ДИПЛОМНАЯ РАБОТА

по теме:

«АНАЛИЗ И СРАВНЕНИЕ НАПИСАНИЯ WEB-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ РАЗНЫХ ФРЕЙМВОРКОВ»

|  |  |
| --- | --- |
| Исполнитель: | Дудоров Артём Витальевич |

2024

# Содержание

[Введение 4](#_Toc173686131)

[Обоснование выбора темы 4](#_Toc173686132)

[Определение целей и задач исследования 4](#_Toc173686133)

[1 Основные понятия и определения 5](#_Toc173686134)

[2 Обзор популярных востребованных веб-фреймворков 6](#_Toc173686135)

[3 Выявление ключевых особенностей веб-фреймворков 8](#_Toc173686136)

[3.1 Django 8](#_Toc173686137)

[3.2 FastAPI 8](#_Toc173686138)

[3.3 Flask 9](#_Toc173686139)

[4 Разработка веб приложений 10](#_Toc173686140)

[4.1 Основная концепция веб-приложения 10](#_Toc173686141)

[4.2 Разработка в соответствии с созданной документацией 10](#_Toc173686142)

[4.2.1 Реализация веб-приложения на Django 10](#_Toc173686143)

[4.2.2 Реализация веб-приложения на FastAPI 15](#_Toc173686144)

[4.2.3 Реализация веб-приложения на Flask 17](#_Toc173686145)

[5 Анализ и интерпретация результатов 22](#_Toc173686146)

[5.1 Django 22](#_Toc173686147)

[5.2 FastAPI 22](#_Toc173686148)

[5.3 Flask 23](#_Toc173686149)

[Заключение 24](#_Toc173686150)

[Дальнейшие планы 25](#_Toc173686151)

# Введение

## Обоснование выбора темы

В настоящий момент, всё больше компаний ищут исполнителей, способных качественно разработать веб-приложение под их потребности, а также улучшить или доработать уже имеющиеся веб-приложения. Если говорить про сферу фриланса, то большинство предложений в среде программирования связано с работой с веб-приложениями: доработка, исправление, разработка.

Сейчас необходимость в специалистах веб-разработки ощущается особенно остро, и можно смело говорить, что в ближайшие несколько лет данная потребность не изменится.

Помимо вышеперечисленного выбранная тема дипломной работы позволит закрепить полученные и приобрести новые, более глубокие знания в написании веб-приложений, что будет несомненным плюсом в личном портфолио каждого программиста.

Таким образом данная тема обусловлена актуальностью в настоящий момент, перспективностью в будущем и практической значимостью в обучении.

## Определение целей и задач исследования

Цель исследования: выявить наиболее удобный и качественный фреймворк для разработки сайтов и веб-приложений, либо же определить, для каких задач больше подходят наиболее известные фреймворки.

Задачи исследования:

* определение наиболее популярных и востребованных фреймворков;
* выявление ключевых особенностей выбранных фреймворков;
* разработка веб-приложения на каждом из них на тему «Магазин приложений»;
* сравнение итоговой реализации приложений.

# Основные понятия и определения

1. Веб-приложение (web application) – программное приложение, работающее на веб-сервере через браузер.
2. Бэкенд (backend) – серверная часть веб-приложения, обрабатывающая запросы от клиента, выполняющая логику приложения и взаимодействующая с базой данных.
3. Фронтенд (frontend) – клиентская часть веб-приложения, взаимодействующая непосредственно с пользователем, отвечающая за отображение данных и предоставления пользователю интерфейса взаимодействия с приложением.
4. Веб-фреймворк (Web-framework) – набор инструментов для написания бэкенда веб-приложения.
5. ORM (Object-Relational Mapping) – технология взаимодействия с базой данных через объектно-ориентированный интерфейс, скрывая детали SQL-запросов.

# Обзор популярных востребованных веб-фреймворков

Проанализировав вакансии на таких сайтах, как hh.ru, career.habr.ru, Yandex.ru, gorodrabot.ru и других, становится ясно, что основной стек разработки, который ожидают от backend разработчика на python это:

* Django,
* DRF,
* FastAPI,
* Flask.

Помимо основного технологического стека, необходимо знать, как работать с SQL и PostgreSQL (также встречаются требование знания MySQL или sqlite).

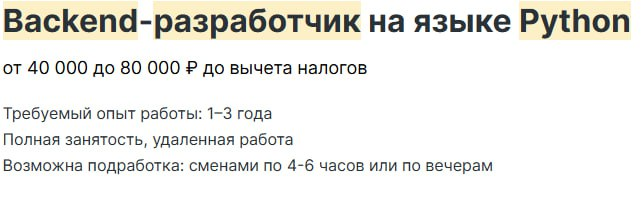


Рисунок 2.1 – вакансия на hh.ru



Рисунок 2.2 – требования к кандидату

# Выявление ключевых особенностей веб-фреймворков

Разберём ключевые особенности каждых из наиболее востребованных фреймворков.

## Django

Django следует философии "batteries-included" (все необходимые компоненты "из коробки"), что означает, что он предоставляет обширный набор встроенных инструментов и функций для разработки веб-приложений. Это включает в себя:

* автоматически создаваемая административная панель, позволяющая управлять данными вашего приложения без написания кода;
* инструмент для взаимодействия с базами данных, позволяющий работать с данными как с объектами Python;
* механизм для валидации и обработки пользовательских вводов;
* модели, обеспечивающие легкость определения структуры базы данных с использованием Python-классов.

В архитектурном плане Django использует паттерн MVT (Model-View-Template), что позволяет разделить логику приложения на три взаимосвязанных компонента:

* Model – модель, отвечающая за структуру и поведение данных;
* View – представление, обрабатывающая логику отображения данных;
* Template – шаблон, отвечающий за оформление и отображение пользовательского интерфейса.

## FastAPI

FastAPI — это относительно новый, высокопроизводительный веб-фреймворк для создания API на языке Python. Он разработан для обеспечения высокой скорости разработки и работы приложений, поддерживает асинхронное программирование и соответствует современным стандартам API, таким как OpenAPI и JSON Schema. Ключевыми особенностями FastAPI являются:

* высокая производительность, так как фреймворк основан на Starlette и Pydantic, а также использует асинхронное программирование, что позволяет обрабатывать множество запросов одновременно, не блокируя сервер;
* автоматическая генерация документация на основе аннотации типов Python, которая доступна в интерактивной форме через Swagger UI и ReDoc:
* высокая скорость разработки с минимальным количеством кода.

## Flask

Flask — это легковесный веб-фреймворк для Python, который известен своей простотой и гибкостью. Он следует принципу минимализма и предоставляет разработчикам базовые инструменты для создания веб-приложений, оставляя возможность добавлять нужную функциональность с помощью расширений. Ключевыми особенностями Flask являются:

* легковесность за счёт минимально набора инструментов для создания веб приложений;
* фреймворк не накладывает ограничения на архитектуру приложения, благодаря чему разработчик может выстроить архитектуру так, как требуется ему;
* поддержка множество сторонних расширений, таких как интеграция с базами данных, системами аутентификации и обработкой форм;
* поддержка шаблонов Jinja2 для создания HTML-шаблонов.

# Разработка веб приложений

## Основная концепция веб-приложения

В ходе исследования будет разрабатываться небольшой вебсайт с формой регистрации и авторизации, главной страницей, страницей товаров, страницей корзины, страницей купленных товаров.

Пользователь должен иметь следующие возможности:

* зарегистрироваться, если впервые на сайте;
* авторизироваться, если не впервые на сайте;
* просмотреть доступные к покупке приложения;
* купить желаемый товар;
* просмотреть купленные приложения.

Для хранения данных будет использована реляционная база данных sqlite.

## Разработка в соответствии с созданной документацией

### Реализация веб-приложения на Django

Разработка приложения на Django начинается с инициализации проекта с помощью команды «django-admin startproject <название проекта>». Данная команда создаёт директорию проекта с сайтом, в которой будут находится приложения, инструменты для администрации, настроек и маршрутизации по сайту. В моём случае проект был назван «software-shop».

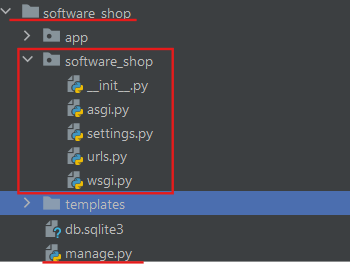


Рисунок 1.1 – созданная директория «software-shop»

После этого необходимо создать приложение в проекте с помощью команды «python manage.py startapp <название приложения>». Перед тем, как выполнять данную команду необходимо перейти в директорию проекта в терминале с помощью команды «cd <название проекта>». После выполнения команд будет создана директория с приложением (в моём случае app), в которой находятся инструменты для настройки отображения страниц, взаимодействия с базой данных и аутентификации пользователей.

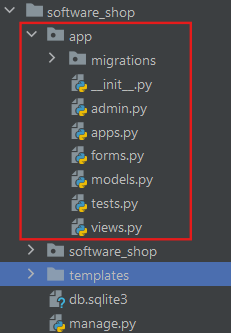


Рисунок 1.2 – созданная директория «app» с приложением

Чтобы настроить отображение на сайте, необходимо использовать файл «views.py». В ней пишутся функции и классы, в которых настраивается логика взаимодействия с сервером.

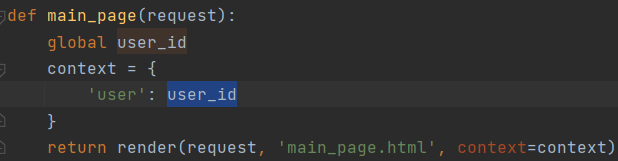


Рисунок 1.3 – реализация главной страницы на Django

Чтобы настроить взаимодействие с базой данных, для начала нужно выполнить команды «python manage.py makemigrations» для проверки не сохранённых связей таблиц и «python manage.py migrate» для сохранения этих связей. Данные команды необходимо использовать каждый раз при добавлении новых моделей (таблиц) в ваше приложение. Для создания новых моделей необходимо пользоваться файлом «models.py», в котором будут создаваться таблицы (в моём случае «Buyers» и «Games»).

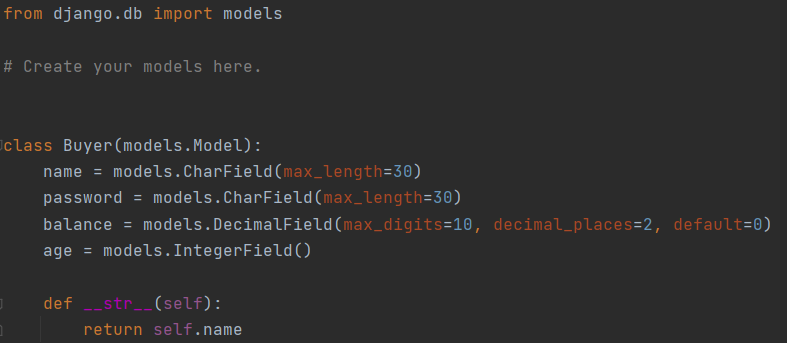


Рисунок 1.4 – класс таблицы данных Buyer

Для настройки маршрутизации по сайту необходимо использовать файл «urls.py» из директории с проектом («software-shop»). В файле необходимо указать URL и функцию, которая должна быть вызвана при запросе этого URL.

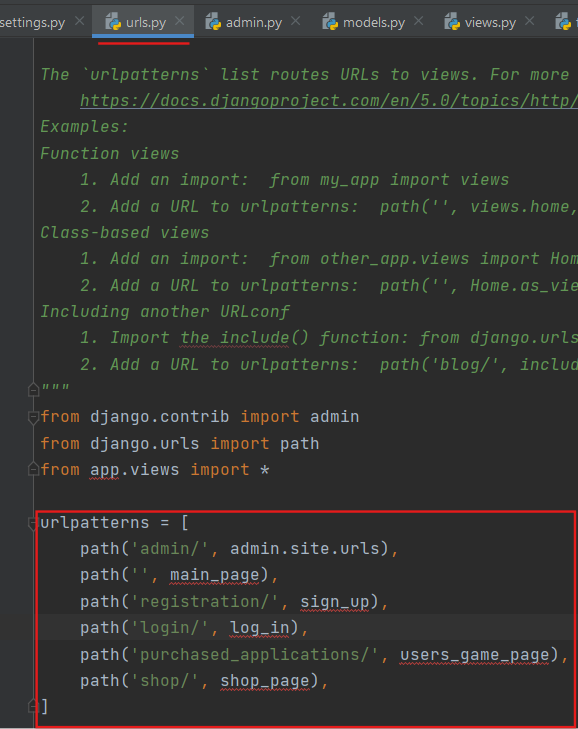


Рисунок 1.5 – настройка маршрутизации в файле «urls.py»

Для того, чтобы взаимодействовать с отображениями страниц также необходимо их подключить к проекту. Чтобы это сделать, необходимо создать папку «templates» в директории с проектом, добавить в неё файлы с расширением «.html», после чего подключить эти шаблоны в файле «settings.py». После этого добавленные шаблоны можно будет использовать в представлениях.

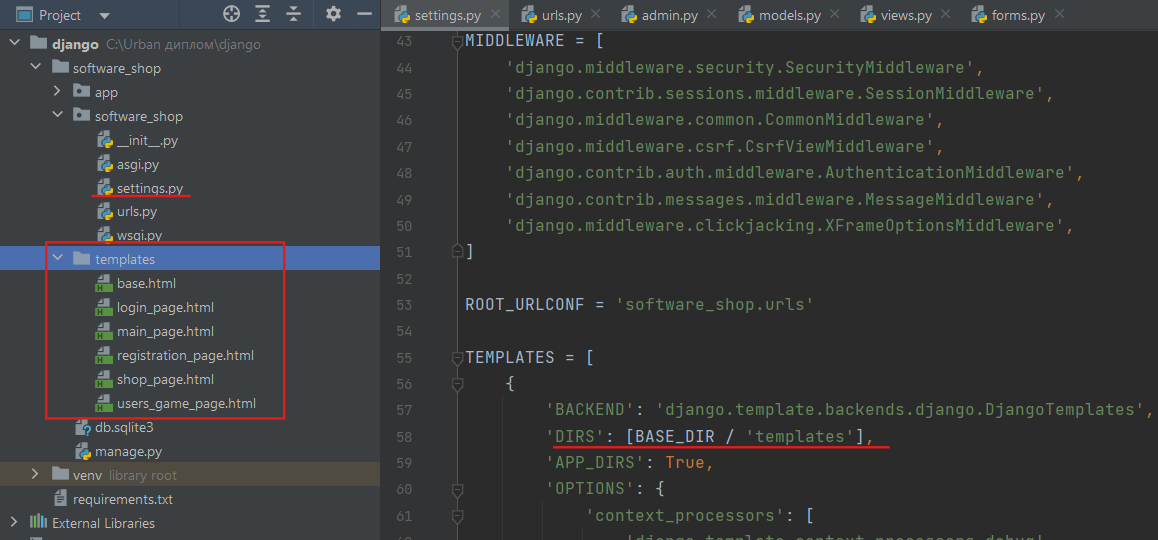


Рисунок 1.6 – добавление шаблонов и подключение их к проекту

Также к проекту необходимо подключить и само приложение в том же файле «settings.py», иначе при запуске сервера будет ошибка, либо ничего не будет отображено.

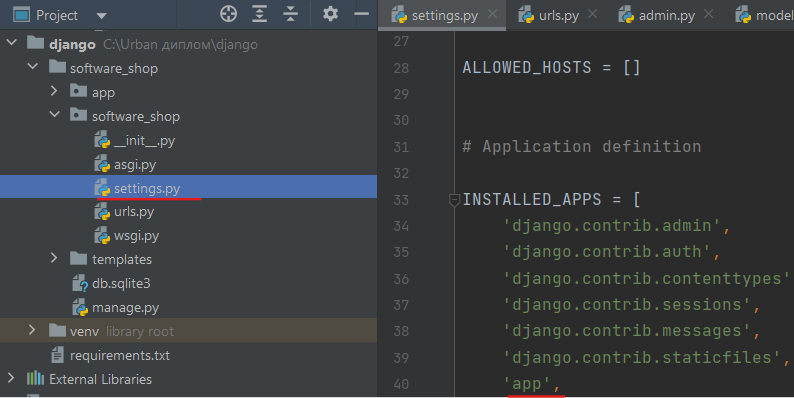


Рисунок 1.7 – подключение приложения к проекту

Чтобы запустить сайт необходимо в консоли выполнить команду «python manage.py runserver», после чего можно будет проверить созданное приложение.

### Реализация веб-приложения на FastAPI

Для разработки проекта на FastAPI достаточно создать файлы, в которых будет находится приложение. Для файлов приложения для удобства была создана папка «app», в которой будут находится все необходимые настройки приложения. Файл запуска «main.py» будет находится вне этой папки.

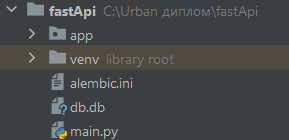


Рисунок 2.1 – созданные папка «app» и файл «main.py»

В папку «app» были добавлены: папка с шаблонами страниц и папка для файлов работы с базой данных. Также был создан файл «\_\_init\_\_.py», чтобы папка «app» идентифицировалась как модуль.

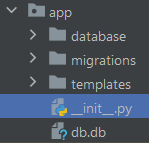


Рисунок 2.2 – наполнение папки «app»

После первичной настройки директории, в проект были также установлены «SQLAlchemy» для работы с базой данных и «alembic» для осуществления миграций баз данных.

Следующим этапом стала первичная настройка приложения, добавление функционала к шаблонам страниц. Отличительной особенностью разработки при помощи FastAPI реализация функционала в виде асинхронных функций с декораторами.

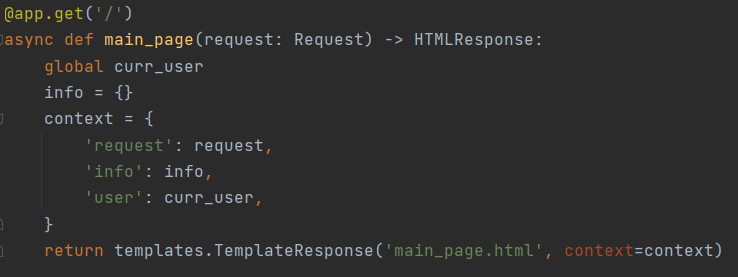


Рисунок 2.3 – реализация главной страницы с помощью FastAPI

Для отображения шаблона какой-либо страницы всегда необходимо возвращать серверу ответ с данным шаблоном с помощью TemplateResponse из модуля «Jinja2Templates» библиотеки «fastapi.templating». Также, неприятной особенностью является то, что если в шаблон не была передана какая-либо переменная, то будет возникать ошибка, из-за чего работа с наследованием шаблонов осложняется: необходимо более детально продумывать, какие блоки должны быть унаследованы и в каком формате.

После первичной настройки приложения была произведена интеграция базы данных в проект. Процесс настройки базы данных (БД) с помощью SQLAlchemy похож на процесс настройки БД в Django с тем отличием, что подключение к БД может быть произведено в любом файле. Для удобства данный файл был назван «db.py»

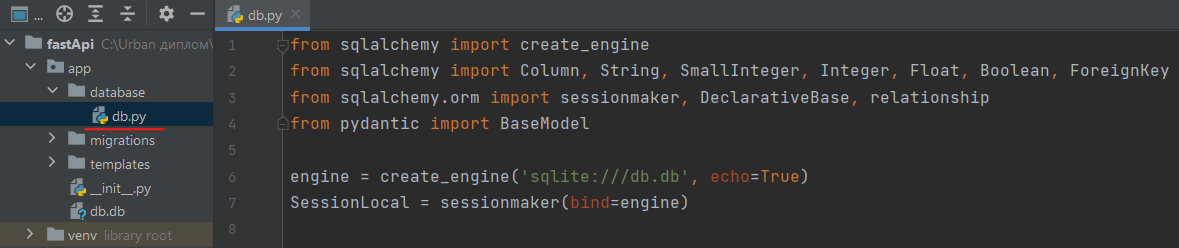


Рисунок 2.4 – инициализация базы данных

После инициализации БД было необходимо произвести миграцию, чтобы создался файл с таблицами и было выполнено подключение БД к проекту. Для этого был использован «alembic». С помощью команды «alembic init app/migrations» в папке приложения была создана папка «migrations», в которой будут находится версии миграции, а также скрипт этих миграций. Также был создан файл «alembic.ini» с помощью которого настраивается взаимодействие с БД.

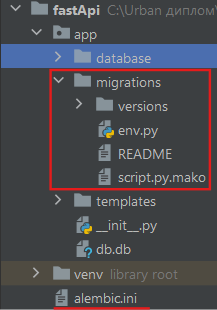


Рисунок 2.5 – миграция базы данных

После инициализации среды миграций необходимо в файле «alembic.ini» указать url для подключения к БД, а в файле «migrations/env.py» указать модели таблиц.

### Реализация веб-приложения на Flask

Разработка приложения на Flask есть нечто среднее между FastAPI и Django. От FastAPI фреймворк взял особенность того, что нет особой строгости к архитектуре проекта, а также лёгкую интеграцию дополнительных модулей. От Django же фреймворк перенял особенность написания кода: при разработке используются обычные функции, но обернутые в декораторы, как в FastAPI.

Чтобы начать разработку на фреймворке Flask необходимо определить директорию и создать файл, в котором будет расположен код запуска приложения. В своей разработке я использовал файл «main.py» с основным кодом приложения, а также директорию «app», в которую поместил основные его настройки.

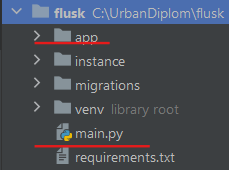


Рисунок 3.1 – директория проекта

После этого, в папке «app» были созданы: папка «databases» для настроек подключения к базе данных, папка «templates» с шаблонами страниц и файл «forms.py», в котором будут находится формы регистрации и авторизации.

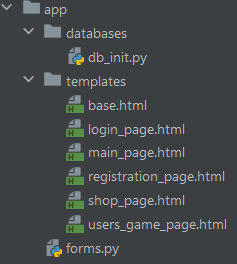


Рисунок 3.2 – папка «app»

После подготовки директории проекта были написаны функции отображения страниц. Для удобства разработки во Flask «с коробки» имеются все необходимые инструменты для написания базового функционала работы сайта.



Рисунок 3.3 – реализация главной страницы с помощью Flask

Чтобы запустить приложение во Flask, в программе необходимо инициализировать объект «Flask», передав в него имя файла, из которого запускается приложение. Дополнительно, можно сразу указать путь до папки с шаблонами страниц.

Как видно на рисунке, реализация отображения схожа с фреймворком FastAPI, но здесь вместо указания конкретного декоратора запроса используется декоратор «route», в котором указывается URl страницы. Также, можно заметить, что передавать в функцию ничего не надо: чтобы получить какую-то информацию из запроса, необходимо использовать импортированную функцию «request» сразу же внутри отображения. Параметры для шаблона указываются аналогично FastAPI с тем отличием, что необходимо передавать только именованные параметры. Здесь нет возможность передать параметры через «context=context».

После реализации базового функционала отображений, было произведено подключение базы данных. Чтобы это сделать, необходимо импортировать библиотеки Flask-SQLAlchemy и Flask-Alembic (или Flask-Migrations). После установки данных библиотек, реализуются классы-таблицы базы данных. В моём случае эта настройка была выполнена в файле «db\_init.py».

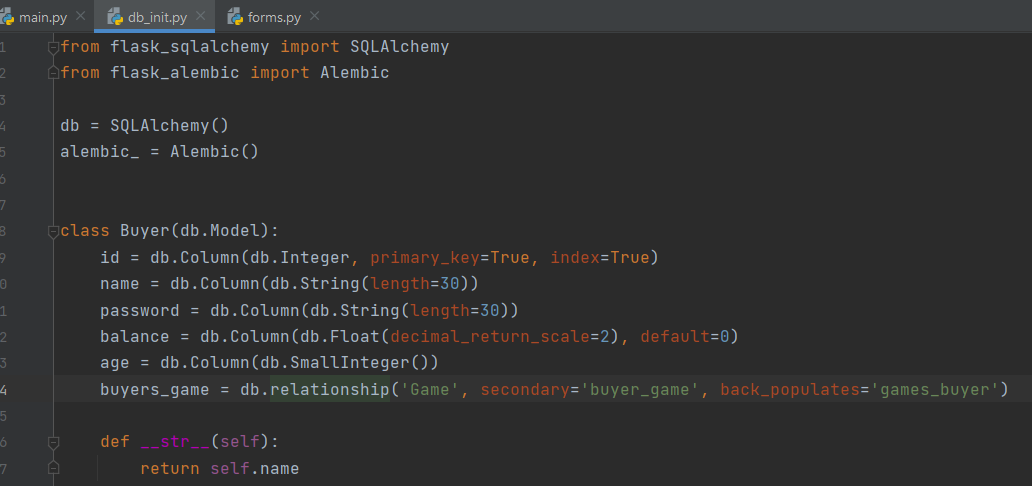


Рисунок 3.4 – реализация таблицы Buyer

Так как для работы с базами данных была использована библиотека Flask-SQLAlchemy, то реализация таблиц похожа на обычный SQLAlchemy. Важной особенностью является то, что необходимо объекты SQLAlchemy и Alembic, чтобы подключить БД к проекту.

Чтобы подключить созданную БД к приложению, в главном файле необходимо прописать URI базы данных и указать объектам БД и Alembic само приложение.

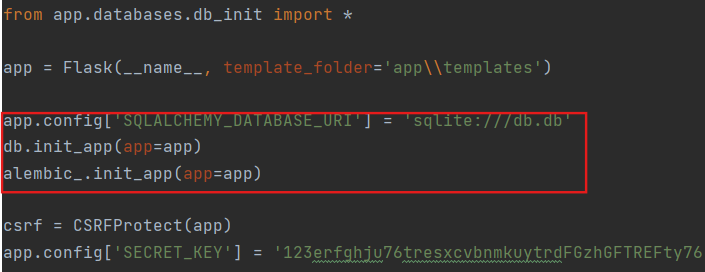


Рисунок 3.5 – подключение БД к проекту

Чтобы произвести первую ревизию, используется команда «flask db revision <название миграции>». В процессе разработки я столкнулся с проблемой, что Flask не видит файл с фабрикой приложений («main.py»). Чтобы решить эту проблему можно использовать ключ «-A», который позволяет явно указать, в каком файле расположена фабрика. Таким образом, для первой миграции я использовал команду «flask -A main.py db revision ‘init’». После выполнения этой команды рядом с файлом «main.py» создаются папки «migrations», в которой хранятся версии изменений, и «instance», в которой хранится база данных.

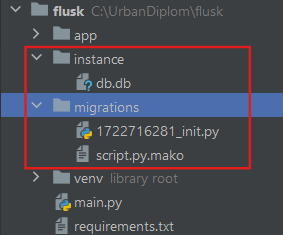


Рисунок 3.6 – первая миграция

После выполнения команды ревизии, была использована команда «flask -A main.py upgrade ‘head’» для того, чтобы применить последнюю миграцию.

После подключения и настройки взаимодействия с базой данных, приложение готово.

Ещё стоит упомянуть про дополнительный модуль Flask-WTF, который делает работу с формами более удобной и безопасной. Данный модуль предоставляет настраиваемую защиту на базе csrf-токена. Но, чтобы использовать этот токен, необходимо явно указать его, как переменную окружения. Делается это непосредственно в программе. Данную информацию необходимо изолировать, но в качестве исследования, я объявил её в главном файле с помощью строки «app.config['SECRET\_KEY'] = '123erfghju76tresxcvbnmkuytrdFGzhGFTREFty76'».

# Анализ и интерпретация результатов

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.

## 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 «с коробки» 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 у 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. Благодаря этим дополнениям, настройка связи происходит достаточно легко и интуитивно: установил дополнение, подключил его к приложению с помощью метода «init\_app» - готово! Если возникает необходимость что-либо сделать в терминале, то на помощь приходит ключевое слово «flask», в которую сразу же добавляются команды дополнений. Также, в данном фреймворке по аналогии с FastAPI используются декораторы, что делает разработку систематизированной и удобной для восприятия.

# Заключение

В ходе исследования были изучены три фреймворка: Django, FastAPI и Flask. На каждом из фреймворках разработаны приложения с одинаковым функционалом, благодаря чему был произведён анализ процесса разработки и требований каждого фреймворка. После анализа особенностей можно прийти к выводу, что Django отлично подходит для большого сервиса с множеством приложений и микро сервисов, FastAPI идеален для написания небольших приложений, которые могут быть интегрированы в проекты, а Flask удобен при разработки среднего по размеру проекта, не требующего больших затрат ресурсов.

# Дальнейшие планы

Благодаря исследованию наиболее известных фреймворков для разработки веб-приложений я улучшил качество своего портфолио backend-разработчика. Теперь я знаю особенности разработки web-приложений и frontend-разработки. С помощью этого портфолио в дальнейшем я смогу брать заказы на различных фриланс-биржах, за счёт чего продолжу расширять портфолио, набираться опыта коммерческой разработки и изучать новые фреймворки.