|  |  |
| --- | --- |
|  | **Онлайн-Университет Urban University** |

**Дипломная работа**

тема:

***Сравнение производительности и удобства использования различных ORM (Object-Relational Mapping) библиотек: Django ORM, SQLAlchemy и Tortoise ORM***

|  |
| --- |
| Выполнил ученик курса  Python-разработчик 22 потока: |
| Четин Сергей Николаевич |
| Преподаватель: |
| Бельц Андрей Викторович |

2024

# Содержание

[Содержание 1](#_Toc177421558)

[Введение 2](#_Toc177421559)

[Структура проекта 3](#_Toc177421560)

[Структура базы данных 4](#_Toc177421561)

[1. Django\_ORM 5](#_Toc177421562)

[2. SQLAlchemy\_ORM 10](#_Toc177421563)

[3. Tortoise\_ORM 15](#_Toc177421564)

[Заключение и выводы 16](#_Toc177421565)

[Приложение 1. Пример файловой структура приложения 20](#_Toc177421566)

[Приложение 2. Список основных библиотек, используемых в проекте 21](#_Toc177421567)

# Введение

Тема дипломной работы выбрана мною, исходя из потребностей одного из моих товарищей, который работает на складе продукции нашей компании и вынужден каждый раз чуть ли не вручную выполнять спектр задач по учёту товара.

Сам я работаю инженером-проектировщиком и когда приходится делать рутинную работу, которая не требует каких-то особых интеллектуальных затрат, то пытаюсь хоть как-то автоматизировать процесс. Так, например, в чертежах AutoCAD я сделал много готовых блоков, готовых решений, шаблонов, которые уже давно используются коллегами в проектировании и помогают сэкономить время на выполнение тех или иных задач. Возможно, именно поэтому я решил ещё дополнительно изучить (хотя бы поверхностно) язык программирования Python, чтобы полученные знания можно было хоть как-то использовать также для совершенствования рабочих процессов. На текущий момент времени могу сказать, что, поступив в Урбан Университет, я ничего не знал и пройдя этот сложный путь обучения, который может будет длиться ещё и ещё, без конца и края, так как практически невозможно познать всего, сейчас я уже многое понимаю о работе программиста.

Но вернёмся к теме… Что такое ORM?

*согласно сайту Википедия:*

**ORM** (англ. Object-Relational Mapping, рус. *объектно-реляционное отображение*, или преобразование) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

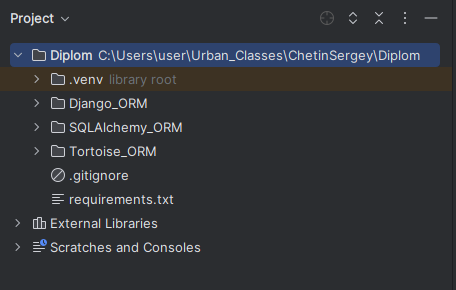
То есть, перефразируя свои словами, с помощью ORM-моделей, используя различные библиотеки на языке Python (например:   
Django\_ORM, SQLAlchemy\_ORM, Tortoise\_ORM, Peewee\_ORM, Pony\_ORM, Tryton\_ORM и многие другие) можно создавать и редактировать некую базу данных.  
Таким образом, если я правильно понимаю, ORM-модели помогают программистам настроить систему управления базами данных (СУБД) и, используя различные возможности тех или иных библиотек, можно настраивать связи между несколькими базами данных по определенным «ключам».

Звучит заманчиво, подумал я и решил попробовать создать некую структуру для подсчёта болтов, гаек, шайб, которые дальше можно собрать в болтовое соединение и посредством ORM-моделей решить проблему учёта данных изделий на складе, где работает мой товарищ.

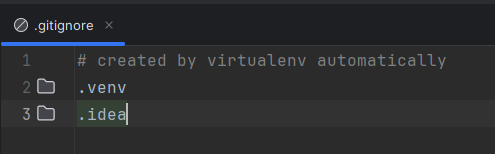
К сожалению, далеко не все задумки удалось реализовать на текущий момент времени и результат можно улучшать бесконечно, но как говорится, что имеем, то имеем… Итак было потрачено много усилий, времени и нервов, надеюсь, что в таком виде реализация подойдёт для сдачи проекта.

# Структура проекта

Согласно темы дипломной работы необходимо разработать приложения для работы с базами данных с использованием Django\_ORM, SQLAlchemy и Tortoise\_ORM, таким образом в одном проекте я создал три директории с соответствующими названиями и погружаясь в ту или иную директорию, можно будет запускать интересующие нас приложения.



И сразу добавил ”.gitignore”, куда поместил папки ”.idea” и ”.venv” (содержится информация о виртуальном окружении (установленных библиотеках, модулях и так далее). Данное действие необходимо, чтобы эта информация не отправилась в репозиторий на github.



Теперь, все подготовительные действия проведены, можно начинать по порядку…

## Структура базы данных

Соответственно, имея номенклатуру болтов, гаек и шайб, мы собираем болтовое соединение, а далее формируем заказ. Казалось бы, простая схема, но мне как новичку в этом вопросе пришлось потратить кучу времени на поиск тех или иных возможностей в решении возникающих проблем.

## Django\_ORM

Django – это популярный веб-фреймворк для создания масштабируемых и надежных веб-приложений. Одним из основных компонентов Django является система объектно-реляционного отображения (ORM), которая позволяет разработчикам взаимодействовать с базами данных с помощью кода на Python, а не писать необработанные SQL-запросы.

С помощью Django ORM можно создавать, извлекать, обновлять и удалять записи в базе данных, используя объекты и методы Python.

Кроме того, Django ORM предоставляет QuerySet API, который позволяет выполнять сложные запросы к вашей базе данных. QuerySet – это коллекция объектов базы данных, которые можно фильтровать, упорядочивать и нарезать, чтобы ограничить результаты определенным подмножеством.

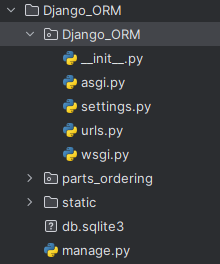
Работа в Джанго начинается с его установки, можно в консоли написать:



Далее создаётся проект следующей командой и названием проекта:



По умолчанию создаётся структура встроенных файлов, которые необходимы для корректной работы Django.

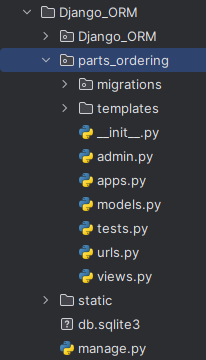


* manage.py — исполняет различные команды, в частности запускает приложение;
* \_\_ init.py \_\_ — стандартный для Python файл, придающий папке, в которой он располагается, свойства модуля;
* settings.py — отвечает за настройки проекта;
* urls.py — включает шаблоны URL-адресов, чем задает систему маршрутизации;
* wsgi.py — содержит параметры конфигурации WSGI (Web Server Gateway Interface), которые применяются при развертывании проекта.

Далее уже можно приступить к созданию самого приложения, в котором и будет структура кода. Проект создаётся следующей командой:



Приложение под названием «parts\_ordering» создаётся также с предустановленными файлами, которые частично могут понадобится в работе проекта.



Основные файлы это:

* admin.py – где через права администратора регистрируются изделия;
* models.py – где хранятся модели наших изделий;
* urls.py – где прописаны различные пути для адресной строки браузера;
* views.py - это то место, где мы размещается «логику» работы нашего приложения. Оно запрашивает информацию из модели, которую мы создали ранее, и передаёт её в шаблон.

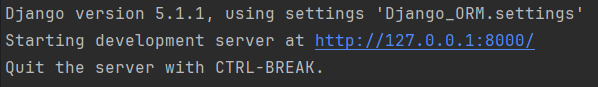
Чтобы запустить выполнение проекта, нужно перейти в консоли к его папке



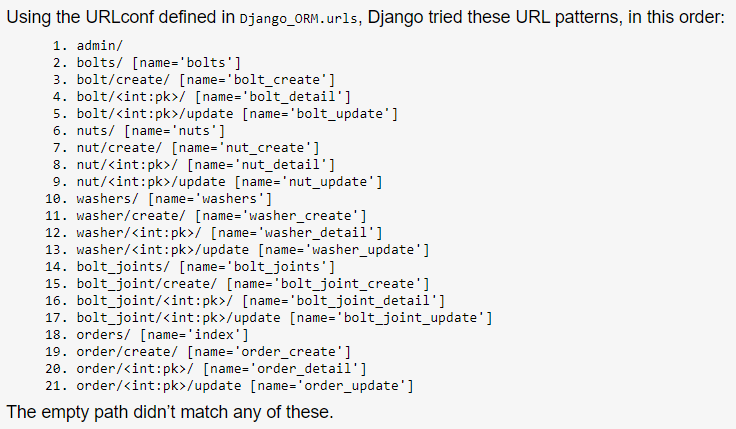
и запустить сервер командой:



После этого в командной строке отобразится адрес, на котором будет размещен проект, — как правило, http://127.0.0.1:8000/. Если ввести его в поисковую строку любого браузера, откроется всё содержимое.

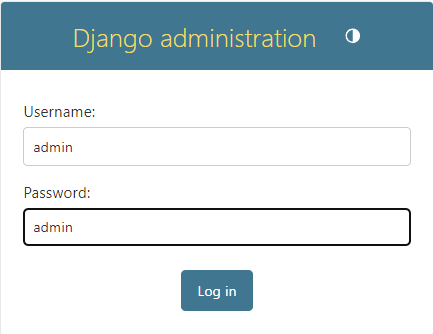


По указанному адресу в браузере появится следующее окно:



где указаны возможные адреса, используемые в приложении.

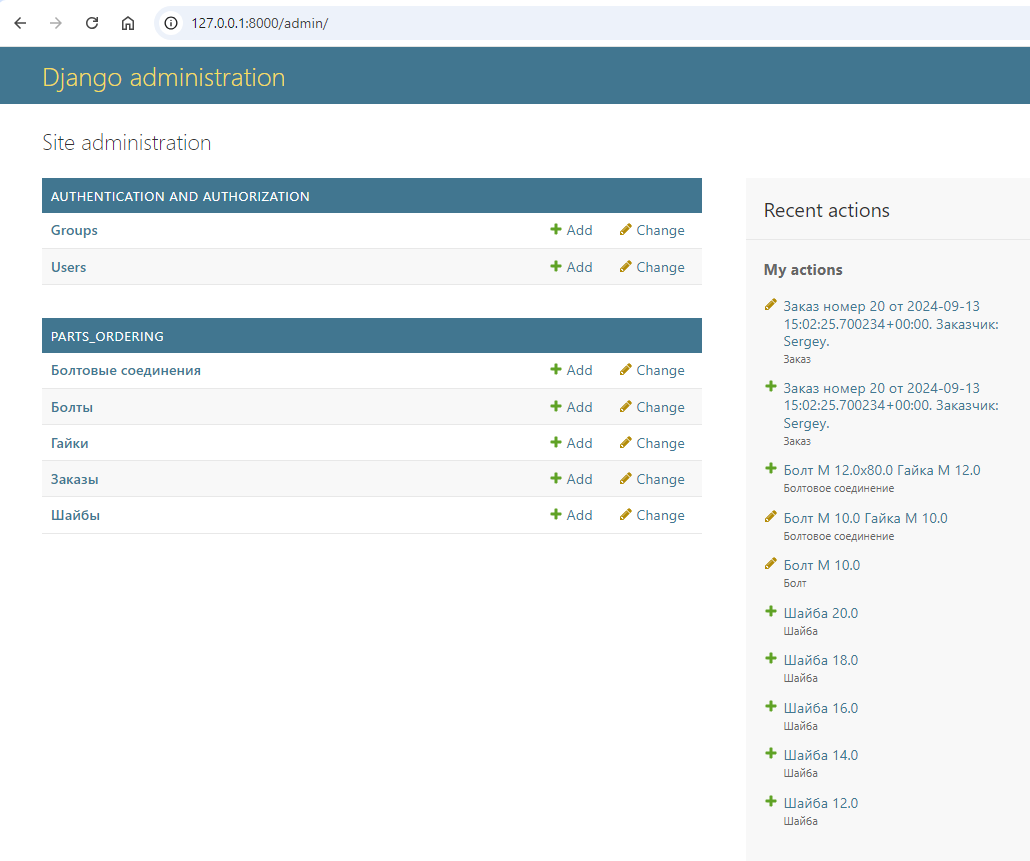
Пройдя по первому адресу, можно будет попасть на страницу регистрации администратора. По умолчанию я сделал



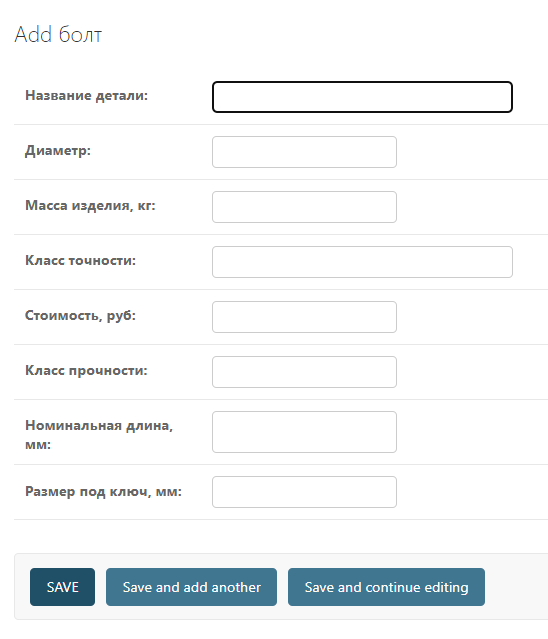
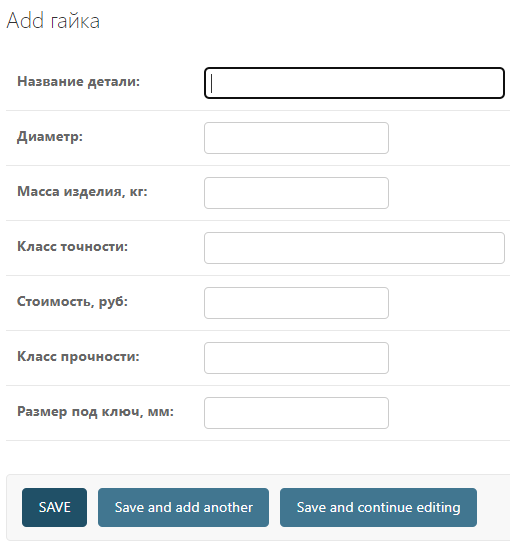
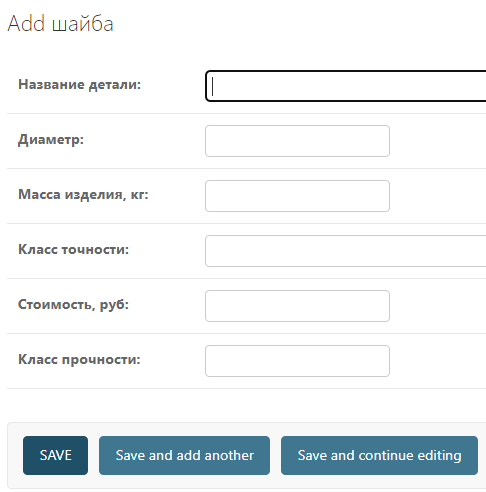
При необходимости можно создать своего суперюзера, командой в консоли:



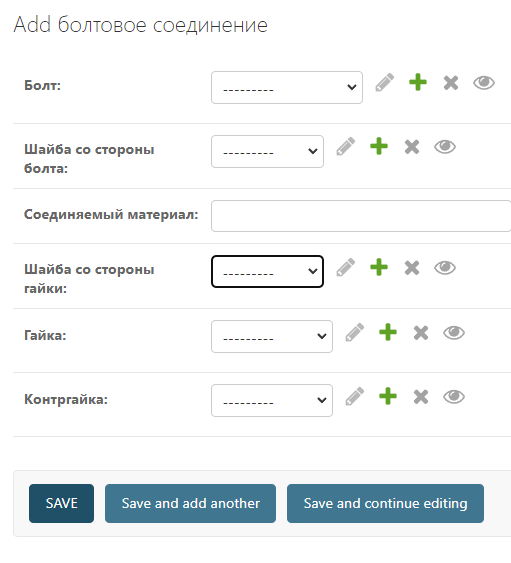
Итак, попадая в приложение, мы видим панель администратора, для создания тех самых интересующих нас изделий (болтов, гаек и шайб) и далее можно собрать их в болтовое соединение и сделать заказ.



Формы создания изделий примерно аналогичные друг другу:

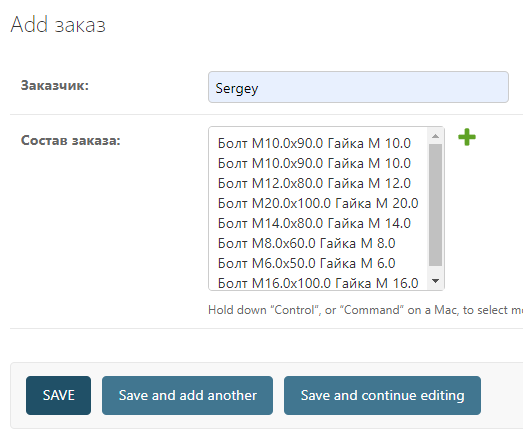
  

А вот создание болтового соединения выглядит следующим образом:



где из списка заготовленных изделий, можно выбрать то, что нужно. Кроме того, нажав на «+» можно прям отсюда создать изделие, которое может быть ещё не добавлено в базу.

В создании заказов следующее окно:



где выделяя либо по 1, либо несколько болтовых соединений, они попадают в заказ после нажатия кнопки «SAVE» и сохраняются в истории.

В таком формате довольно удобно управлять базой данных, где наглядно выбирается объект и вставляется туда, куда нужно, сохраняя при этом связи.

Разумеется, здесь реализованы далеко не все возможности Django\_ORM, всегда есть то, что можно улучшить, однако времени на всё про всё слишком мало…

## SQLAlchemy\_ORM

SQLAlchemy — это Python-библиотека, которая позволяет работать с реляционными базами данных с помощью ORM. Реляционные базы данных хранят информацию в виде связанных между собой таблиц. К таким базам относят Oracle, Microsoft SQL Server, PostgreSQL и другие.

Сила SQLAlchemy — в её ORM. ORM позволяет управлять базами данных с помощью методов объектов в коде и при этом не использовать SQL-запросы. На самом деле это очень удобно, так как позволяет писать привычный код, не переключаясь на SQL.

Ещё одна особенность SQLAlchemy в том, что мы можем заменить базу данных в любой момент — например, с SQLite на PostgreSQL. Код приложения останется ровно тем же и продолжит работать. Нам не придётся переписывать ничего, кроме тех мест, где мы указываем, какую базу данных используем.

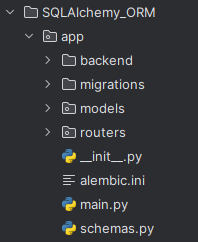
Всё благодаря тому, что библиотека состоит из двух компонентов — SQLAlchemy Core и SQLAlchemy ORM.

* SQLAlchemy Core — это как бы абстракция над SQL. Компонент умеет общаться с SQL и превращать Python-код в понятные для баз данных запросы с помощью SQL Expression Language. При этом он понимает, с каким именно видом хранилища данных взаимодействует.
* SQLAlchemy ORM — удобный для разработчика интерфейс для управления базами данных. Компонент позволяет работать с ними и коммитить запросы прямо в Python-коде — без необходимости переписывать всё на язык SQL.

Работа с SQLAlchemy начинается также с её установки. Для этого необходимо прописать в консоли:



В SQLAlchemy никакой готовой структуры не создаётся, поэтому для создания приложения необходимо вручную создавать директории и пакеты.



Для запуска приложения нам дополнительно потребуется фреймворк. Выбор пал на FastAPI, т.к. он был в учебной программе.

FastAPI — это современный, быстрый (высокопроизводительный) веб-фреймворк для создания API используя Python, в основе которого лежит стандартная аннотация типов Python.

Его также потребуется дополнительно установить



Далее, чтобы запустить сервер нужно переместиться в папку с приложением, это:

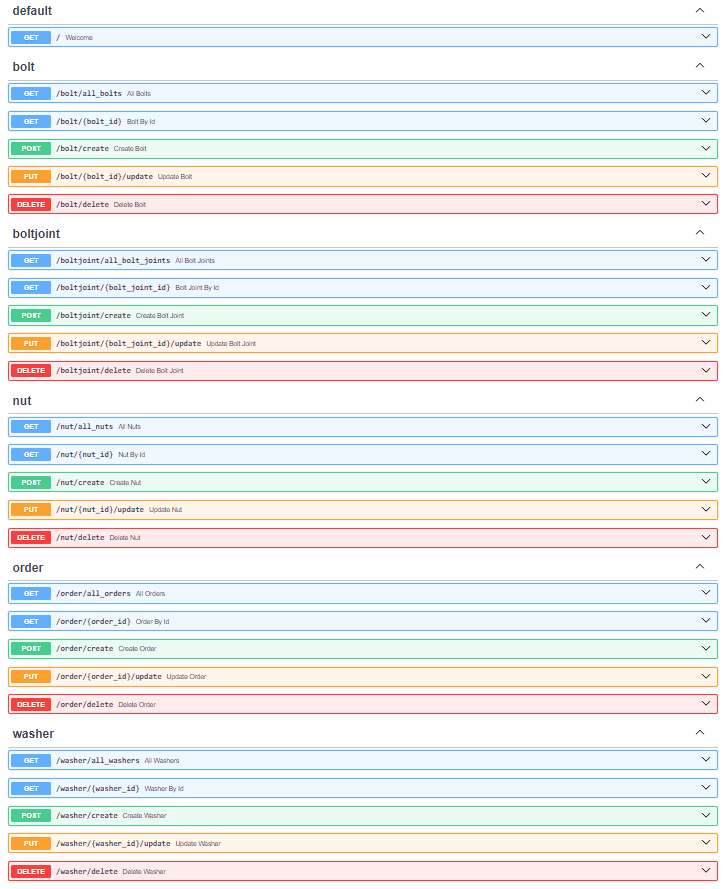


А затем запускаем команду



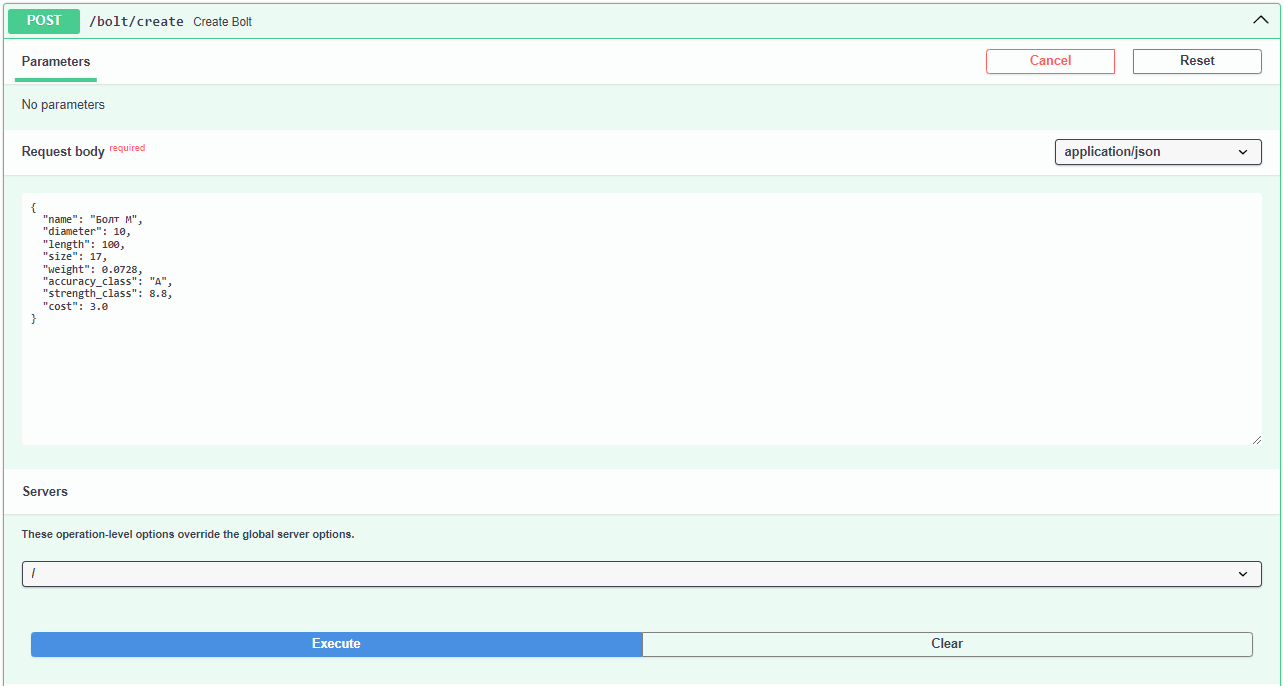
Далее программа запускает создание таблиц, если их не было и через фреймворк FastAPI можно перейти в браузере по адресу:

http://127.0.0.1:8000/docs - попадаем в Swagger, в котором предлагается формировать или редактировать интересующие нас изделия.



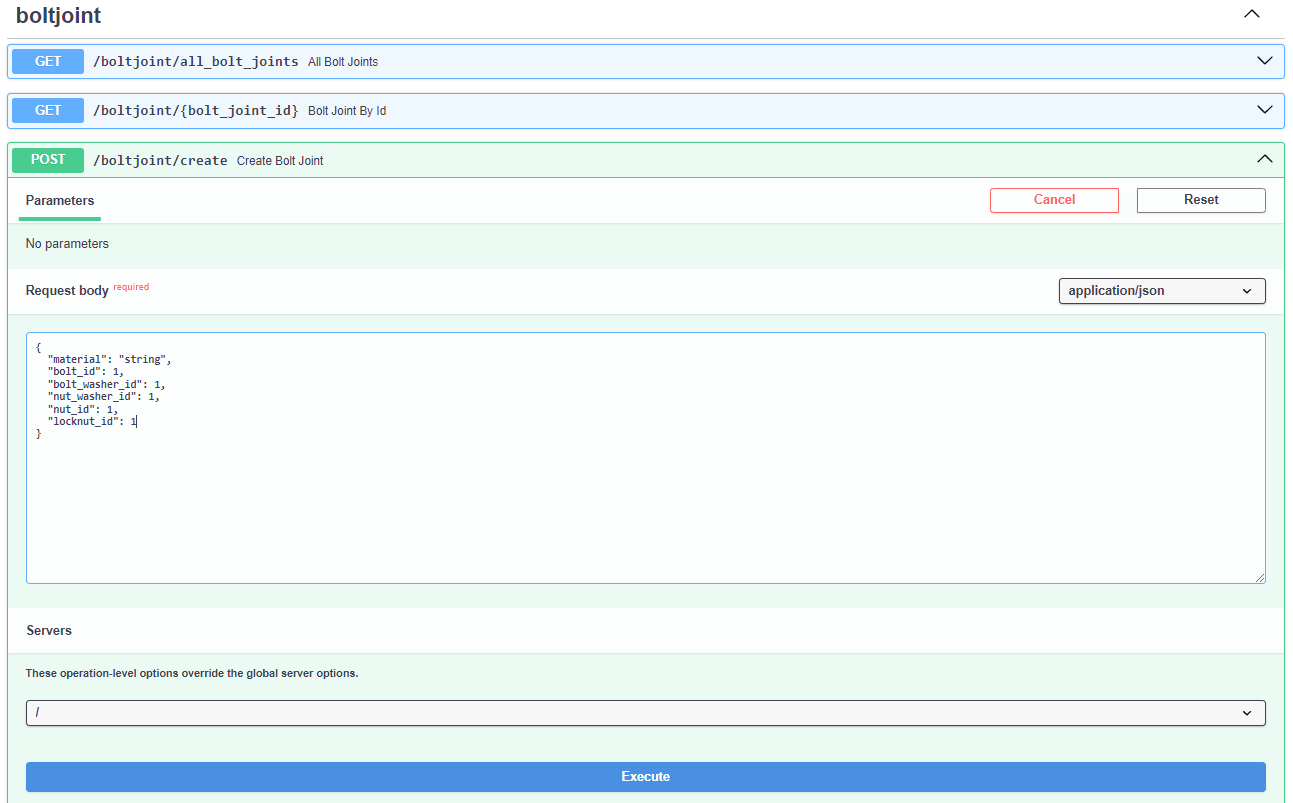
Чтобы создать болт нужно перейти в соответствующий раздел и нажать кнопку

«Try it out», а далее по имеющейся схеме заполнить сведения и потом «Execute»:

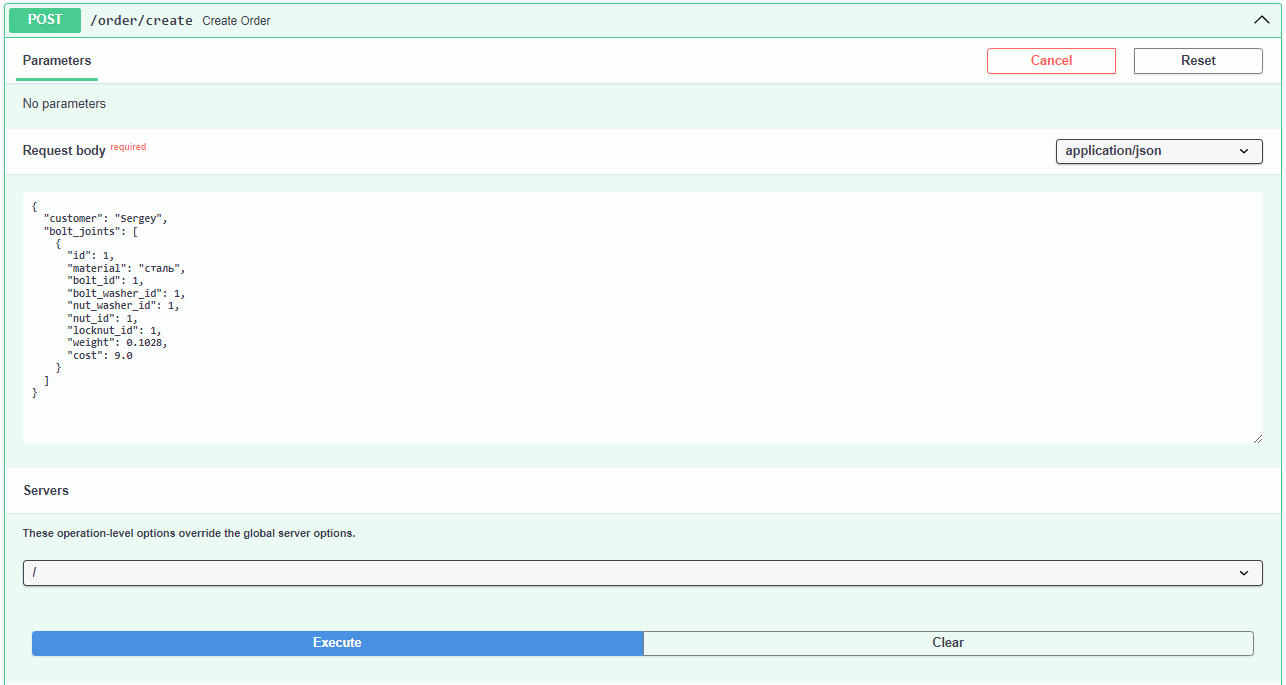


- аналогично в гайках и шайбах…

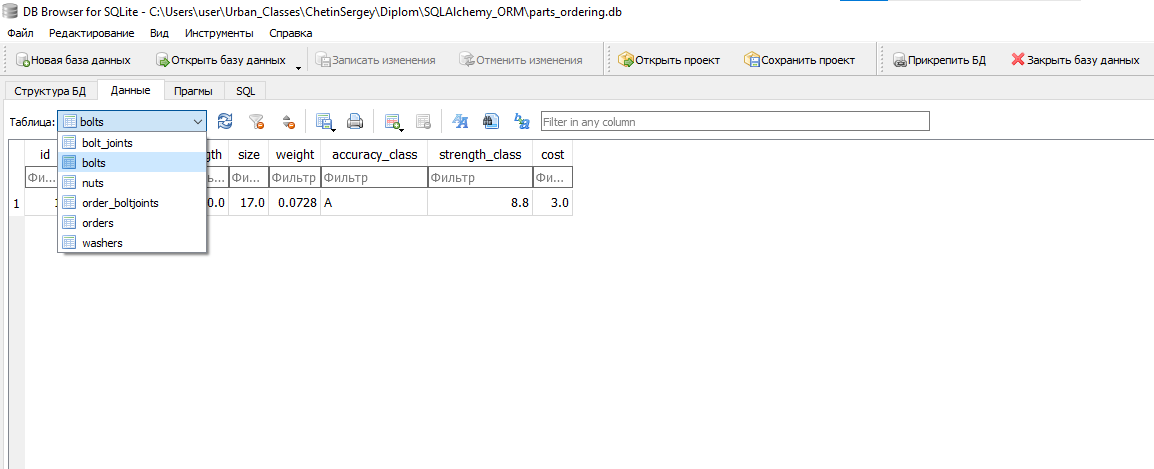
В болтовом соединении нужно просто проставить номера id и оно готово:



При создании заказов также:



Наша база данных со всеми таблицами создаётся в «parts\_ordering.db», который мы можем открыть в обычном DB Browser и посмотреть всю структуру там.



Хочется отметить, что создание изделий в базу данных при таком способе не совсем ускоряет процесс и нужно контролировать номера id у изделий, однако, когда мы видим наглядную структуру таблиц, то понимать как к ним обращаться и что-либо менять через программный код становится проще.

## Tortoise\_ORM

Tortoise ORM — это простой в использовании asyncio ORM (Object Relational Mapper), созданный на основе Django.

Разработчики указывают, что Tortoise ORM — все еще молодая библиотека, и могут появляться изменения, ломающие существующий код при обновлении.

Tortoise ORM был создан с учетом отношений и восхищения превосходным и популярным Django ORM. В его дизайне заложено, что вы работаете не только с таблицами, вы работаете с реляционными данными.

Tortoise ORM был разработан так, чтобы быть функциональным, но в то же время знакомым, чтобы облегчить переход разработчиков, желающих перейти на asyncio (асинхронность).

Queryset Evaluation в Tortoise ORM

Это момент, когда ваш запрос преобразуется в конкретные объекты. В Django это сделано неявно, поэтому очень важно знать, когда ваш Queryset превратится в объекты.

В Tortoise ORM с этим намного проще, так как асинхронность позволила сделать Queryset Evaluation явным. Queryset строится ровно до тех пор, пока на нем не будет вызван await.

Например:

*task\_queryset = Task.filter(name="Task name")  
print(task\_queryset)  
# Output: <Queryset object>  
  
task =* ***await*** *task\_queryset  
print(task)  
# Output: <Task>*

Опять же для запуска приложения потребуется установить библиотеку «tortoise‑orm».



И предварительно провалившись в директорию с Tortoise\_ORM:



Запускаем приложение и заполняем базу данных также как в алхимии:



# Заключение и выводы

Итак, подводя итоги опять же повторим, что же такое ORM?

**ORM** (Object-Relational Mapping) — это технология, которая позволяет работать с базами данных, используя объектно-ориентированный подход. ORM создает объекты, которые представляют данные в базе данных, и позволяет работать с ними так же, как с обычными объектами в коде.

**Преимущества ORM**

***Простота:*** ORM предоставляет простой интерфейс для работы с базой данных, который может быть понятным любому программисту. ORM скрывает сложности SQL-запросов, позволяя работать с данными на более высоком уровне абстракции.

***Переносимость:*** ORM может работать с различными СУДБ, что делает его более переносимым, чем SQL. Это позволяет разработчикам легко переносить свое приложение на другую СУБД без изменения кода.

***Сопровождаемость:*** ORM может значительно упростить сопровождение приложения, так как изменения в структуре базы данных могут быть внесены непосредственно в код ORM, а не в каждый SQL-запрос.

***Безопасность:*** ORM может предотвратить SQL-инъекции, поскольку ORM автоматически экранирует данные, которые передаются в базу данных.

**Недостатки ORM**

***Сложность:*** ORM может быть сложным для понимания, особенно для новых разработчиков. ORM требует определенных знаний и опыта, чтобы использовать его эффективно.

***Производительность:*** ORM может быть менее эффективным, чем работа с SQL напрямую. ORM должен обрабатывать запросы и преобразовывать их в SQL, что может замедлить производительность.

***Ограничения:*** ORM может иметь ограничения в отношении того, какие запросы могут быть выполнены. В случае, когда нужно выполнить сложный запрос или использовать специфичные функции базы данных, может потребоваться написание SQL-запроса напрямую.

**Плюсы Django**

Принцип «Всё включено»

Фраза «всё включено» означает, что большинство инструментов для создания приложения — часть фреймворка, а не поставляются в виде отдельных библиотек.

Django содержит огромное количество функциональности для решения большинства задач веб-разработки. Вот некоторые из высокоуровневых возможностей Django:

**ORM**

* Миграции базы данных
* Аутентификация пользователя
* Панель администратора
* Формы
* Стандартизированная структура

Django как фреймворк задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность.

Благодаря одинаковой для всех проектов структуре гораздо проще найти уже готовые решения или получить помощь от сообщества. Огромное количество увлеченных разработчиков поможет справиться с любой задачей гораздо быстрее.

**Недостатки** **Django**:

Django ORM сегодня (судя по данным в интернете) значительно уступает последней SQLAlchemy.

Django ORM основан на шаблоне Active Record, который хуже, чем шаблон Unit of Work, используемый в SQLAlchemy.   
Active Record – реализация активной записи, в которой один экземпляр объекта сопоставляется с каждой строкой базы данных, и данные легко доступны из базы данных. Здесь нет необходимости заранее настраивать схему базы данных, и они могут легко использоваться пользователями, поскольку основная идея в Django заключается в том, что он может понять структуру напрямую, просто взглянув на схему базы данных.

*Django развивается медленно*

Django является большим и монолитным фреймворком. Это позволяет сообществу разрабатывать сотни универсальных модулей и приложений, но снижает скорость разработки самого Django. Кроме того, фреймворк должен поддерживать обратную совместимость, поэтому он развивается относительно медленно.

**Плюсы SQLAlchemy**

*1) Реализация уровня доступа к данным*

SQLAlchemy использует реализацию Data Mapper, которая действует как средний уровень между вашим приложением и базой данных и передает данные между ними, сохраняя при этом их соединение независимо друг от друга. Это обеспечивает большую гибкость между двумя уровнями, а также более эффективное использование базы данных.

*2) Лучше со сложными запросами*

С точки зрения обработки сложных запросов SQLAlchemy имеет преимущество, поскольку он намного лучше взаимодействует с базой данных и, как результат, его можно использовать для написания сложных запросов без необходимости возвращаться к необработанному SQL.

*3) Поддержка сообществ и баз данных*

SQLAlchemy имеет гораздо большее сообщество вместе с совершенно потрясающей документацией, которая свидетельствует о том, что члены сообщества вкладывают в это свое время.

Резюмируем всё вышесказанное об SQLAlchemy:

* SQLAlchemy — это библиотека для Python, которая умеет общаться с базами данных прямо в Python-коде.
* Главное её достоинство — ORM. С помощью него можно работать с базами данных, как будто это обычные объекты в языке Python.
* Библиотека поддерживает много популярных диалектов SQL — например, MySQL, SQLite и PostgreSQL.
* SQLAlchemy умеет выполнять все запросы, доступные в SQL. Поэтому вы не заметите отличий от самого SQL, когда перейдёте на неё.
* В библиотеке есть привычные запросы: select, update, delete, filter, where и другие.

**Недостатки SQLAlchemy**:

более сложная документация в обучении.

**Tortoise ORM**:

Молодая библиотека, созданная примерно в 2018 году.

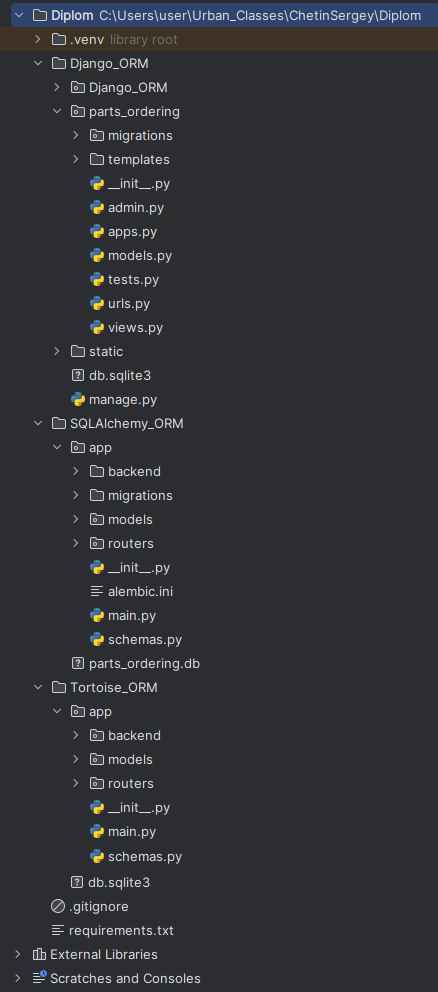
Tortoise ORM крайне знаком всем, кто до этого работал с Django ORM. Дело в том, что разработчики создали свою ORM, вдохновляясь Django ORM.   
Однако Django ORM идет только в пачке с Django и, хотя в Django добавляются возможности для асинхронной работы, это все еще остается не тем инструментом, который можно выбрать, думая об асинхронности.

Тестовый фреймворк использует существующий фреймворк Python Unittest, требует только его initializer() и finalizer() вызывается для настройки и удаления тестовых баз данных.

Приложение ORM Init полностью настраивается на основе предоставленных параметров

* Компонуемые модели, вдохновленные Django;
* Поддерживает отношения, такие как ForeignKeyField и ManyToManyField;
* Поддерживает множество стандартных полей;
* Комплексный формат API запросов;
* Транзакции.

# Приложение 1. Пример файловой структура приложения



# Приложение 2. Список основных библиотек, используемых в проекте

Django==5.1.1  
django-json-widget==2.0.1  
fastapi==0.114.2  
uvicorn==0.30.6  
Jinja2==3.1.4  
pydantic==2.9.0  
SQLAlchemy==2.0.34  
SQLAlchemy-Utils==0.41.2  
tortoise-orm==0.21.6  
tortoise~=0.1.1