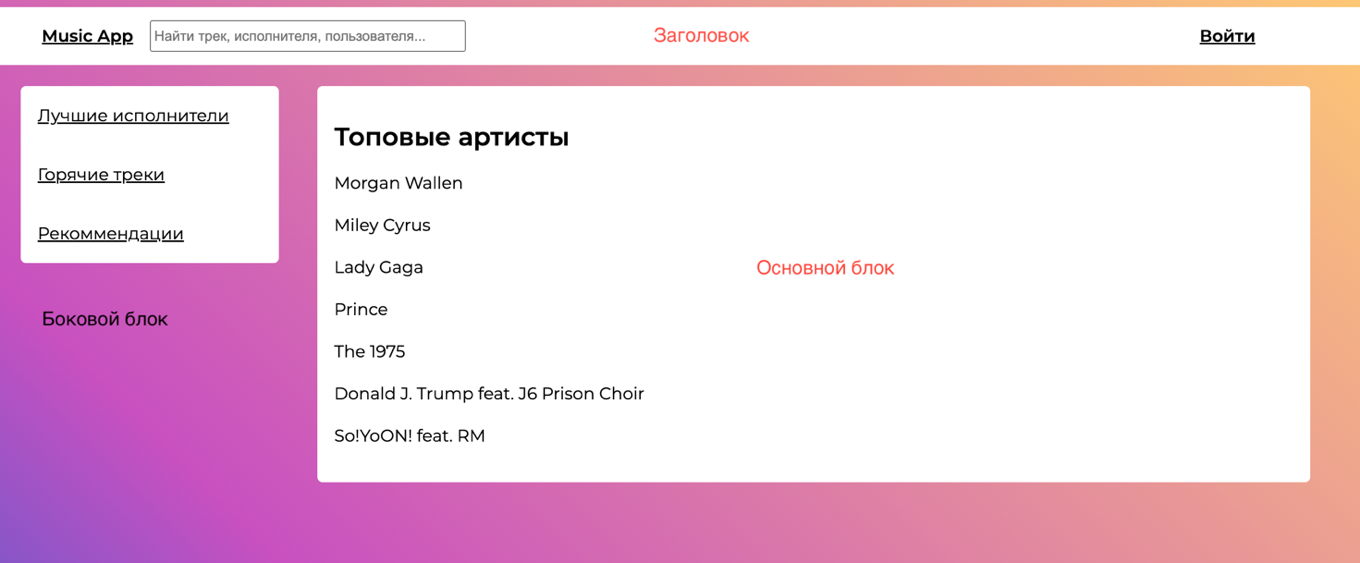
Основные возможности пользователей, не прошедших этап регистрации, или аутентификации:

* Осуществить регистрацию на сервисе
* Осуществить процесс аутентификации
* Осуществить поиск контента, включающего в себя треки по названию, исполнителей по имени, пользователей по никнейму.
* Просмотреть профили пользователей и их контент (треки, плейлисты)
* Просмотреть содержание плейлиста
* Просмотреть подборку трендовых исполнителей и музыкальных произведений

Список возможностей гораздо расширяется, если пользователь зарегистрирован и аутентифицирован:

* Добавить трек в профиль
* Добавить плейлист в профиль
* Создать плейлист
* Оставить комментарий к плейлисту/удалить
* Управлять личным контентом
* Просмотреть персональные рекомендаций пользователей
* Подписаться на пользователя

Сервис предоставляет веб-страницы, в которых используется вложенность блоков контента. Три блока: заголовок, боковой и основной блок.



Заголовок – верхняя часть веб-страницы. В левой части блок содержит в себе: иконку приложения, блок ввода поиска контента, в правой: ссылку на страницу аутентификации. Иконка содержит ссылку, переход по которой приведет к корневому маршруту приложения. Блок поиска позволяет ввести желаемое наименование и с помощью клавиши Enter осуществиться переход на страницу с результатами поиска. В правой части располагаются ссылки меняющиеся динамически в зависимости от статуса аутентификация пользователя. Аутентифицированный пользователь получит возможность перейти по ссылке в свой личный профиль и также рядом будет располагаться ссылка на выход из аутентифицированного статуса. В обратном случае будет ссылка на страницу аутентификации

Боковой блок содержит в себе прямые ссылки к страницам с подборками горячих треков, лучших исполнителей, рекомендациями. У аутентифицированного пользователя дополнительно появится ссылка на страницу создания плейлиста.

Основной блок содержит себе основную визуальную часть контента веб-сервиса, где происходит вывод результатов представлений http запроса по всем маршрутам сервиса. Вместе с этим также в данном блоке происходит вывод вспомогательного текста статусов выполнений различных операций и ошибок.

С целью подробно раскрыть каждую функциональную возможность пройдемся по каждой из них.

Регистрация на сервисе происходит следующим образом. Незарегистрированный пользователь должен пройти по следующему маршруту в адресной строке «accounts/register/». Можно ввести данный маршрут в адресной строке, а можно попасть на форму кликая по следующим ссылкам: «Войти» -> «Зарегистрироваться». Ему предоставится форма, в которой нужно заполнить следующие данные: никнейм, emal, пароль, повторить пароль. Никнейм и email обладают требованиями уникальности. Пароли должны совпадать и быть больше 6 символов. В успешном случае, пользователь попадет в свой личный кабинет, в неуспешном ему вернется форма и будет выведена ошибка.

Аутентификация пользователя происходит по маршруту «accounts/login/». Попасть туда можно за счет клика по ссылке заголовка «Войти». Пользователю предоставиться форма, которую ему необходимо заполнить персональными данными. В успешном случае произойдет переадресация в личный профиль, в обратном вернется форма и будет выведена ошибка.

Осуществление поиска контента происходит следующим образом. В блоке ввода, находящемся на заголовке, необходимо ввести наименование искомой позиции. Это может быть название трека, название исполнителя, полный никнейм пользователя. При нажатии на клавишу «enter», будет выполнена переадресация на страницу «search/search?query=», в которой появятся результаты поиска.

Для того, чтобы просмотреть профили пользователей, необходимо пройти по маршруту «content/user/<int:pk>», где pk означает персональный id пользователя. Во время пользования сервисом в разных местах будут встречаться отметки о пользователях в виде никнеймов. За ними закреплены ссылки, ссылающие на профиль пользователя.

Просмотреть содержание плейлиста можно по маршруту «content/playlist/<int:pk>», где pk означает id плейлиста. Также, как и пользователями за каждой отметкой закреплена нужная ссылка.

Просмотра трендовых музыкальных произведений и исполнителей возможен по маршрутам: «search/top-artists» «search/top-tracks». Сделать это можно таким же образом кликнув по ссылкам в боковом блоке.

Для того, чтобы добавить трек в профиль необходимо быть аутентифицированным пользователем. Выполнив данное условие, появятся кнопки «Добавить» под каждым встретившимся музыкальным треком, за исключением просмотра их в вашем личном профиле. Необходимо нажать на эту кнопку и удостовериться в положительном статусе появившимся в начале основного блока. Трек, который был добавлен ранее, повторно добавить нельзя, об этом напомнит сообщение.

Для того, чтобы отметить понравившийся плейлист и добавить в своей профиль, необходимо быть аутентифицированным пользователем. В этом состоянии будут доступны кнопки «Добавить» в режиме просмотра контента плейлиста.

Создать плейлист могут аутентифицированные пользователи. Для них будет доступна ссылка в боковом блоке, пройдя по которой будет предоставлена форма создания. Необходимо ввести название, по желанию добавить описания и выбрать треки, которые ранее добавлены были в профиль. Возможно создать плейлист без добавления треков. Невозможно создание одним пользователем двух плейлистов с одинаковыми названиями.

Аутентифицированный пользователь может выразить свое мнение о плейлисте, оставив комментарий под ним. В режиме просмотра будет доступен блока ввода сообщения, которое необходимо заполнить и кнопка отправить под ним. Пользователь может удалить свой комментарий в любой момент, а пользователь, который является создателем плейлиста может удалить любые комментарии, в том числе чужих пользователей.

Подписаться на пользователя может аутентифицированный пользователь в режиме просмотра страницы профиля пользователя, на которого он хочет подписаться. В этом режиме будет доступна кнопка «подписаться».

Для просмотра персональных рекомендаций необходимо пройти по маршруту «search/recommendations». Сделать это так же можно кликну по ссылке в боковом меню. На данном этапе разработки предлагаются профили, на которых подписаны профили, подписчиком которых в свою очередь являетесь вы. Требуется аутентификация.

Управление личным контентом доступно аутентифицированным пользователям по маршруту «accounts/personal». Здесь появляются весь контент, отмеченный пользователем, как понравившийся, а также подписки, добавленные треки. В этом режиме появляются кнопки «Удалить», или «Отписаться», что позволит хозяину страницы удалить контент, который больше не желаем на странице.

Совокупность всех возможностей разработанного сервиса позволяет пользователям находить новые музыкальные произведения, делиться, обсуждать среди таких же любителей и ценителей музыки, создавая тем самым персональный мир вокруг себя из людей, находящихся на одной музыкальной волне.

# Функции системы

По умолчанию библиотека Django предоставляет нам огромный набор инструментов для обработки http запросов и генерация ответов. На протяжении всей разработки веб-сервиса данные инструменты в виде объектов функций, классов использовались практически в каждом созданном блоке кода. Стоит отметить данные объекты и описать присущие им особенности.

Первоначальная настройка работы нашего веб-сервиса определяется в модуле «project/settings.py», где определяются различные переменные, передающиеся в ядро библиотеки Django и обеспечивающие работоспособность сервиса в целом, взаимосвязь между различными структурами и определение поведения алгоритма.

Работа над проектом складывалась таким образом, что после выполнения необходимых настроек, связанных с базой данных и конфигурационным файлом, далее необходимо было определить URL маршруты и для каждого разработать представление в виде обработчика запроса.

Взаимосвязь между URL маршрутом и обработчиком обеспечивалась в файлах «app/urls.py» и «project/urls.py». В них содержится переменная `urlpatterns`, которая является списком и содержит в себе функции Django `path(route, view, \*\*kwargs, name)` импортированную из модуля `django.urls`. Данная функция включает в себя следующие аргументы: route – название маршрута, view – функция обработчик маршрута, name – имя, используемое при ссылке на этот маршрут. Дополнительно используется функция `include`, импортированная из того же модуля, что и `path` и которая включает в себя набор `path`, является неким смысловым объединением для собранного блока.

Все обработчики маршрутных запросов собраны в едином модуле «views.py» и представлены в виде объектов функций и классов. Они используют в качестве аргумента переменную request, которая является объектом запроса, отправленным клиентом на сервер приложения. Она содержит множество атрибутов и методов, используемых для получения дополнительной информации, такие как: метод http запроса, словарь с параметрами переданные различными методами, словарь с загруженными файлами, объект пользователя, в случае авторизации, путь к запрашиваемой странице, словарь с дополнительными данными, объект сессии. Возвращаемым значением любого представления являются встроенные функции `render` и `redirect` импортируемые из модуля `django.shortcuts`. Функция `render` используется для создания HTTP-ответа на основе шаблона HTML и контекста переданных данных в шаблон в виде словаря. Аргументами функции являются переменная request, переданная в качестве аргумента той же функции представления, template\_name - название шаблона HTML, располагающегося в `template `папке, context – словарь передающий данные обработки в шаблон, ключи которого будут являться переменными шаблона. Функция `redirect` - обеспечивает переадресацию пользователя на другой URL маршрут, переданный в качестве аргумента функции.

Таким образом, исходя из технического задания к проекту, у нас сформировалась переменная `urlpatterns`:



Отсюда видно, что у нас появилось 4 блока маршрутизации, некоторые из которых являются объединением маршрутом. О каждом поподробнее.

**`/`**

Корневой маршрут под названием `index`. Обрабатывается функцией `views.index`. Возвращает страницу с приветствием пользователя.

**`/accounts/login` - login**

Обработчиком данного маршрута является класс `MyLoginView`. Класс наследуется от встренного в библиотеку Django класса `LoginView`, мы его переопределили, модифицировав только метод `form\_invalid` дополнением в виде flash-сообщения в случае неверно введенных данных.

Метод класса `get` возвращает встроенную форму для введения никнейма и пароля.

Метод `post` валидирует данные, осуществляет поиск на соответствие в базе данных и в успешном случае выполняет авторизацию пользователя и переадресовывает на страницу с личным профилем. В обратном случае отрабатывает запрос с GET методом и выдается причина отказа.

**`/accounts/personal` - profile**

Обработчиком данного маршрута является разработанная нами функция `profile`. За функцией закреплен встроенный в библиотеку Django декоратор `login\_required`, который выполняет проверку пользователя на статус авторизации и только в успешном случае дает функции выполнится, в обратном производит переадресацию по запросу get на маршрут `/accounts/login`. Тело функции содержит сбор персональных данных и контента, закрепленных за пользователем, а именно: созданные и понравившиеся плейлисты, добавленные треки, подписчики и подписки. Все это выполняется путем обращения по моделям таблиц БД к самой БД, с помощью ORM инструмента Django. Собранные данные становятся аргументов функции `render` вместе с request и `content/profile.html` шаблона HTML.

**`accounts/register` - registration**

Данный маршрут инициализирует экземпляр класса представления `RegistrationView`, наследуемый от встроенного в Django класса представления `View`. Целью данного обработчика является возможность регистрации на нашем веб-сервисе.

В классе определен один атрибут с названием шаблона HTML - `registration/register.html` и два метода get и post, аргументы которых наш экземпляр класса и request.

Метод `get` принимает GET запрос и предоставляет созданную нами форму регистрации в роли экземпляра класса RegistrationForm, который наследуется от UserCreationForm, наследуемый от ModeForm. Возможности Django при создании форм позволяют определить внутри класса формы еще один класс с закрепленным названием Meta, атрибутами которого является модель БД, относящаяся непосредственно к форме и дополнительные данные, в роли названия заполняемых строк в форме и т.д. Это обеспечивает интерактивную взаимосвязь модели и формы.

Метод `post` принимает запрос через HTTP метод POST. В этом случае вновь инициализируется экземпляр класса формы RegistrationFrom, куда в качестве аргументов передаются данные, отправленные этим методом. Данные проходят валидацию и в успешном случае выполняется следующий алгоритм. Создается переменная `user` во время объявления которой, происходит сохранение пользовательских данных в базу данных и идет присвоение данных созданной переменной. Следующим шагом происходит процесс аутентификации с помощью встроенной в библиотеку Django функции `login`, которой передается request и переменная user. Итогом этого процесса является переадресация на личную страничку пользователя. В случае, если данные формы не прошли валидацию, то возвращается тот же ответ, что и в методе `get`, но с переданными данными формы и текстом ошибки, в каком поле и где она была совершена.

**`content/create-playlist` - create-playlist**

Обработчиком данного маршрута является класс CreatePlaylistView, наследуемый от класса View. Данный класс включает в себя возможность создания плейлиста. Класс имеет декоратор method\_decorator (из модуля django.utils.decorators), принимающий аргументом функцию login\_required, который в случае не пройденной аутентификации пользователя переадресовывает на маршрут `accounts/login`.

Атрибутом класса является шаблон, хранимый в переменной `template\_name`. Методами являются функции `get` и `post`.

Метод `get` отвечает за GET метод HTTP запроса и принимает на вход стандартные аргументы self , request. В теле функции генерируется экземпляр класса формы CreatePlaylistForm и возвращает его в контексте шаблона. Класс формы привязан к модели Playlist и принимает аргументом объект пользователя, который в последствии метода `\_\_init\_\_` определяет атрибут модели Playlist.creator по умолчанию пользователя создателя плейлиста и список треков доступных пользователю, которые содержит его профиль.

Метод `post` отвечает за POST метод запроса HTTP и аналогично принимает на вход аргументы self и request. В теле метода происходит инициализация класса формы, что и в get методе и передаются аргументы в виде переменной request.POST, несущей в себе словарь данных и объект пользователя. Форма производит проверку на правильность заполненной формы и в положительном случае сохраняет плейлист в базу данных вместе с генерированием flash сообщения об успешном статусе, далее происходит переадресация на представление с наименованием `profile`. В неудачном случае генерируется сообщение о причине неудачи и возвращается исходный шаблон HTML в контексте которого сгенерированный экземпляр формы.

**`playlist/<int:pk>` - playlist**

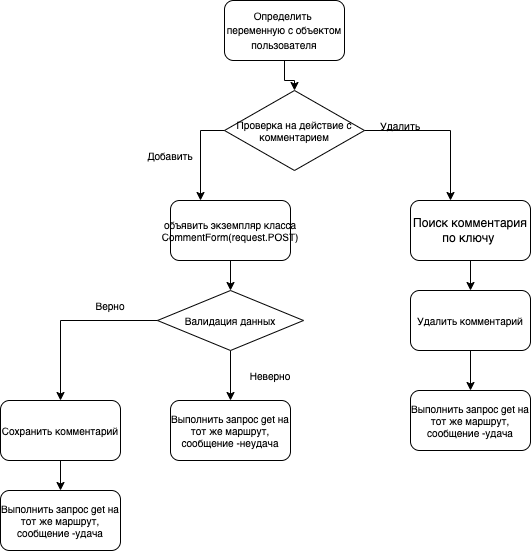
Данный маршрут является динамически образуемым, <int:pk> содержит в себе значение, которое является значением id необходимого плейлиста. Обработчиком данного маршрута является класс PlaylistView, наследумый от класса View.

Атрибутом класса является переменная с наименованием шаблона HTML. Методами класса является функции get и post.

Метод get предоставляет пользователю обзор на плейлист и предоставляет форму для комментариев. Метод принимает на вход три аргумента: self, request и pk, который передается из маршрута и форматируется в тип число. Тело функции содержит следующий алгоритм:

1. Объявляется переменная playlist, которой присваивается объект модели Playlists, соответствующий записи в БД с в таблице с плейлистами, у которого id равен аргументу pk.
2. Далее объявляется переменная tracks, которой присваиваются объекты модели Tracks, связанных с найденным выше объектом плейлиста по его атрибуту tracks, являющимся внешним ключом со связью многие-ко-многим.
3. Подобным образом создается переменная с объектами комментариев, которые привязаны к плейлисту связью многие-к-одному по внешнему ключу.
4. Создается экземпляр класса формы CommentsForm, привязанной к модели Comments.
5. Происходит проверка на создателя профиля. В случае если объект пользователя зарегистрирован и атрибут creator объекта плейлиста (найденного в шаге 1) равен объекту пользователи, то переменной owner присваивается флаг True, в обратном случае False.
6. Возвращается функция render, ответственная за HTTP ответ с шаблоном нашего представления, request и context в виде словаря с значениями переменных, объявленными в шагах (1-4).

Метод post ответственен за публикацию комментария к плейлисту, а так же его удаление. Метод обладает декоратором на проверку авторизации пользователя. Аргументами являются self, request, pk. Алгоритм метода:



**`user/<int:pk>` - user**

Маршрут обрабатывается представлением в виде класса ProfileView, наследуемого от класса View. Задача представления собрать весь контент связанный с пользователем (добавленные треки, созданные и добавленные плейлисты, подписки и подписчиков) и представить в виде ответа HTTP. А также возможность оформить подписку на пользователя.

Атрибутом класса является названием шаблона HTML, который используется в ответе метода get. Шаблон содержит вывод контента с возможностью взаимодействия, а кнопку «подписаться» посылающую сигнал методом POST на тот же маршрут.

Метод get принимает GET запрос HTTP вместе со следующими аргументами: self, request, pk, который является id объекта пользователя в БД. Тело функции метода содержит алгоритм:

1. С помощью блока try/except/finally выполняется поиск объекта пользователя по `pk` в базе данных. В блоке except перехватывается исключение CustomUser.DoesNotExist в случае, если пользователь с таким id отсутствует в базе данных. В таком случае выдается сообщение о неудачном поиске и возвращается `redirect` на страницу `index`.
2. В успешном случае блока следующий этап — это проверка на соответствие пользователя обладателя страницы, в случае положительного результата формируется сообщение и возвращается `redirect` на страницу `profile`, где у пользователя появляется возможность управления.
3. В обратном случае происходит сбор данных о контенте объекта пользователя (шаг 1), с помощью ORM Django.
4. Возвращается функция `render`, аргументами которой являются request, self.template\_name и контекст в виде словаря с ключами передающимися в шаблон, содержащие значения контента.

Метод `post` ответственен за возможность пользователя оформить подписку на объект пользователя просматриваемой страницы в `get` запросе. Метод доступен аутентифицированным пользователям, за счет наличия проверяющего декоратора `login\_required`.

Тело функции содержит алгоритм:

1. Присвоение переменной пользователя веб сервиса
2. Определение действия необходимого на выполнение, доступные варианты – отписаться/подписаться.
3. Поиск объекта пользователя, над которым необходимо совершить действие.
4. Проверка на соответствие пользователя, отправившего запрос, и пользователя, определенного в действии 3. В случае положительного результата, формируется сообщение о невозможности действия.
5. Далее еще одна проверка на наличие, или в случае «отписаться» на отсутствие характерной записи в базе данных. В случае положительного результата формируется соответствующее сообщение.
6. При удачном прохождении проверок осуществляется само действие в БД и формируется соответствующее сообщение
7. Возвращается `redirect` на тот же маршрут с методом `get`

**`search/search` - search**

Обработчиком маршрута является класс представления SearchView. Целью данного представления является поиск контента, соответствующему сообщению, переданному с помощью формы ввода на заголовке.  
Атрибутами класса являются:

1. HTML шаблон с названием «content/searching.html», настроенный на вывод результатов поиска. В случае отсутствия соответствия какого либо элемента, выдается соответствующее сообщение.
2. api\_track\_query – машрут GET запроса к API сервису Musixmatch – главному источнику музыкального контента. Содержит необходимые параметры необходимые для поиска на соответствие запросу, которым впоследствии дополняется
3. api\_artists\_query – подобный 2 атрибуту маршрут, работающий на поиск музыкальных исполнителей.

В классе определен один метод get, который принимает на вход 2 аргумента: self, request. Тело содержит алгоритм:

1. Определение сообщения переданного через форму заголовка
2. Проверка на пустое сообщение, если положительный результат, то возвращается `render` с пустой переменной context.
3. В обратном случае, с помощью библиотеки requests происходит HTTP запрос по маршрутам определенным в атрибутах 2 и 3 и дополненными данными 1 шага и глобальной переменной API ключа, полученного на сайте сервиса. Результаты присваиваются переменным.
4. На данном этапе используется разработанный вспомогательный класс APIDataMixins, с целью заполнения контента в локальную базу данных. Его последовательность действий состоит из 1. – обработка HTTP ответа, проверяется статус выполнения запроса, 2 – форматируются результаты данных, результат приходит в усложненной многоуровневой структуре, форматирование позволяет придать более лаконичную структура в виде словаря и возвращается список словарей, 3 – итерация элементов 2 этапа, с заполнением базы данных. На этом этапе проводится полный цикл действий класса миксина и возвращаются данные 2 этапа.
5. Поиск пользователей на соответствие сообщения 1 шага.
6. Возврат HTTP ответа в виде функции `render` со стандартными аргументами и контекстом, содержащим собранные данные в шаге 3-5.

**`search/top-artists` - top-artists / `search/top-tracks` - top-tracks**

Два маршрута, представлениями которых являются классы TopArtistsView и TopTracksView и выполняющих идентичный алгоритм. Целью классов предоставить пользователям актуальные сведения об музыкальных произведениях и исполнителях.

Атрибуты классов:

1. template\_name – названия шаблонов, присущие результатам запроса. Шаблон по результатам произведений содержит формы, отправляющие данные на маршрут `content-manager` с возможностью добавить трек в профиль зарегистрированному пользователю.
2. query – маршрут для get HTTP запроса к API сервису Musixmatch.
3. params – выбранные параметры поиска, фильтрующие запрос по различным критериям.

Классы содержат метод get, алгоритм которого:

1. GET HTTP запрос к API сервису с помощью библиотеки requests, uде используются атрибуты 2 и 3 с добавлением API ключа и присвоение результата переменным response.
2. Выполнение алгоритмов класса APIDataMixins, которому в аргументы передается response. Возвращаются данные, предварительно загруженные в базу данных.
3. Возвращается `render` с аргументами request, template\_name, context, которому переданные значения переменной, определенной во 2 шаге.

**`search/recommendations` - recommendations**

Обработчиком маршрута является класс представления RecommendationsView(View). Задача данного обработчика - представить пользователю рекомендованных пользователей. В нашем случае рекомендациями являются подписки пользователей, на которых сам пользователь подписан.

Атрибутом класса является переменная с названием шаблона HTML, который содержит верстку веб-страницы для представления контента.

Методом класса является функция `get`, принимающая аргументы: self, request.

Тело функции метода составляет алгоритм:

1. Объявляется тип переменной «список» recommendations, в которую в последствии будут добавлены пользователи.
2. Происходит проверка на статус авторизации пользователя. В случае положительного результата, происходит шаг 3, в отрицательном 4.
3. Переменная объявленная в 1 шаге переназначается в качестве вызова метода модели CustomUser.get\_users\_recommendations(), заменяя CustomUser на экземпляр модели пользователя. Метод модели CustomUser содержит функцию, которая выполняет следующий алгоритм:
   1. Объявляется пустая переменная с типом данных «список» recommendations
   2. Запрос в БД с целью нахождения подписчиков объекта модели и присваивание переменной followers.
   3. Проверка переменной followers на наличие результатов, в отрицательном случае переход на шаг 3.5.
   4. Итерация результатов поиска в переменной followers:
      1. Нахождение у элемента в виде объекта пользователя подписчиков с помощью метода класса у атрибута CustomUser.followers и присваивание переменной user\_following
      2. Итерация объектов user, входящих в user\_following:
         1. Объект user проходит проверку на соответствие пользователю экземпляра self и на вхождение в списки followings /recommendations, для предотвращения рекомендации пользователю самого себя и имеющихся его подписок.
         2. В случае положительного результата объявляется переход на следующий объект итерации user.
         3. В отрицательном исходе объект user добавляется в список recommendations.
   5. Возврат значения переменной recommendations.
4. Возврат функции `render`, в которой аргумеенты: request, self, context, с переданным значением переменной recommendations.

**`content-manager` - manager**

Данный маршрут является вспомогательным маршрутом, обработчик которого принимает `post` запросы с данными различных форм и занимается управлением контента пользователей, а именно добавлением/удалением плейлистов и музыкальных произведений. Обработчиком маршрута является функция, принимающая аргумент request и задекорированная login\_requiered.

Тело функции:

1. Присвоение значения `request.POST.get(‘content’) переменной content.
2. Определение контента, возможны варианты `playlists` и `track`. В первом случае обрабатывается шаг 2.1, во втором 2.2.
   1. Если контент – плейлист.
      1. Нахождение объекта плейлиста в БД с помощью id, пришедшего в форме запроса POST.
      2. Определение необходимого действия, удалить/добавить.
         1. В случае добавить:
            1. Проверка на имеющуюся подобною запись в БД, если запись имеется, пользователь добавлял плейлист ранее. Формируется сообщение о статусе.
            2. В обратном случае происходит запись в БД и формируется сообщение
         2. В случае удалить:
            1. Проверка на соответствие атрибута объекта плейлиста creator и объекта пользователя:

В случае если положительный результат, происходит удаление объекта плейлиста, формируется сообщение о статусе.

В обратном случае происходит проверка на имеющуюся подобную запись в БД:

В положительном случае происходит удаление объекта взаимосвязи объектов пользователя с плейлистом. Формируется сообщение о статусе.

В обратном формируется сообщение о недоступности выполнения.

* 1. Если контент – трек.
     1. Поиск объекта трека в БД с помощью id, пришедшего в запросе POST.
     2. Определение действия удалить/добавить пришедшего в POST форме.
        1. В случае добавить:
           1. Проверка на подобную запись в БД с объектом трека и объекта пользователя. Если подобная запись имеется, формируется сообщение о невозможности выполнения действия.
           2. В обратном случае создается запись и сохраняется в БД, формируется сообщение о статусе.
        2. В случае удалить:
           1. Проверка на подобную запись в БД. Если запись отсутствуют, формируется сообщение о невозможности выполнения действия.
           2. В обратном случае происходит удаление соответствующей записи и формируется сообщение о статусе.

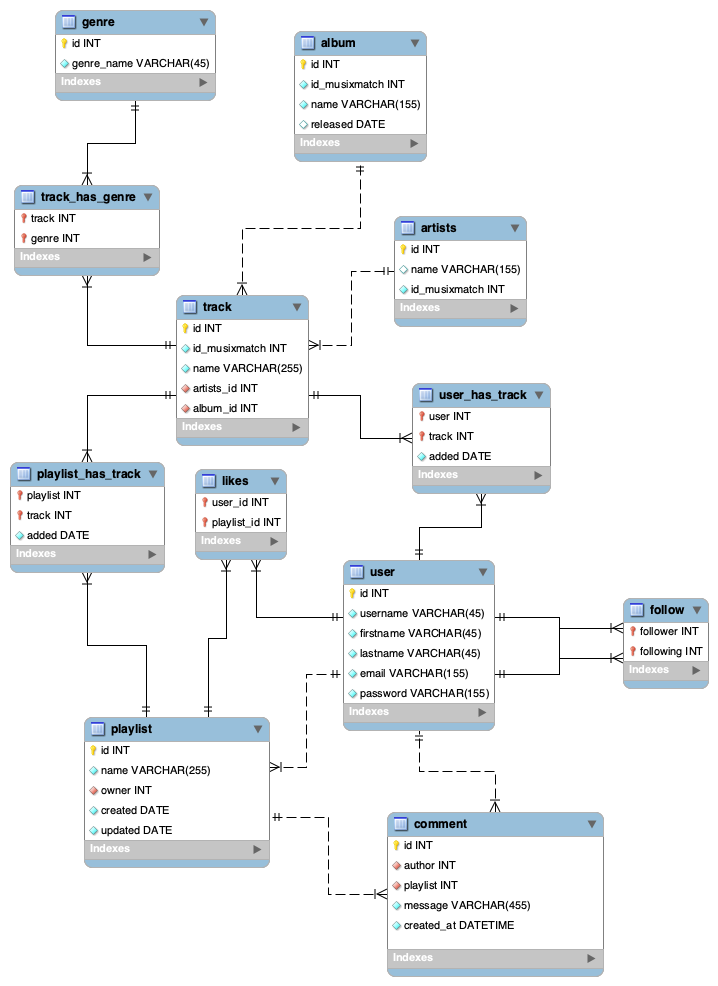
1. Далее определение значений ключей `next` и `query`.
2. Проверка на наличие результатов значений:
   1. В случае наличия обоих результатов возвращается `redirect` с аргументом `next + query`
   2. В случае наличия только `next` возвращается `redirect(next)`
   3. В случае отсутствия возвращается `redirect(‘profile’)`

На протяжении всей работы веб-сервиса происходит постоянное взаимодействие с моделями базы данных. Каждая модель представляет собою таблицу в БД с присущими ей атрибутами в виде колонок в таблице. ORM Django позволяет использовать встроенные методы объектов для распаковки данных. Вместе с этим в каждой модели определены методы, облегчающие разработку представлений. Подробнее о моделях в следующей графе.

# Структура данных

Разработанная структура данных содержит следующие сущности: пользователь, музыкальное произведение, исполнитель, жанр, альбом, плейлист, комментарий. Особенности, атрибуты и методы, взаимосвязи – все определены в модуле app/models.py в виде классов объектов и называются моделями. На основе прописанных моделей Django создает конфигурацию и выполняет загрузку моделей в БД с помощью команд для миграций: `python manage.py makemigrations` и `python manage.py migrate`.

Наглядным способом структура БД выглядит на рисунке ниже.



Отсюда видно следующие взаимосвязи:

Трек => Альбом: многие к одному

Трек => Жанр: многие ко многим

Трек => Исполнитель: многие к одному

Трек => Пользователь: многие ко многим

Трек => Плейлист: многие ко многим

Пользователь => Плейлист: один ко многим (создатель)

Пользователь => Плейлист: многие ко многим (отметки нравится)

Пользователь => Пользователь: многие ко многим (подписки)

Пользователь => Комментарий: один ко многим

Плейлист => Комментарий: один ко многим

Каждая модель сущности в виде класса включает в себя набор атрибутов и методов. Атрибуты каждой модели отражаются в рисунке, представленном выше. Модель CustomUser, которая характеризуется таблицей user, обладает набором методов, позволяющих выполнять следующие возможности сервиса, а именно: подписаться, отписаться, выполнять проверку на подписку и собирать рекомендации.

# Заключение

В результате проделанной работы было разработано приложение, отвечающее требованиям к основным функциональным возможностям. Получилось предоставить площадку, где пользователи могут осуществлять поиск любимых музыкальных произведений и взаимодействует друг с другом на основе своих предпочтений. Проект разрабатывался на локальной машине и следующим этапом стала сборка приложения в изолированную виртуальную среду с использованием программного обеспечения Docker с целью размещения его на удаленной машине и вывод в открытый доступ. На данном этапе возникли сложности, которые привели к ограничениям выполнения задач. Процесс завершился на этапе сборки контейнера, содержащего образец приложения и базы данных MySQL. Трудности возникли с в момент распределения портов работы между локальной машиной и виртуальной средой, а также совместимости работоспособности контейнера с виртуальной средой Linux на машине MacOS. Решением первой проблемы оказалось указания переменной HOST в конфигурационном файле `project/settings.py` в виде названия образца, в котором содержится БД, а именно ‘mysql`. Решением второй проблемы стало определение в виде опции интеграции платформ между собой `--platform=linux/amd64`.

Веб-сервис обладает огромный потенциалом для развития. Помимо завершения задачи по размещению сервиса в открытом доступе, развитие проекта возможно в виде разработки тестирования, модификации существующих представлений с целью оптимизации и усовершенствования, внедрение дополнительных возможностей, указанных в ТЗ проекта.

# Литература

1. Бизли Д., Джонс Б. Python. Книга рецептов. - пер. с анг. Б. В. Уварова. — М.: ДМК Пресс, 2019. – 648 с.
2. Щерба А. В. Программирование на Python: Первые шаги. — М.: Лаборатория знаний, 2022. – 250с.
3. Златопольский Д. М. Основы программирования на языке Python. — М.: ДМК Пресс, 2017. – 284 с.
4. Кольцов Д. М. Pyтноn. Полное руководство. — СПб.: Издательство наука и техника, 2022. – 480с.
5. Мухин Н. А., Волкова Я. П. MySQL руководство администратора. 2005. – 621 c
6. Кузнецов М.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. – 581 с
7. Шаббир Чаллавала и др. MySQL 8 для больших данных/ Шаббир Чаллавала, Джадил Лакхатария, Чинтан Мехта, Кандарп Патель. — М.: ДМК Пресс, 2018. – 226с.
8. Тахагоки С., Вильямс Хью Е. Руководство по MySQL. Пер. с англ. — М.: Русская редакция, 2007. – 544 с.
9. Головатый А. Django. Подробное руководство. — М.: Символ-плюс, 2014. – **512** c.
10. Форсье, Дж. Django. Разработка веб-приложений на Python — М.: Символ-плюс, 2014. – **343** c.
11. Зыков, С. В. Программирование. Объектно-ориентированный подход/ учебник и практикум для академического бакалавриата. — М.: Юрайт, 2019. – 155 с.
12. Владимир Дронов Django 3.0. Практика создания веб-сайтов на Python — СПб.: BHV, 2021. – 704 c.
13. Иан Милл, Эйдан Хобсон Сейерс Docker на практике. — М.: ДМК Пресс, 2020. – 516 с.