Urban University

Дипломная работа по курсу "Python-разработчик"

Тема: Анализ и сравнение написания web-приложений с использованием разных фреймворков

2024 год

# Содержание

[1. Введение 3](#_Toc185844887)

[2. Основные понятия и определения 3](#_Toc185844888)

[3. Обзор фреймворков 4](#_Toc185844889)

[4. Обзор проекта 5](#_Toc185844890)

[5. Структура проекта 5](#_Toc185844891)

[5.1 Домашняя страница 6](#_Toc185844892)

[5.2 Страница с выбором предсказания 6](#_Toc185844893)

[5.3 Страница с выбранным предсказанием 7](#_Toc185844894)

[5.4 Страница с перечнем всех предсказаний 7](#_Toc185844895)

[5.5 Шаблоны страниц 7](#_Toc185844896)

[6. Написание веб-приложения на разных фреймворках 8](#_Toc185844897)

[6.1 Написание веб-приложения на фреймворке Django 8](#_Toc185844898)

[6.2 Написание веб-приложения на фреймворке FastAPI 13](#_Toc185844899)

[6.3 Написание веб-приложения на фреймворке Flask 17](#_Toc185844900)

[7. Сравнение фреймворков Django, Flask и FastAPI 19](#_Toc185844901)

[8. Заключение 22](#_Toc185844902)

[Приложение 1. Пример файловой структуры проекта 23](#_Toc185844903)

[Приложение 2. Список необходимых библиотек 23](#_Toc185844904)

# Введение

Быстрый рост популярности веб-сайтов и приложений в сети Интернет связан с современным уровнем развития информационных технологий. Современные веб-приложения обладают огромными функциональными возможностями, позволяя автоматизировать и усовершенствовать множество выполняемых процессов.

Учитывая достаточно большое количество фреймворков, разработчикам становится достаточно сложно выбрать один из них для разработки конкретного приложения. В дипломной работе проводится анализ и сравнение разработки веб-приложений с использованием трех популярных фреймворков Python: Django, Flask и FastAPI. Работа включает в себя создание простых веб-приложений на каждом из фреймворков, а также оценку их производительности, удобства разработки и доступной документации. Исследуется, как каждый из этих фреймворков справляется с различными задачами, а также рассматриваются реальные примеры использования на практике. В итоге работу завершает обобщение полученных результатов и рекомендации по выбору подходящего инструмента в зависимости от специфики проектов.

*Цель*

Обоснование выбора фреймворка для разработки веб-приложений на Python на основе их сравнительного анализа.

*Задачи*

1. Изучить основные характеристики фреймворков Django, Flask и FastAPI;

2. Разработать аналогичные простые веб-приложения на каждом из фреймворков;

3. Провести сравнение по критериям производительности и удобства разработки.

# Основные понятия и определения

Обзор основных понятий в области написания web-приложений с использованием разных фреймворков:

1. *Фреймворк (Framework):* Программная платформа, которая предоставляет готовые компоненты и инструменты для разработки приложений. В дипломной работе используются Django, Flask и FastAPI.

2. *Веб-приложение (Web Application):* Программное приложение, которое работает на веб-сервере и доступно через браузер. В контексте данного проекта, веб-приложение будет использоваться для сравнения и демонстрации работы фреймворков Django, Flask и FastAPI.

3. *Бэкенд (backend)* – серверная часть веб-приложения, обрабатывающая запросы от клиента, выполняющая логику приложения и взаимодействующая с базой данных.

4. *Фронтенд (frontend)* – клиентская часть веб-приложения, взаимодействующая непосредственно с пользователем, отвечающая за отображение данных и предоставления пользователю интерфейса взаимодействия с приложением.

5. *ORM (Object-Relational Mapping, объектно-реляционное отображение)* – технология взаимодействия с базой данных через объектно-ориентированный интерфейс, скрывая детали SQL-запросов.

*6. Шаблонизатор* – программное обеспечение, позволяющее использовать шаблоны для генерации конечных документов с помощью декларативного языка разметки.

7. *Web-сокет* – протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и web-сервером, используя постоянное соединение.

8. *Диспетчеризация* – это процесс поиска и вызова необходимой функции или метода для уже известного типа данных.

9. *API (application programming interface) –* программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими (в противоположность пользовательскому интерфейсу, используемому для взаимодействия конечного пользователя с программой). Проще говоря, это способ взаимодействия какого-то программного кода с набором каких-то программных компонентов, с помощью которых одна компьютерная программа (например, бот или сайт) может использовать другую программу.

Данные термины помогают понять ключевые аспекты в области написания web-приложений с использованием разных фреймворков.

# Обзор фреймворков

*Python* – это интерпретируемый язык программирования, который становится все более популярным для создания веб-приложений. Тем не менее, существует множество веб-фреймворков на выбор, но все они имеют разные варианты использования.

*Django* следует архитектурному шаблону model-template-view и принципу DRY (не повторяйтесь).

*Flask* основан на Werkzeug WSGI toolkit и движке шаблонов Jinja2.

*FastAPI* это микро-веб-фреймворк, основанный на Pydantic.

В дипломной работе приводится сравнение этих фреймворков на основе различных факторов, таких как эффективность, удобочитаемость, варианты использования и т.д.

*Django* – это веб-фреймворк с полным стеком с открытым исходным кодом, написанный на Python, который следует архитектурному шаблону MTV (модель-шаблон-представление) и принципу DRY (не повторяйтесь). Оно было сделано Адрианом Головати и Саймоном Уиллисоном и было публично опубликовано в 2005 году. Данное название основано на имени певца Джанго Рейнхардта.

*Flask* – это удобный для начинающих веб-фреймворк с открытым исходным кодом, написанный на Python, который поддерживает шаблоны Jinja, которые можно использовать для создания представлений. Оно было сделано Армином Ронахером из Pocoo, международной группы энтузиастов Python, созданной в 2004 году. Название "Flask" является отсылкой к более раннему фреймворку Bottle.

*FastAPI* также является микро-веб-фреймворком Starlette-ASGI (Asynchronous Server Gateway Interface) с открытым исходным кодом, написанным на Python и используемым для создания RESTful API. Оно было сделано Себастьяном Рамиресом в декабре 2018 года. Название основано на скорости разработки API, отсюда и "FastAPI".

# Обзор проекта

В связи с наступающим Новым 2025 Годом хочется создать праздничное настроение, для этого выполнено веб-приложение в новогодней тематике.

Главной задачей веб-приложения является выбор предсказания на 2025 год. Для реализации данной задачи потребуется модуль *random* для генерации псевдослучайных чисел, которые определяют порядковый номер в перечне пожеланий. Предварительно в базу данных добавляется таблица с пожелания, включающая в себя две позиции – заголовок (title) и описание (content).

# Структура проекта

Проект включает в себя следующие ключевые компоненты: домашняя страница, страница с выбором предсказания, страница с выбранным предсказанием и страница с перечнем всех предсказаний.

## 5.1 Домашняя страница

На домашней странице в верхней части отображается меню со ссылками на все страницы сайта. В нижней части приведена информация «Вы находитесь на главной странице» (рис. 5.1).



Рис. 5.1 Главная страница

## 5.2 Страница с выбором предсказания

При нажатии на ссылку «Узнать предсказание на 2025 года» открывается страница с выбором предсказания. В нижней части данной страницы есть кнопка «Узнать предсказание». На данной странице также в верхней части отображается меню со ссылками на все страницы сайта (рис. 5.2).



Рис. 5.2 Страница с выбором предсказания

## 5.3 Страница с выбранным предсказанием

При нажатии на кнопку «Узнать предсказание» в базе данных рандомно выбирается предсказание, которое открывается на новой странице с выбранным предсказанием – в нижней части выводятся заголовок (title) и описание (content). На данной странице также в верхней части отображается меню со ссылками на все страницы сайта (рис. 5.3).



Рис. 5.3 Страница с выбранным предсказанием

## 5.4 Страница с перечнем всех предсказаний

В разработанном веб-приложении также можно ознакомиться с полным перечнем всех предсказаний, которые сохранены в базу данных. При нажатии на ссылку «Просмотр всех предсказаний» открывается новая страница, в нижней части которой выводятся заголовок (title) и описание (content) всех предсказаний. На данной странице также в верхней части отображается меню со ссылками на все страницы сайта (рис. 5.4).

## 5.5 Шаблоны страниц

*menu.html* - базовый шаблон, на его основании пишутся остальные. Данный шаблон используется только в фреймворке Django.

*main\_page.html* – главная страница.

*predictions.html* – страница с выбором предсказания.

*prediction.html* – страница с выбранным предсказанием. Данный шаблон принимает предсказание с ключем «Prediction» и выводит его заголовок (title) и описание (content).

*viewing.html* – страница с перечнем всех предсказаний. Данный шаблон принимает предсказания с ключем «Predictions», перебирает их с помощью цикла for и выводит заголовок (title) и описание (content) каждого предсказания.



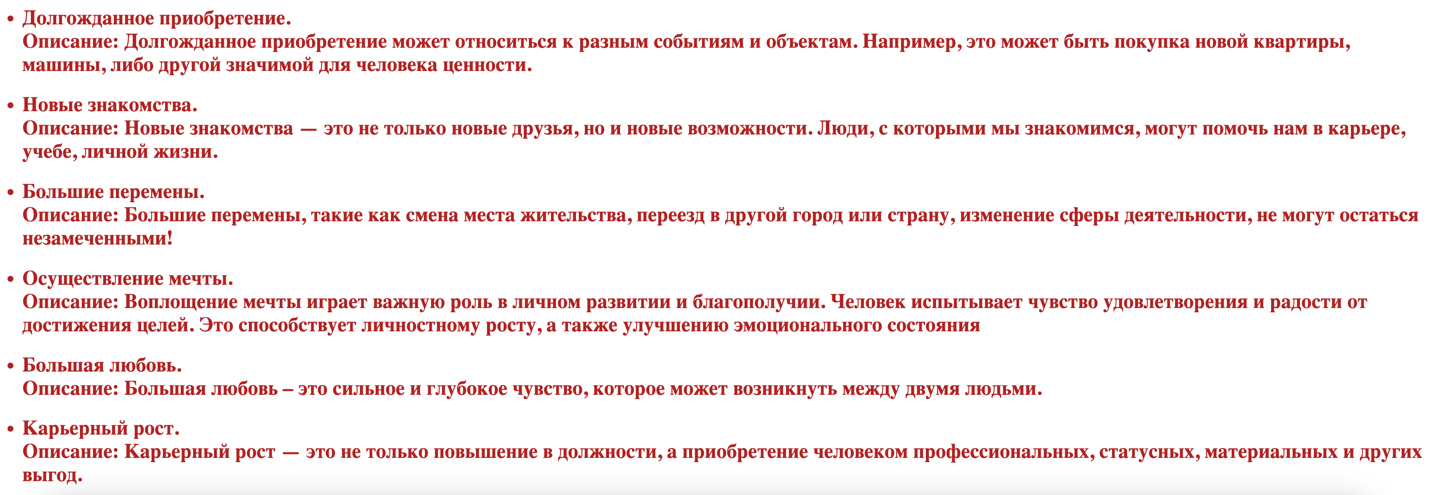


Рис. 5.4 Страница с перечнем всех предсказаний

# Написание веб-приложения на разных фреймворках

Ниже приводится написание веб-приложения на фреймворках Python: Django, Flask и FastAPI.

## Написание веб-приложения на фреймворке Django

Сначала нужно установить Django, для этого в консоли PyCharm нужно прописать команду «pip install django». Далее нужно создать наш проект, в консоли PyCharm прописываем команду «django-admin startproject NewYear» (где NewYear – наименование проекта) – появляется папка с несколькими файлами (рис. 6.1).

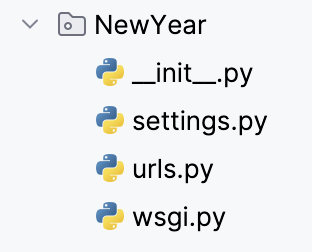


Рис. 6.1 Директория проекта NewYear

После этого необходимо создать приложение в проекте с помощью команды «python manage.py startapp predictions» (где predictions – наименование приложения). После выполнения команды будет создана директория с приложением predictions, в которой находятся инструменты для настройки отображения страниц и взаимодействия с базой данных. (рис. 6.2).

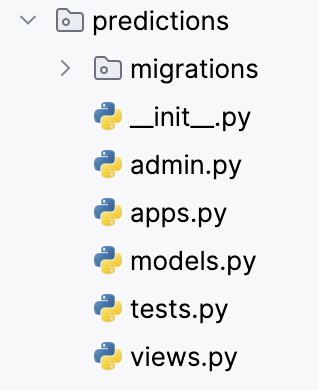


Рис. 6.2 Директория приложения predictions

Далее к проекту необходимо подключить и само приложение в файле «settings.py», иначе при запуске сервера будет ошибка (рис. 6.3).



Рис. 6.3 Подключение приложения predictions

Также в файле «settings.py» для взаимодействия с отображениями страниц необходимо спуститься в TEMPLATES и в параметре «DIRS» прописать нашу папку с шаблонами: [BASE\_DIR / 'templates'] (рис. 6.4).



Рис. 6.4 Указание директории с шаблонами

Чтобы настроить отображение страниц на сайте, необходимо использовать файл «views.py», расположенный в директории приложения predictions. В нем пишутся функции, в которых настраивается логика взаимодействия с сервером (рис. 6.5).



Рис. 6.5 Функции в файле «views.py»

Функция «Main\_page» отображает главную страницу.

Функция «Predictions» отображает страницу с выбором предсказания.

Функция «Viewing» отображает страницу с перечнем всех предсказаний. Для этого прописан дополнительный context в виде словаря, в котором выбираются все объекты с базы данных (Predictions.objects.all()) и передаются в шаблон «viewing.html» в виде ключа «Predictions».

Функция «Prediction\_random» отображает страницу с рандомно выбранным предсказанием. Для этого прописан дополнительный context в виде словаря, в котором выбираются все объекты с базы данных predictions=(Predictions.objects.all()), рандомно выбирается один объект с помощью модуля *random* (predictions[random.randint(0,len(predictions)-1)]) и передается в шаблон «prediction.html» в виде ключа «Prediction».

Для создания моделей необходимо пользоваться файлом «models.py», расположенный в директории приложения predictions. В данном файле прописывается класс по создаванию таблицы «Prediction» со столбцами заголовок (title) и описание (content) (рис. 6.6).

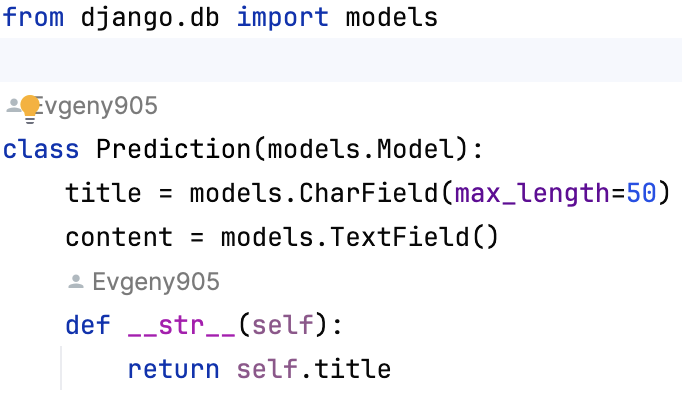


Рис. 6.6 Создание модели в файле «models.py»

Чтобы настроить взаимодействие с базой данных, для начала нужно выполнить команды «python manage.py makemigrations» для проверки несохранённых связей таблиц и «python manage.py migrate» для сохранения этих связей. Данные команды необходимо использовать каждый раз при добавлении новых моделей (таблиц) в приложение.

После этого мы создаем суперпользователя (для админки), выполнив команду «python manage.py createsuperuser» и в файле admin.py регистрируем модель, чтобы потом с помощью администратора добавлять нужные нам предсказания, где будут указаны их заголовки (title) и описания (content) (рис. 6.7).

Далее переходим в файл «urls.py», расположенный в директории проекта NewYear, и прописываем в нем все пути, которые мы хотим сделать с помощью path (рис. 6.8).

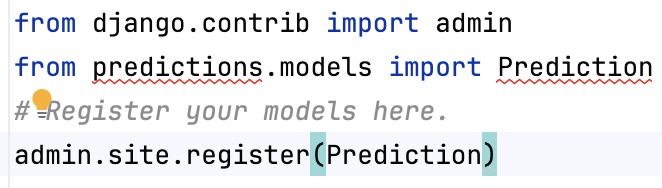


Рис. 6.7 Регистрация модели в файле «admin.py»

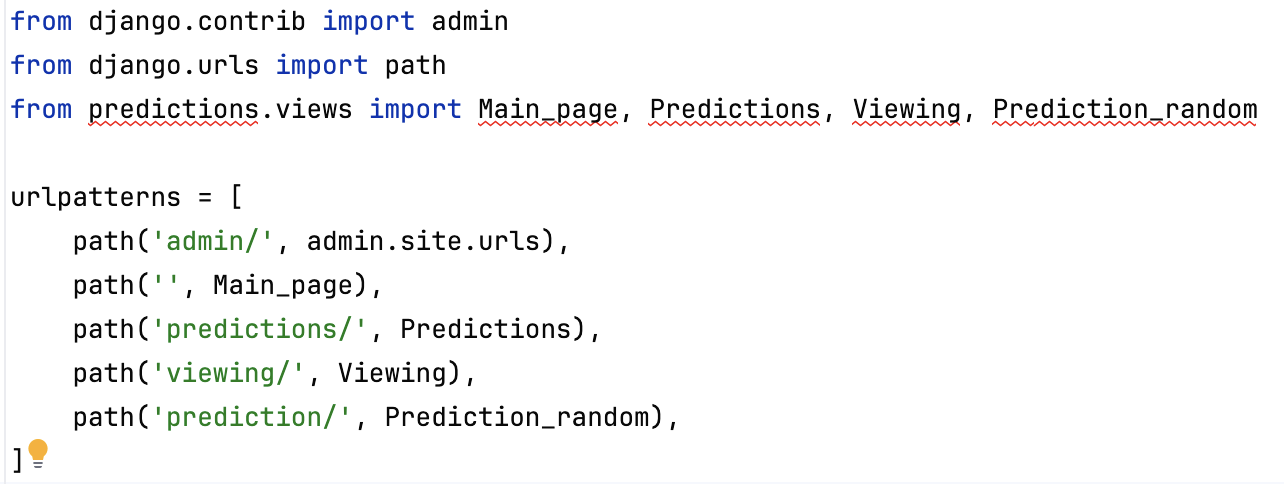


Рис. 6.8 Указание всех путей в файле «urls.py»

Чтобы запустить проект осталось прописать в консоли команду «python manage.py runserver».

Переходим по ссылке «http://127.0.0.1:8000» на главную страницу нашего сайта. Для добавления в базу данных предсказаний переходим по адресу «http://127.0.0.1:8000/admin/» и в разделе «Predictions» нажимаем кнопку "Add" и заполняем нужные нам параметры (рис. 6.9).

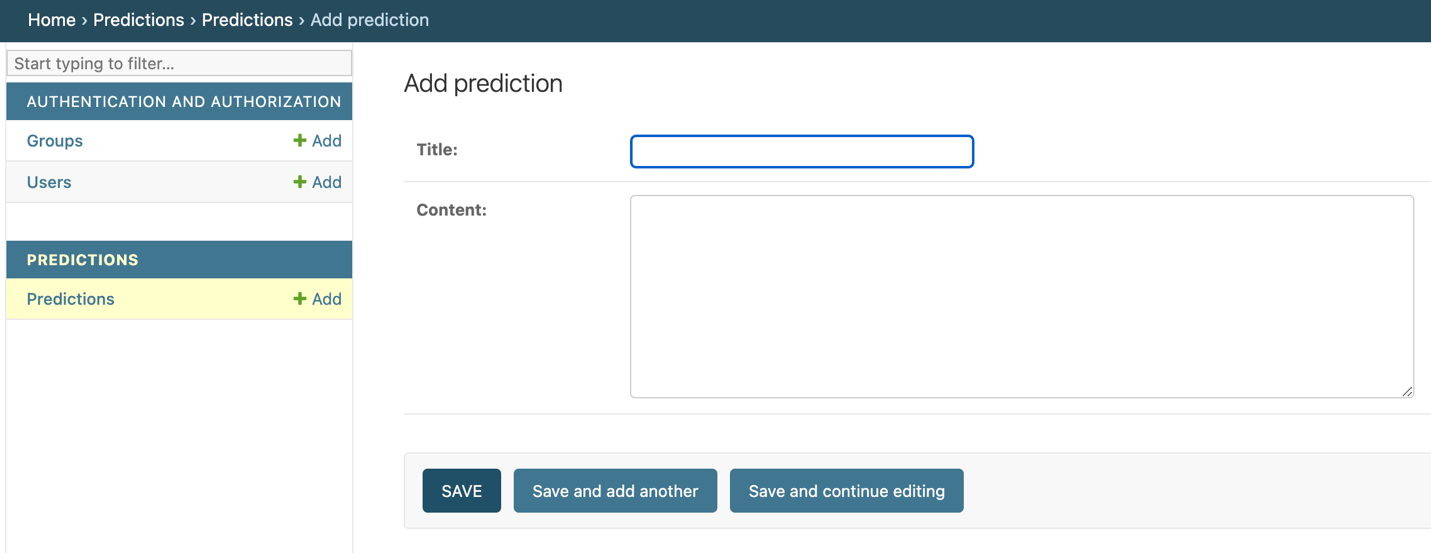


Рис. 6.9 Добавление объектов в базу данных на странице администратора

## Написание веб-приложения на фреймворке FastAPI

Сначала нужно установить FastAPI, для этого в консоли PyCharm нужно прописать команду «pip install fastapi».

Дополнительно устанавливаем библиотеки Pydantic (для создания класса базовых моделей) и SQLAlchemy (для работы с базой данных): «pip install pydantic», «pip install sqlallchemy».

Для взаимодействия с отображениями страниц в директории «templates» с шаблонами создаем новые одноименные шаблоны, ранее используемые в Django, но уже без наследования от базового шаблона menu.html (рис. 6.10). Данные манипуляции связаны с тем, что наследование шаблонов поддерживается только в Django (что безусловно является его преимуществом), для фреймворков Flask и FastAPI каждый шаблон создается независимо друг от друга (наследование не поддерживается).

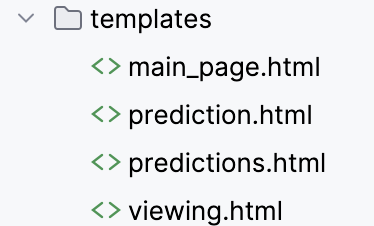


Рис. 6.10 Создание шаблонов в директории «templates»

Для работы с фреймфорком FastAPI нам понадобится импортировать библиотеки fastapi, typing, pydantic, sqlalchemy и random (рис. 6.11).

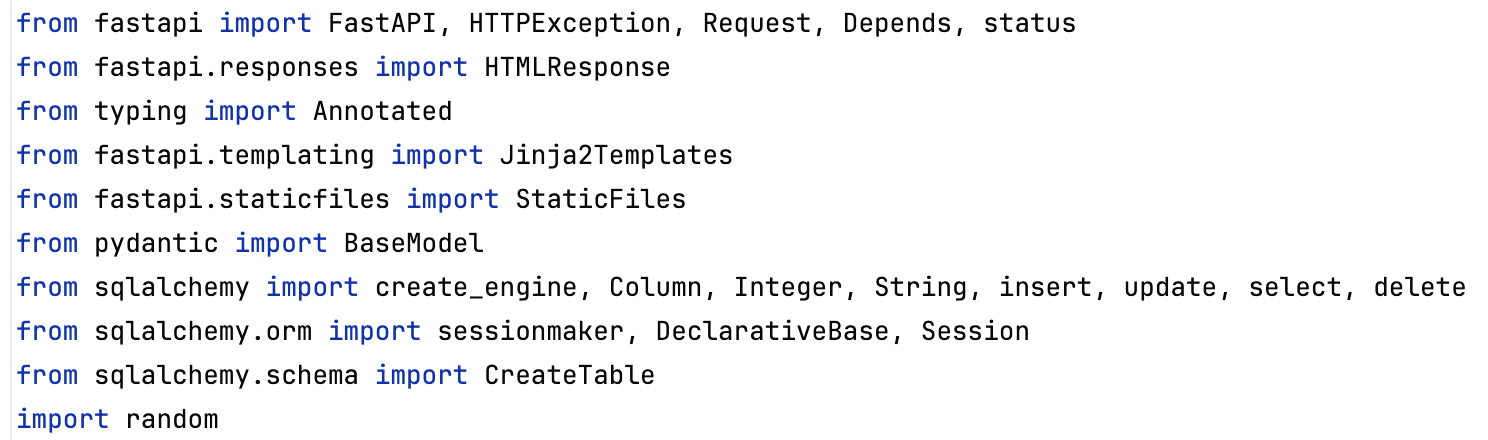


Рис. 6.11 Импорт библиотек для работы с «FastAPI»

Далее для реализации проекта в файле «main.py» создаем наше приложение «app», подключаем статические файлы с указанием директории «static», а также роутер (рис 6.12). В папке backend подключаем новую базу данных «Predictions\_base.db» (рис. 6.13). Стоит отметить, что ранее созданная база данных «db.sqlite3» в Django фреймворками FastAPI и Flask не поддерживается.

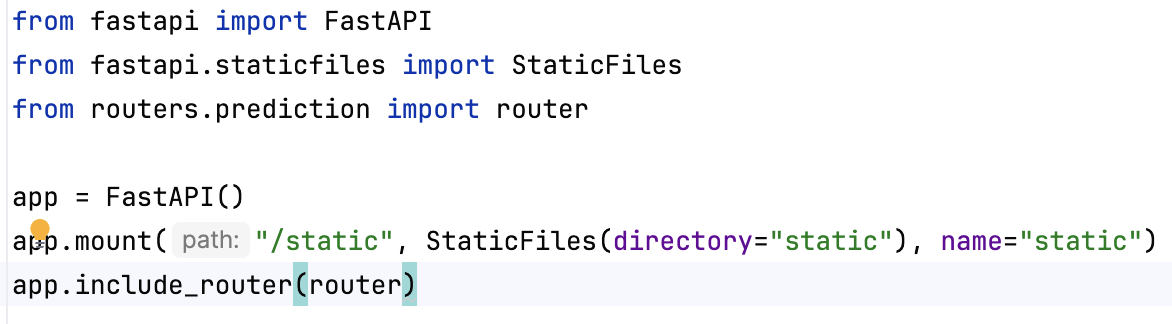
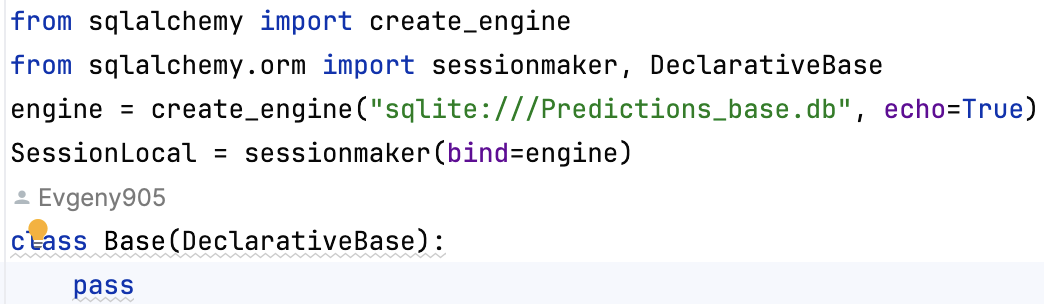


Рис. 6.12 Создание приложения «app», подключение к статическим файлам и роутеру



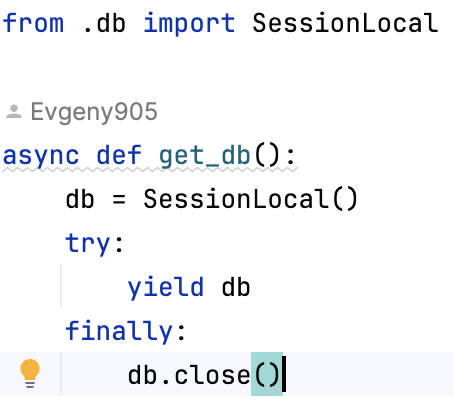


Рис. 6.13 Подключение к базе данных

После этого в пакете «models» создаём описание структуры базы данных для работы с ней (так называемая декларация схемы или таблица) (рис. 6.14).

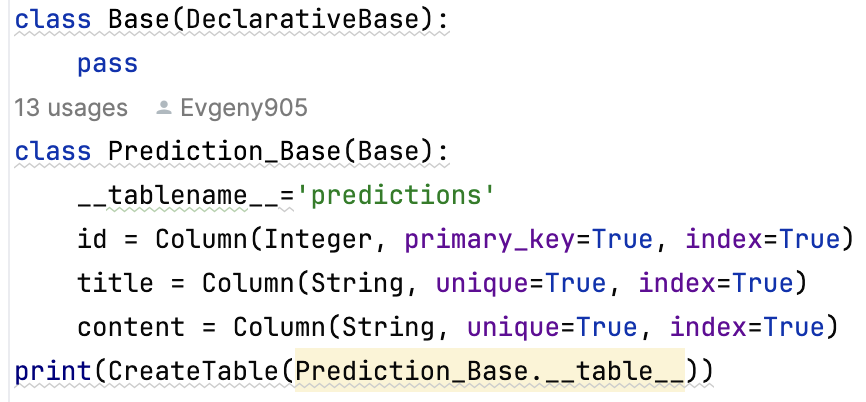


Рис. 6.14 Создание таблицы базы данных

В файле «schemas.py» с помощью библиотеки «Pydantic» создаем модели данных, которые проверяют входные данные на соответствие заданным типам и структуре (рис. 6.15).

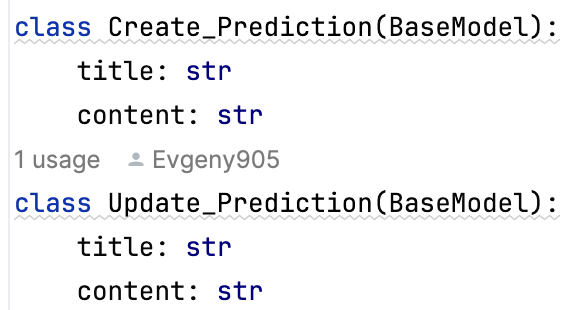
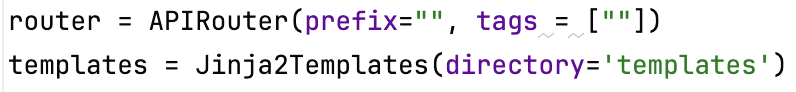
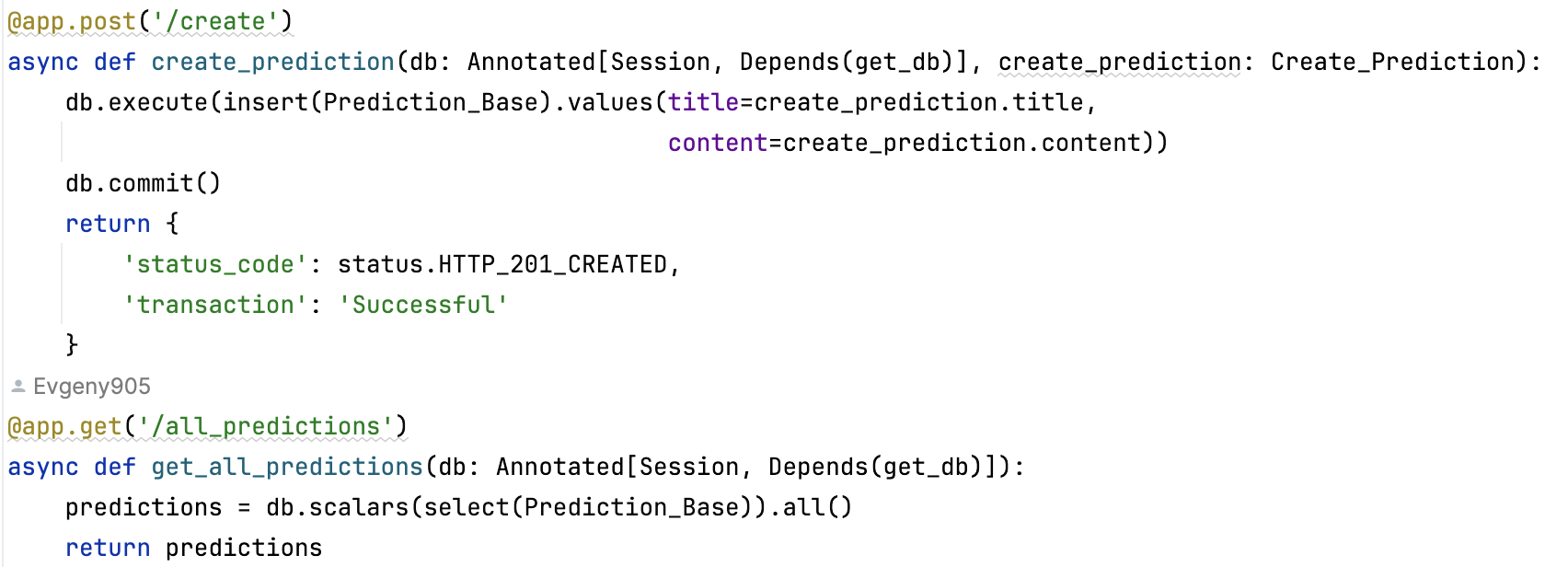
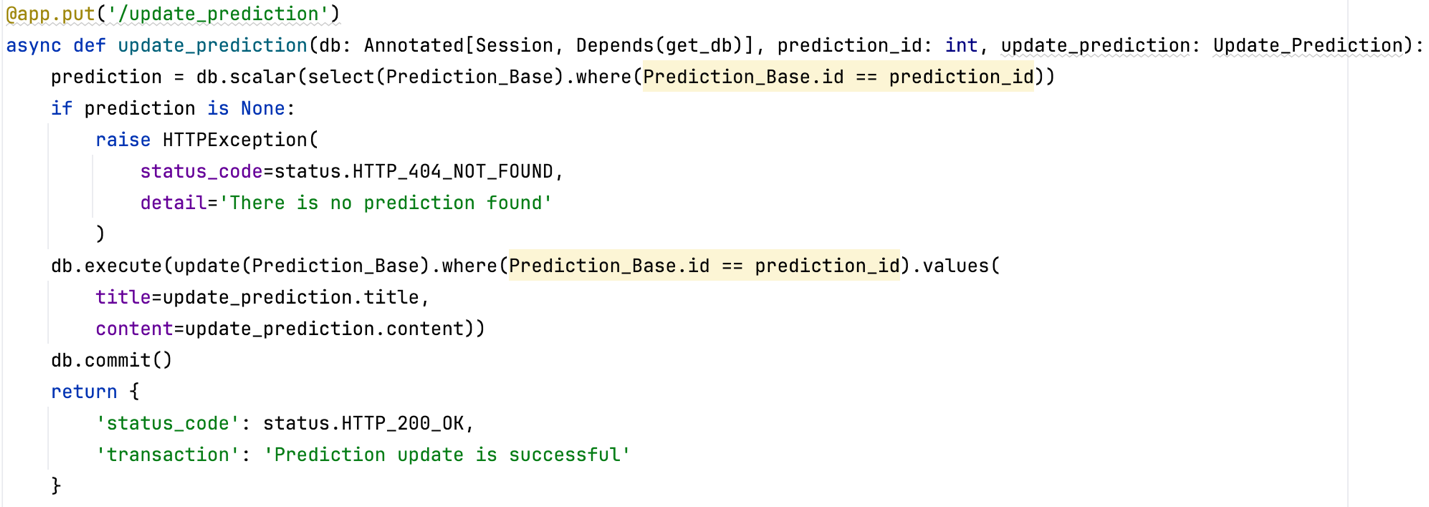


Рис. 6.15 Создание моделей данных с помощью библиотеки "Pydantic"

Далее в пакете «routers» создаем роутер и прописываем шаблоны для работы с базой данных с использованием запросов POST (создание объекта), GET (просмотр объектов), PUT (изменение объектов) и DELETE (удаление объектов). При этом не забываем подключить наши шаблоны с указанием директории «templates» с помощью Jinja2Templates (рис. 6.16).







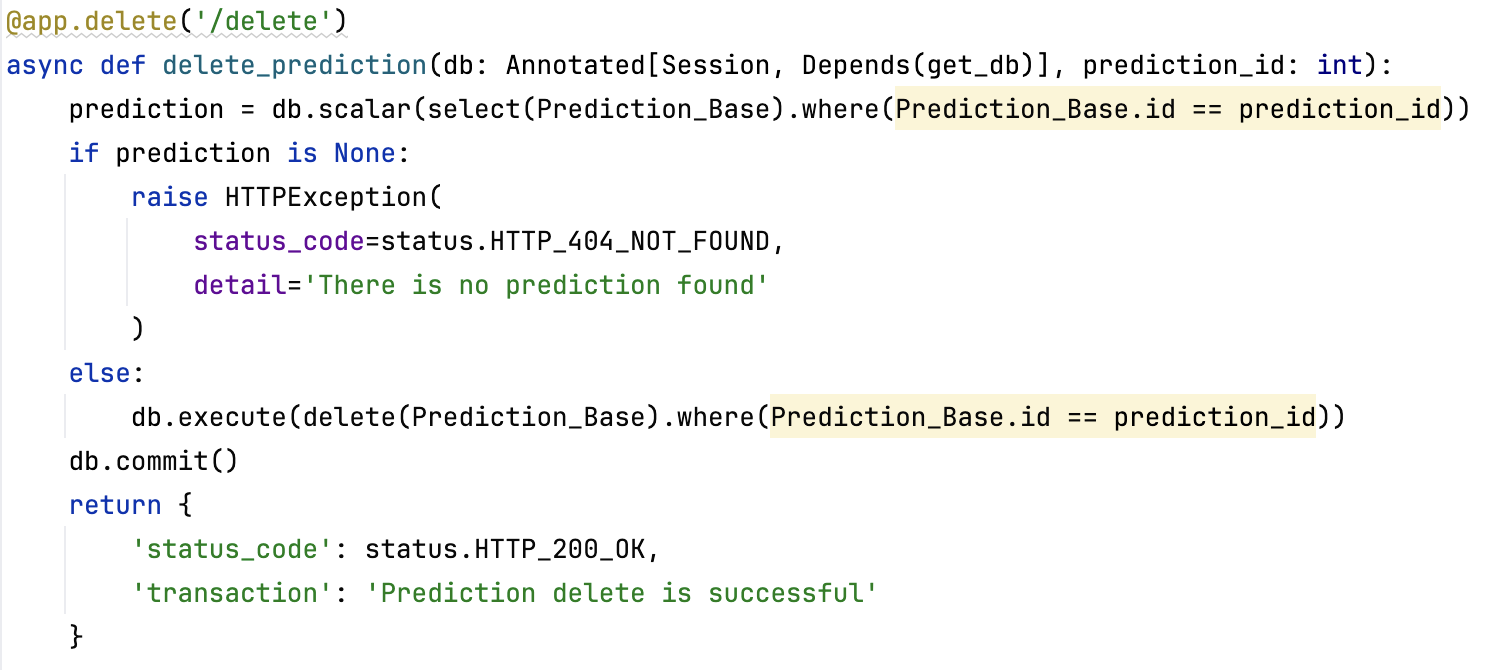


Рис. 6.16 Создание шаблонов для работы с базой данных

Также прописываем шаблоны для настройки проекта и отображения их на сайте (рис. 6.17).

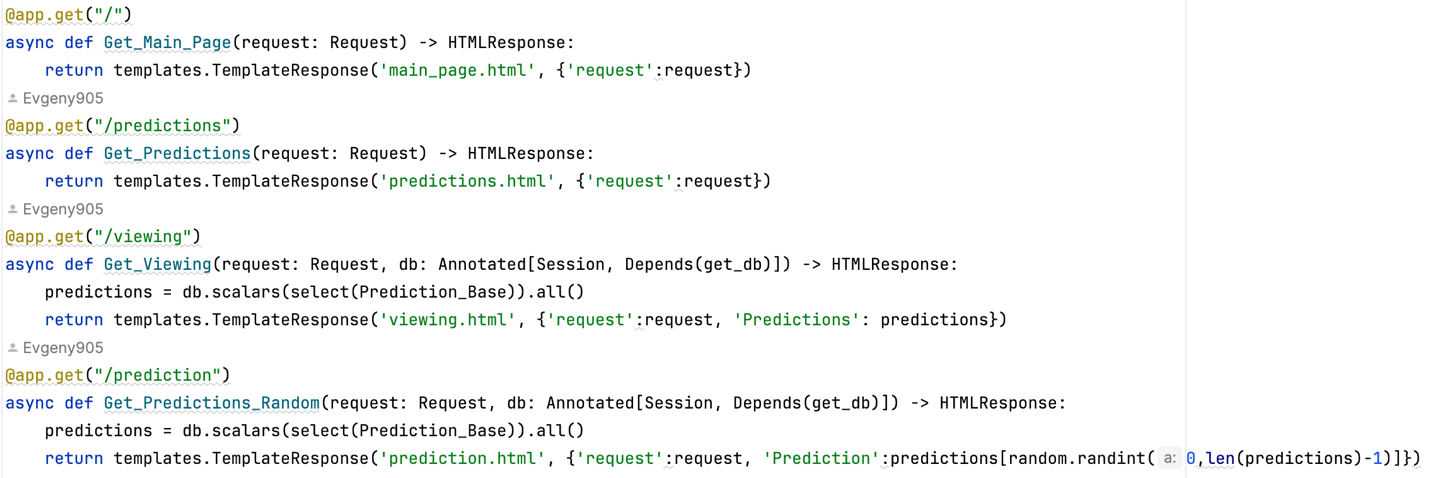


Рис. 6.17 Создание шаблонов для отображения их на сайте

Для работы с базой данных необходимо дополнительно установить библиотеку alembic – прописываем в консоли команду «pip install alembic». В появившемся файле настроек «alembic.ini» указываем нашу базу данных: sqlalchemy.url = sqlite:///Predictions\_base.db. После этого выполняем миграцию, прописав в консоли команды «alembic upgrade head» и «alembic revision --autogenerate -m "Initial migration"».

Чтобы запустить проект осталось прописать в консоли команду «python -m uvicorn main:app» (где main – наименование нашего модуля с приложением app).

Переходим по ссылке «http://127.0.0.1:8000/docs#/» на интерактивную документацию API (Swagger) (6.18). Для добавления в базу данных предсказаний в разделе «POST» заполняем нужные нам параметры и нажимаем кнопку "Execute" (рис. 6.19).

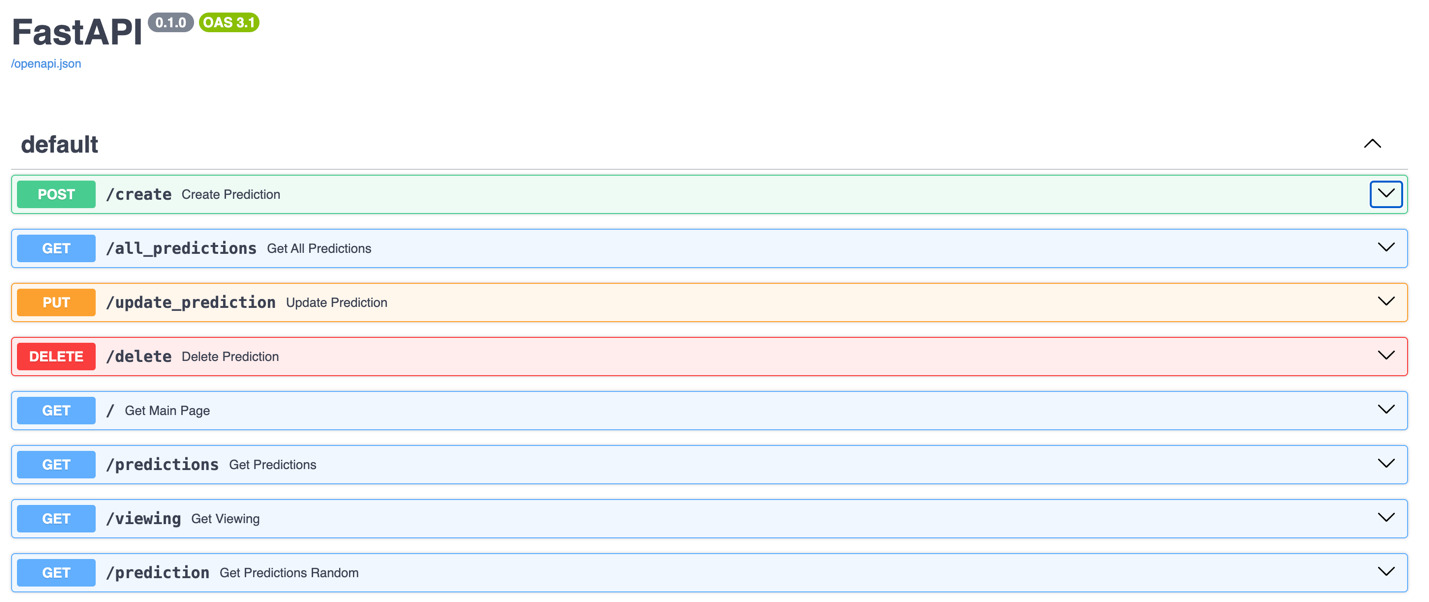


Рис. 6.18 Интерактивная документация API (Swagger)

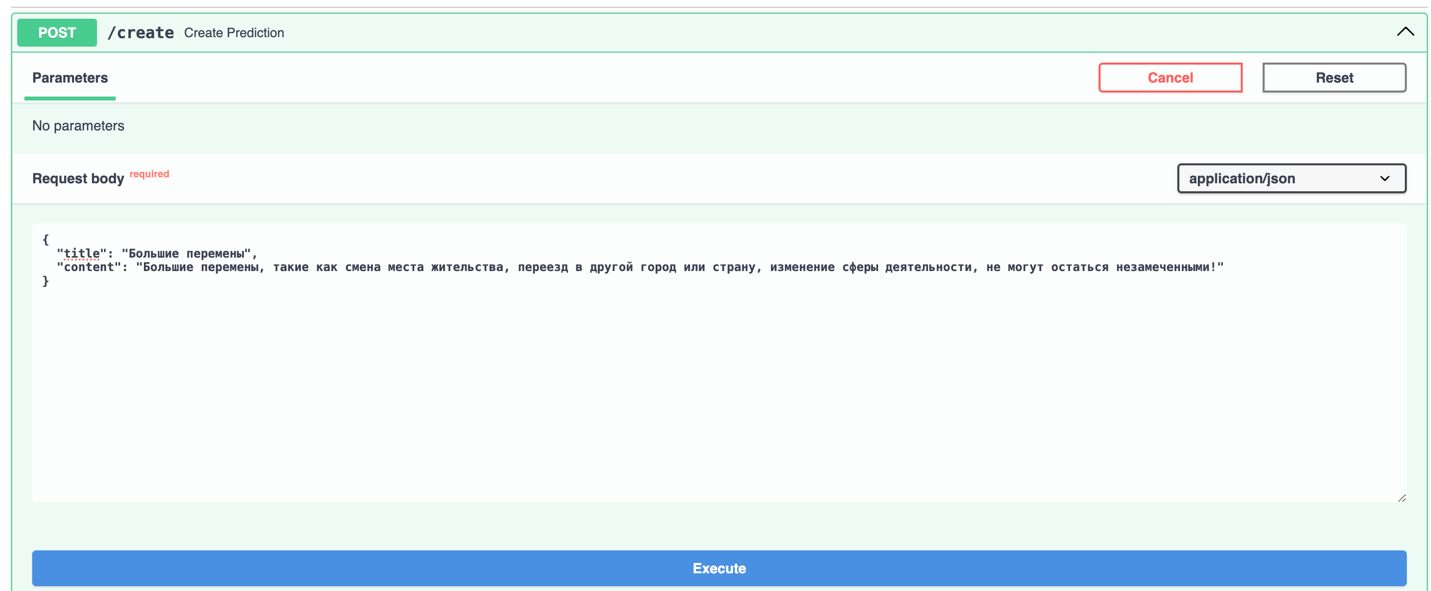


Рис. 6.19 Добавление предсказаний в базу данных

## Написание веб-приложения на фреймворке Flask

Сначала нужно установить Flask, для этого в консоли PyCharm нужно прописать команду «pip install flask».

Далее для реализации проекта создаем пакет «Flask», в котором выстраиваем структуру аналогично ранее созданному пакету «FastAPI». Импортируем необходимые библиотеки (рис. 6.20), создаем наше приложение «app», загружаем конфигурацию по умолчанию и переопределяем в конфигурации часть значений через переменную окружения (рис. 6.21). При этом указываем базу данных «Predictions\_base.db», ранее созданную в фреймворке FastAPI.

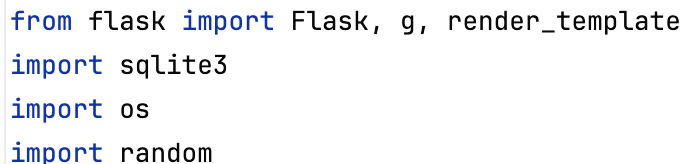


Рис. 6.20 Импорт библиотек для работы в фреймворке Flask

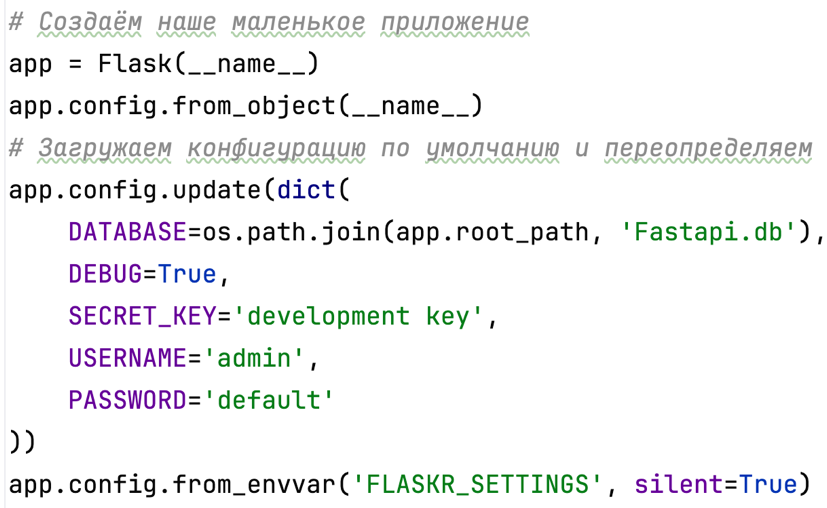


Рис. 6.21 Создание приложения «app» и переопределение значений в конфигурации

Для соединения с базой данных прописываем функцию «connect\_db». Для проверки соединения с базой данных создаем функцию «get\_db» (рис. 6.22).

Рис. 6.22 Соединение с базой данных

Чтобы настроить отображение страниц на сайте, прописываем функции, в которых настраивается логика взаимодействия с сервером (рис. 6.23).



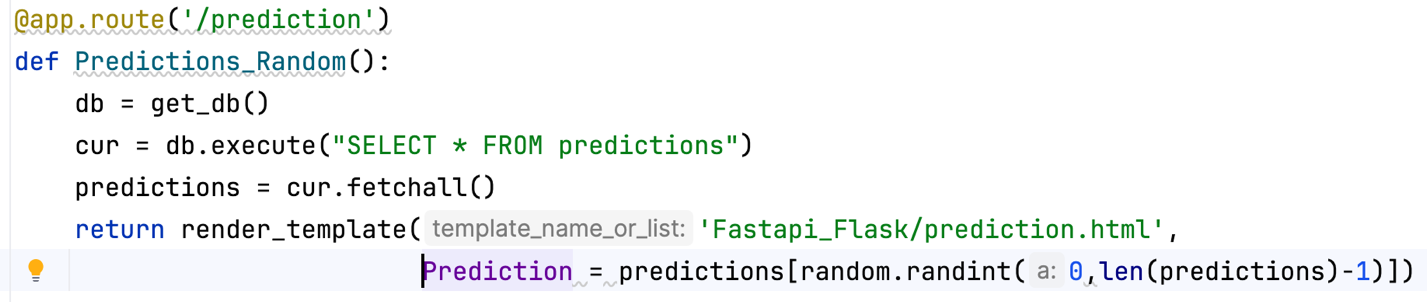


Рис. 6.23 Функции с логикой взаимодействия с сервером

Чтобы запустить проект прописываем команду «app.run()» (рис. 6.24).

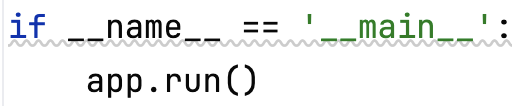


Рис. 6.24 Запуск проекта

# Сравнение фреймворков Django, Flask и FastAPI

После разработки одного и того же сайта на трёх разных фреймворках и анализа получившихся проектов можно прийти к выводу, что разработка на каждом из них хоть и схожа, но имеет отличительные особенности.

*Django*

Разработка на Django подразумевает очень строгие требования к иерархии и архитектуре проекта, но за счёт этого разработчик меньше задумывается о подключении тех или иных приложений к проекту, настройки связи с базой данных, а также меньше беспокоится о защите. Благодаря строгости в архитектуре, большинство моментов по связи отдельных компонентов программы берёт на себя сам фреймворк. Если понять, как именно должны располагаться файлы, то разработка на Django становится не сложной и понятной. Минусом же Django является его же плюс: новичку будет достаточно сложно разобраться в том, как правильно взаимодействовать с файлами фреймворка.

Django использует ORM (Object-Relational Mapping) для работы с базами данных. Это упрощает процесс взаимодействия с базой данных и позволяет сосредоточиться на логике приложения. Сам ORM предоставляет абстракцию над базой данных, позволяя разработчикам работать с объектами Python вместо SQL-запросов.

Django предлагает мощные механизмы для работы с шаблонами и представлениями, что позволяет создавать динамические и интерактивные веб-страницы.

Django поддерживает архитектуру MTV (Model-View-Templates), которая разделяет приложение на три компонента: модель, представление и контроллер. Это улучшает организацию кода и упрощает разработку.

*FastAPI*

Разработка на FastAPI – очень простой и незамысловатый процесс, т.к. в отличии от Django фреймворк не накладывает строгих ограничений на архитектуру проекта, а также не имеет предустановленных инструментов для взаимодействия с базой данных и защиты. FastAPI – пустой холст для разработчика: на данном фреймворке разработчик сможет реализовать практически любую задумку именно так, как он хочет. Также, несомненным плюсом данного фреймворка является разработка посредством асинхронных функций, что делает его крайне быстрым. Ещё один немаловажный плюс – использование декораторов запросов. Данная особенность делает разработку отображений более систематизированной, понятной и удобочитаемой. Название отображает самое явное назначение этого фреймворка: он идеально подходит для разработки различных API вашего сервиса.

Главным минусом является отсутствие предустановленных библиотек для взаимодействия с базами данных и защиты данных (работ с формами). Эта особенность позволяет FastAPI быть менее требовательным к ресурсам сервера, но приводит к сложности со связью между компонентами. Несмотря на то, что уже давно выведен каноничный набор FastAPI, SQLAlchemy и Alembic, возникают сложности с миграциями и взаимодействием с базами данных.

*Flask*

Flask является чем-то средним между Django и FastAPI. В данном фреймворке также предоставлен достаточно малый набор инструментов для разработки приложения. Но, также, как и FastAPI, это дает преимущество перед Django в плане ресурсоемкости. И также, как и FastAPI, Flask не предъявляет очень строгих требований к архитектуре проекта. Но в отличии от FastAPI, данный фреймворк имеет свои аналоги-дополнения популярным библиотекам для взаимодействия с базами данных и прочим. Также, в данном фреймворке по аналогии с FastAPI используются декораторы, что делает разработку систематизированной и удобной для восприятия.

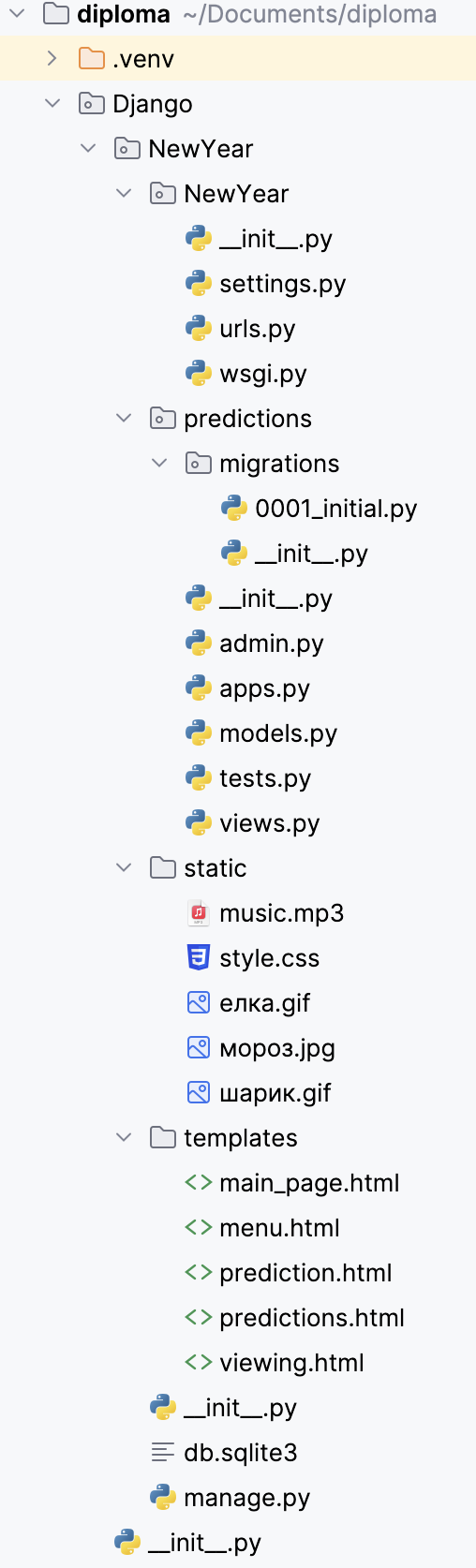
Сравнение фреймворков Django, FastAPI и Flask представлено в таблице.

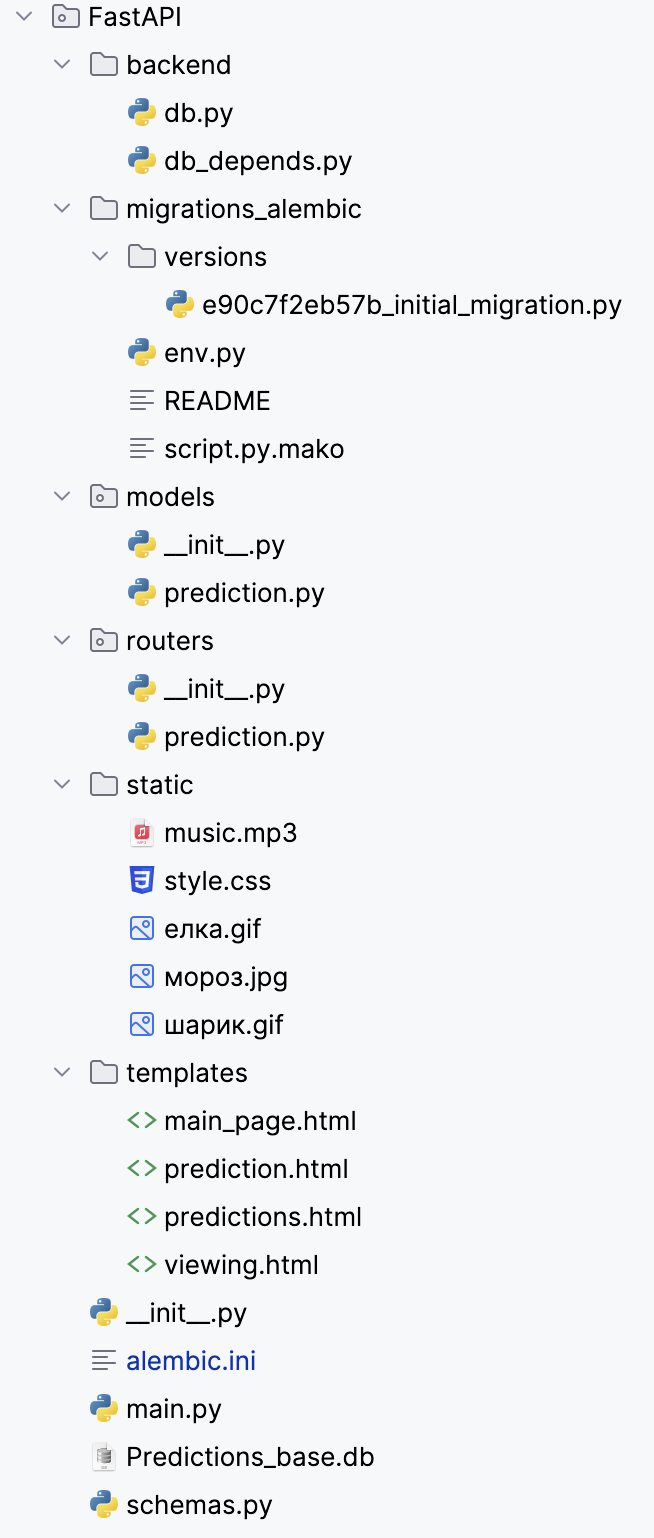
| **Параметры** | **Django** | **FastAPI** | **Flask** |
| --- | --- | --- | --- |
| **Год разработки** | 2005 | 2018 | 2004 |
| **Тип** | Веб-фреймворк с полным стеком | Микро-веб-фреймворк | Микро-веб-фреймворк |
| **Пример использования** | Создание сложных веб-приложений и API | Создание API и микросервисов | Идеально подходит для создания небольших веб-приложений и простых API |
| **Производительность** | Быстрый способ создания большого веб-приложения | Очень быстрый для создания API и микросервисов | Медленнее из-за ручной проверки и синхронизированного программирования. |
| **Масштабируемость** | Масштабируемый, но его ORM и шаблонизатор могут замедлить масштабирование | Высокая масштабируемость, поскольку в нем используется асинхронизация кода и изменение типов. | Сложно масштабировать, поскольку отсутствует встроенная поддержка ORM или кэширования. |
| **Кривая обучения** | Сложный для начинающих | Просто для начинающих | Умеренный для начинающих |
| **Инструменты для работы с базами данных** | Полный набор | Ограничено, встроенной поддержки нет | Ограничено, встроенной поддержки нет |
| **Асинхронное программирование** | Да, это можно сделать с помощью Asyncio, но немного медленнее. | Да, но быстрее благодаря Pydantic. | Нет, но можем мы использовать другие библиотеки. |
| **ORM (Объектно-реляционное отображение)** | ДА | НЕТ | НЕТ |
| **Сообщество** | Большой и активный | Небольшой, но растущий | Большой и активный |
| **Документация** | Огромный | Небольшой, но все еще растущий | Большой |
| **Преимущества** | Это отличный выбор для тех, кто хочет создавать безопасные, масштабируемые, гибкие и простые в обслуживании веб-приложения | Это отличный выбор для создания высокопроизводительных API и микросервисов | Это отличный выбор для создания веб-приложений малого и среднего уровня, где производительность не является большой проблемой и разработчикам требуется гибкость. |
| **Недостатки** | Это может быть сложным для начинающих, сложным в отладке и не подходящим для небольших проектов | Основные файлы могут быть перегружены и отсутствует встроенная защита. | Отсутствует встроенная поддержка кэширования, ORM, асинхронизации и т. д. |

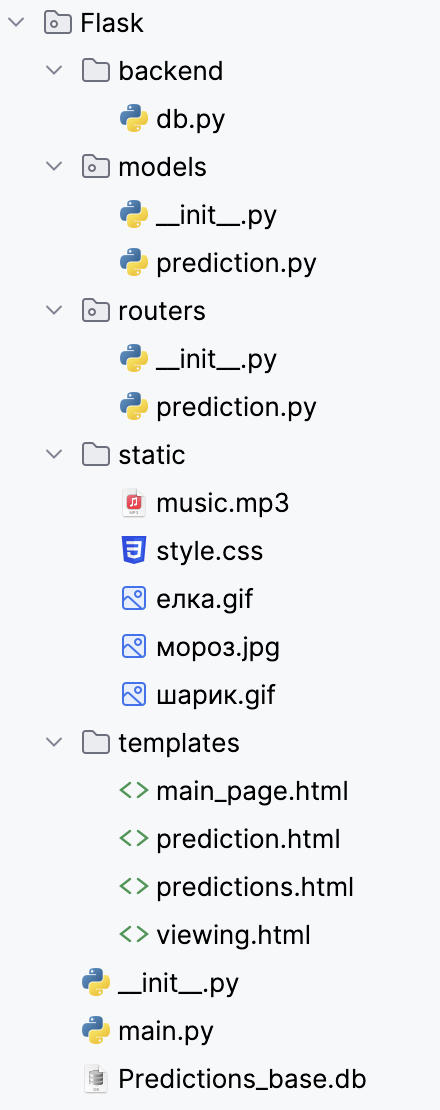
# Заключение

Таким образом, теперь мы можем сделать вывод, что FastAPI можно использовать из-за его высокой производительности при разработке API, Flask можно использовать для создания небольших и средних проектов с большой гибкостью, а Django - для создания многофункциональных крупномасштабных приложений.

# Приложение 1. Пример файловой структуры проекта







# Приложение 2. Список необходимых библиотек

alembic==1.14.0

annotated-types==0.7.0

anyio==4.7.0

asgiref==3.8.1

blinker==1.9.0

click==8.1.7

distlib==0.3.9

Django==5.1.4

fastapi==0.115.6

filelock==3.16.1

Flask==3.1.0

h11==0.14.0

idna==3.10

itsdangerous==2.2.0

Jinja2==3.1.4

Mako==1.3.8

MarkupSafe==3.0.2

platformdirs==4.3.6

pydantic==2.10.3

pydantic\_core==2.27.1

sniffio==1.3.1

SQLAlchemy==2.0.36

sqlparse==0.5.3

starlette==0.41.3

typing\_extensions==4.12.2

uvicorn==0.34.0

virtualenv==20.28.0

Werkzeug==3.1.3