**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 студент группы  Python-разработчик 27.05: |
| Волчек Михаил Валерьевич |
|  |
|  |

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

Сравнение производительности и удобства использования различных ORM-библиотек, в том числе Django ORM, SQLAlchemy и Tortoise ORM, нужно для выбора подходящего решения для конкретных задач разработки.

Производительность можно оценить по таким метрикам, как скорость обработки запроса и время загрузки тестовых данных. Это поможет выбрать наиболее эффективное решение для конкретных условий проекта.

Удобство использования можно сравнить по таким параметрам, как простота настройки и освоения, синтаксис, наличие современных возможностей, например асинхронности и валидации данных. Например, Tortoise ORM считается более простым и понятным инструментом для работы с базами данных в небольших асинхронных приложениях.

Таким образом, сравнение позволяет выбрать ORM, которая лучше соответствует требованиям и задачам разработки.

**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».

# 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.

## 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.

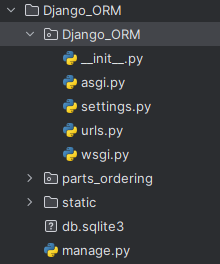
Работа в Джанго начинается с его установки, можно в консоли написать:



Далее создаётся проект следующей командой и названием проекта:



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

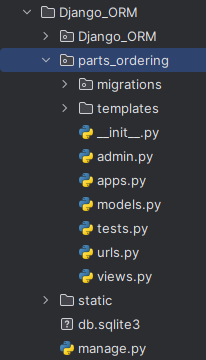


* manage.py — исполняет различные команды, в частности запускает приложение;
* \_\_ init.py \_\_ — стандартный для Python файл, придающий папке, в которой он располагается, свойства модуля;
* settings.py — отвечает за настройки проекта;
* urls.py — включает шаблоны URL-адресов, чем задает систему маршрутизации;
* 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.

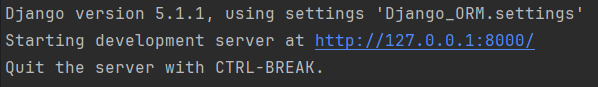
Чтобы запустить выполнение проекта, нужно перейти в консоли к его папке



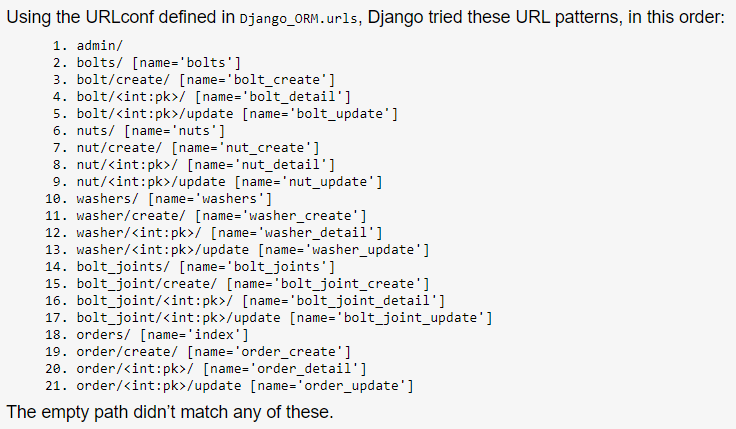
и запустить сервер командой:



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.

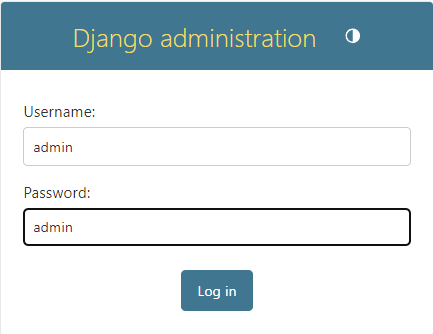


По указанному адресу в браузере появится следующее окно:



где указаны возможные адреса, используемые в приложении.

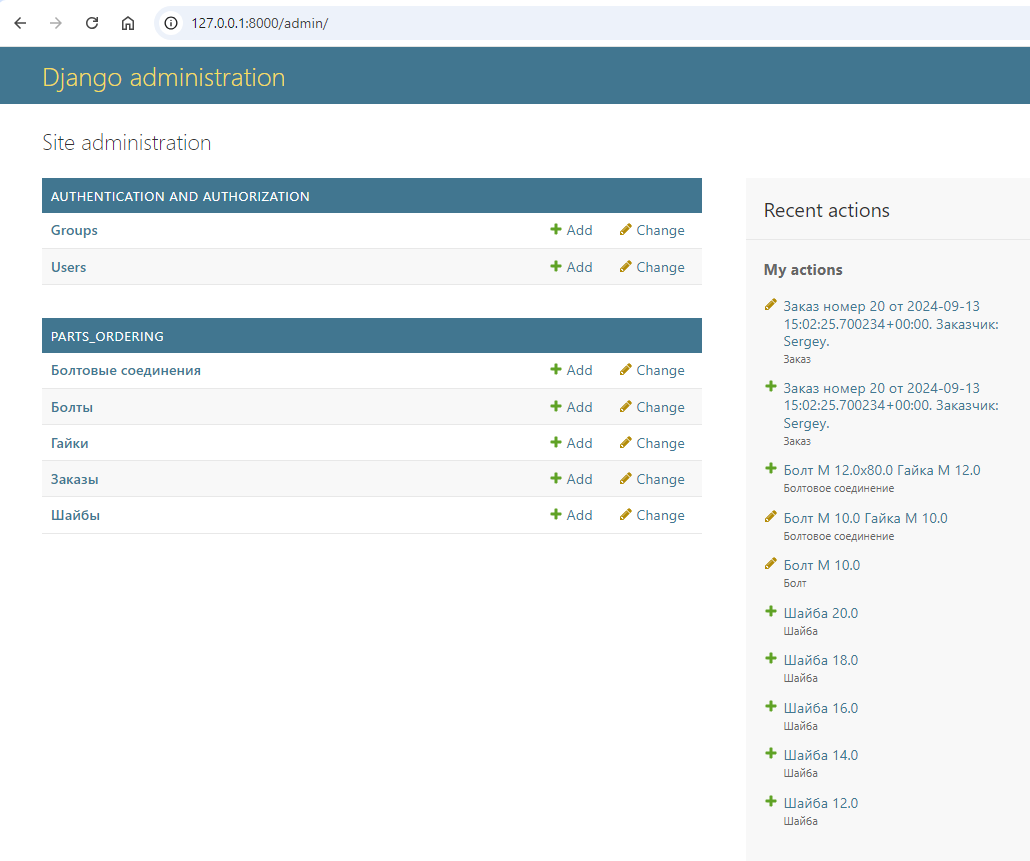
Пройдя по первому адресу, можно будет попасть на страницу регистрации администратора. По умолчанию я сделал



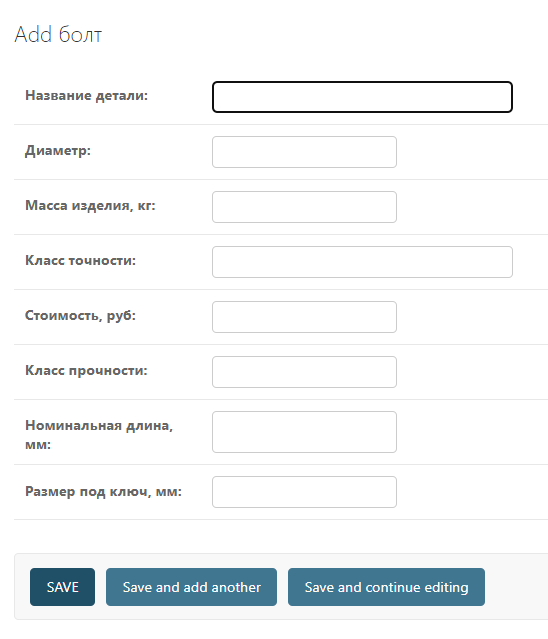
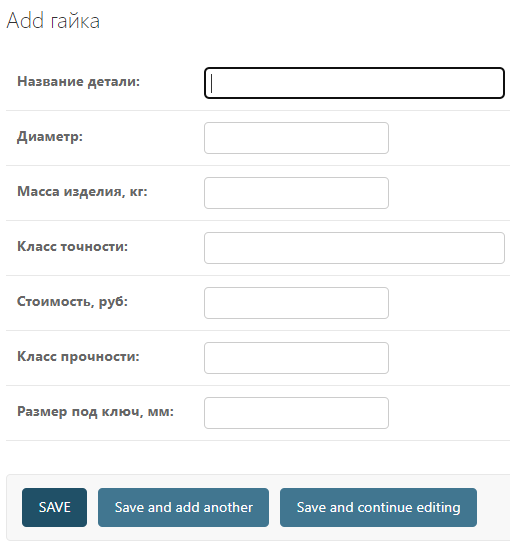
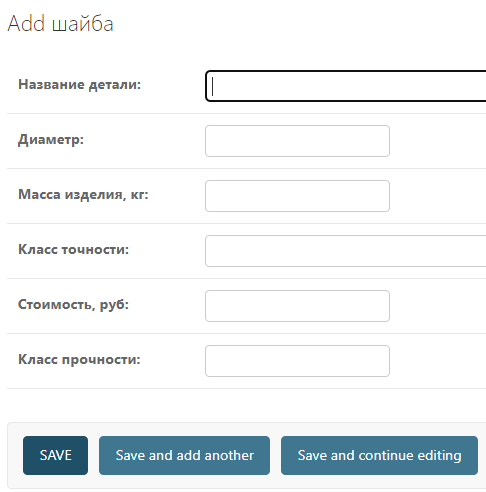
При необходимости можно создать своего суперюзера, командой в консоли:



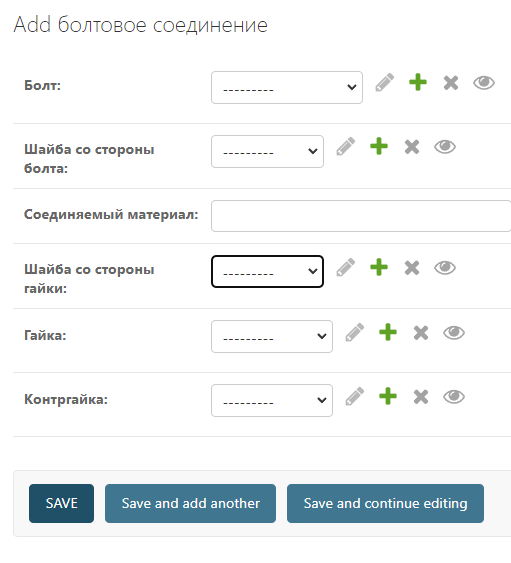
Итак, попадая в приложение, мы видим панель администратора, для создания тех самых интересующих нас изделий (болтов, гаек и шайб) и далее можно собрать их в болтовое соединение и сделать заказ.



Формы создания изделий примерно аналогичные друг другу:

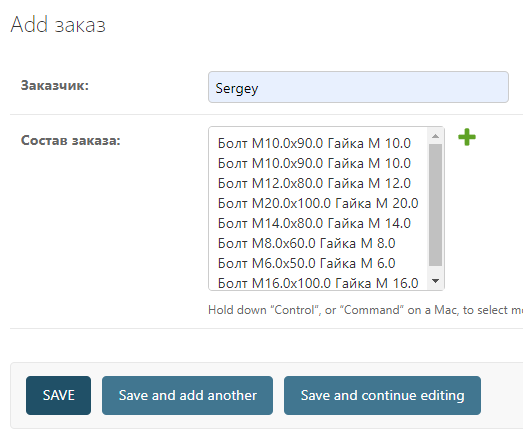
  

А вот создание болтового соединения выглядит следующим образом:



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.

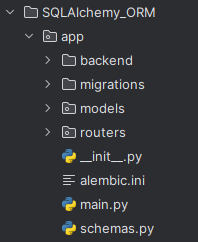
Всё благодаря тому, что библиотека состоит из двух компонентов — SQLAlchemy Core и SQLAlchemy ORM.

* 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:

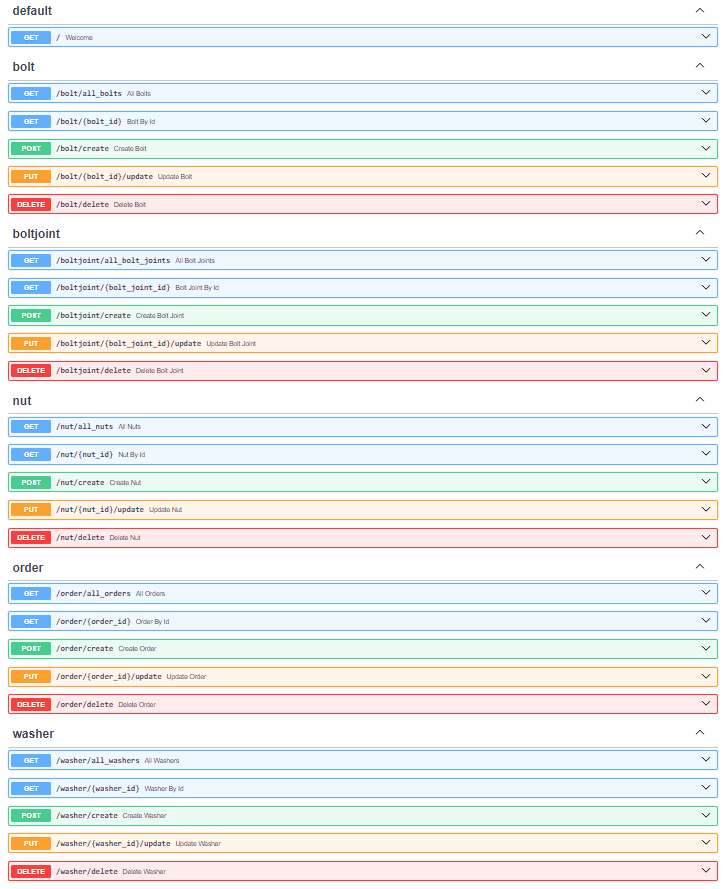


А затем запускаем команду



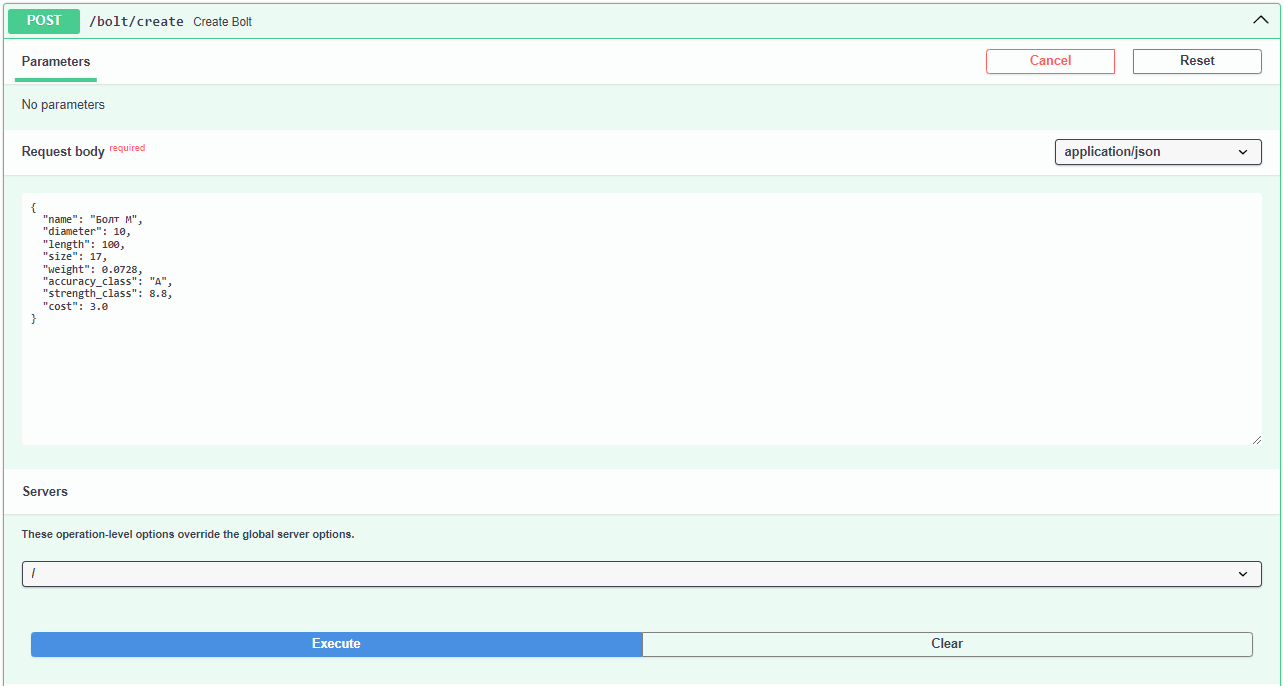
Далее программа запускает создание таблиц, если их не было и через фреймворк FastAPI можно перейти в браузере по адресу:

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



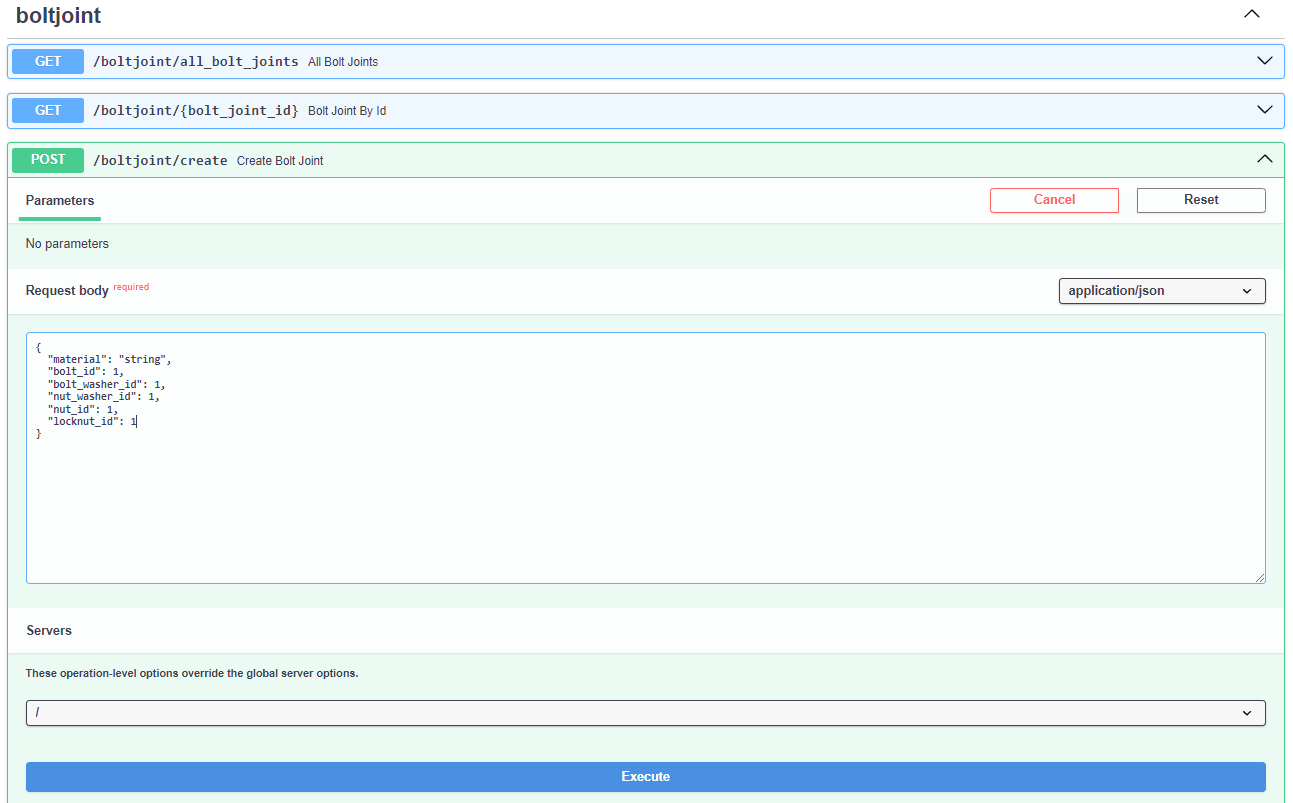
Чтобы создать болт нужно перейти в соответствующий раздел и нажать кнопку

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

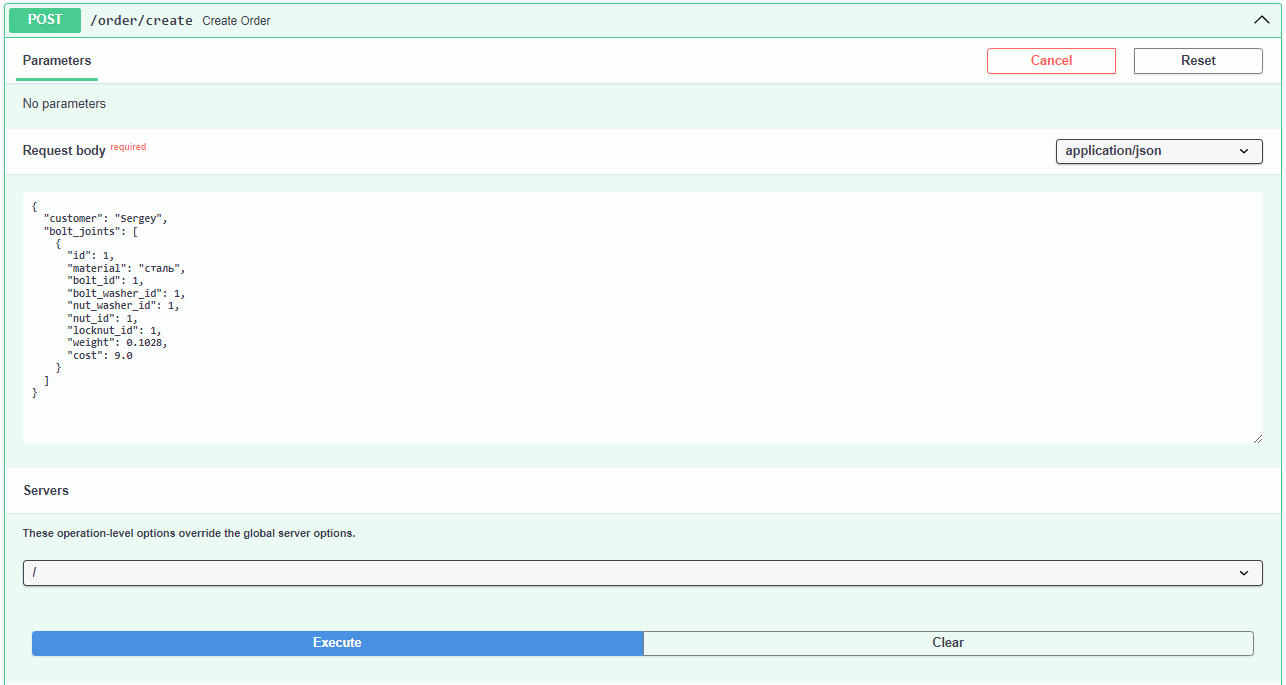


- аналогично в гайках и шайбах…

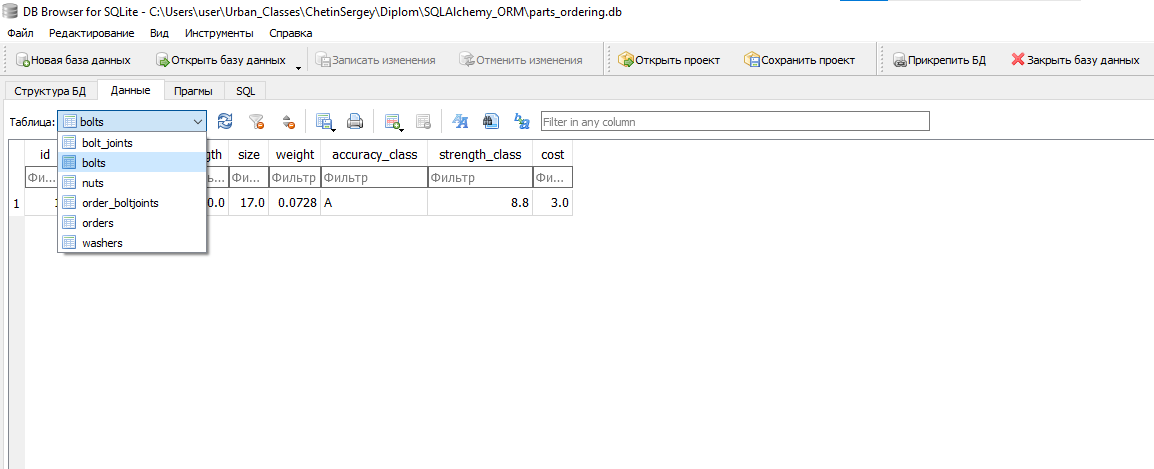
В болтовом соединении нужно просто проставить номера id и оно готово:



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

**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 фреймворка Django:**

* Простота. Создать простой сайт с помощью программирования на Django можно максимально быстро, обладая минимальными знаниями в Python.
* Гибкость. Базовые функции фреймворка можно изменять и настраивать под потребности конкретного проекта.
* Стандартизация. Разобраться в проекте на Django просто, если уже было с ним дело.
* Безопасность. В фреймворк уже встроены методы защиты и middleware, обеспечивающие безопасность данных пользователей и всего веб-приложения.
* Удобное администрирование. Благодаря графическому административному интерфейсу проще управлять БД и содержимым сайта.
* Популярность. У Django обширная документация и огромное сообщество лояльных разработчиков.
* Поддержка разных баз данных. Благодаря Django можно легко интегрировать веб-сервис с любой популярной БД без большого количества сложных настроек.

**Минусы фреймворка Django:**

* Некоторая ограниченность. Есть вещи, которые в Django заданы строго, и изменить их нельзя.
* Требовательность к ресурсам. Из-за обилия встроенных инструментов Django требует больше мощностей для запуска, чем нативный код на Python.
* Слишком большая сложность для простых проектов. Маленький одностраничный сайт может быть быстрее собрать с нуля, чем писать полную структуру проекта и программировать на Django.
* Избыточность. У Django есть все инструменты для создания высоконагруженных приложений, но для небольших сайтов их зачастую слишком много.
* Устаревший ORM. Ключевой минус компонента — отсутствие поддержки SQLAlchemy, которая является основным инструментом работы с базами данных на языке Python.
* Отсутствие поддержки WebSocket. Этот протокол позволяет браузеру и серверу обмениваться информацией в режиме реального времени, но фреймворк не поддерживает данную функцию.

**Плюсы SQLAlchemy**

* абстрагирование взаимодействия базы данных с помощью ORM;
* поддержка нескольких баз данных, например MySQL, PostgreSQL, SQLite;
* большое сообщество с обширной документацией;
* возможность оптимизировать производительность через контроль за выполнением запросов;
* поддержка кэширования на уровне объектов, что позволяет кэшировать часто используемые объекты базы данных в памяти.

**минусы SQLAlchemy:**

* более крутая кривая обучения для новичков благодаря концепциям ORM;
* небольшие накладные расходы на производительность.

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.

**Tortoise ORM**:

Молодая библиотека, созданная примерно в 2018 году.

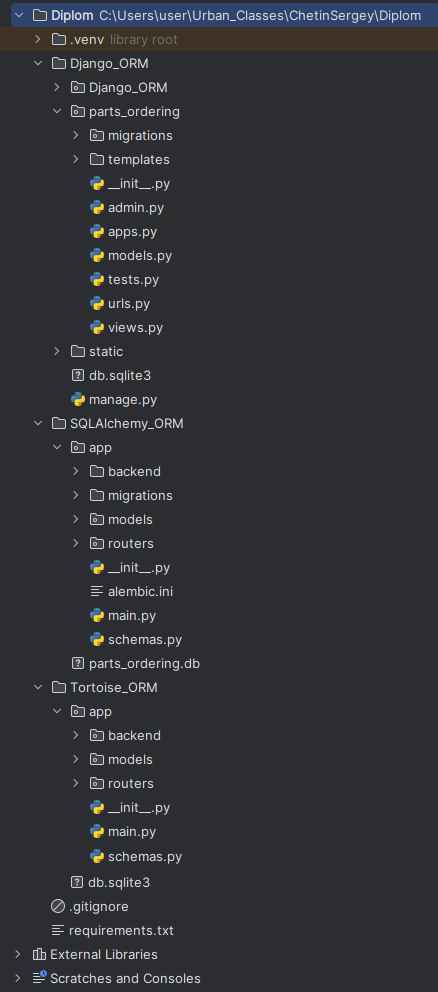
**Плюсы Tortoise ORM:**

* Асинхронность. 13 Библиотека подходит для современных асинхронных фреймворков, таких как FastAPI и Sanic.
* Поддержка нескольких баз данных. В том числе SQLite, PostgreSQL и MySQL.
* Интуитивный API. Он похож на Django ORM, что упрощает переход для разработчиков Django.
* Мощный конструктор запросов. Библиотека поддерживает сложные запросы.

**Минусы Tortoise ORM:**

* Относительная новизна. По сравнению с другими ORM в ней может быть меньше ресурсов и поддержки сообщества.
* Ограниченная поддержка миграции. По сравнению с более устоявшимися ORM.
* Более сложная кривая обучения для разработчиков, не знакомых с концепциями asyncio.
* Недостаточно подробная документация для продвинутых случаев использования.

# 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