**ДИПЛОМНАЯ РАБОТА**

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

**Выполнил: Супранович Алексей**

**2024 г.**

**Содержание**

1. Введение ……………………………………………………. 2
   1. Актуальность темы ………………………………….. 2
   2. Цели и задачи исследования ……………………...… 2
   3. Основные понятия и определения ………………….. 3
2. Обзор исследуемых ОРМ библиотек …………………...… 4
   1. Django ORM ………………………………………….. 4
   2. SQLAlchemy ORM …………………………………… 4
   3. Tortoise ORM …………………………………………. 5
3. Практическая реализация приложений …………………… 7
   1. Описание задачи ……………………………………... 7
   2. Реализация приложения на Django ORM …………... 8
   3. Реализация приложения на SQLAlchemy ORM …... 14
   4. Реализация приложения на Tortoise ORM ………… 23
4. Сравнение и анализ результатов …………………………. 28
   1. Django ORM ………………………………………… 28
   2. SQLAlchemy ORM ………………………………….. 29
   3. Tortoise ORM ………………………………………... 30
5. Заключение ………………………………………………… 31
6. **Введение**

Дипломная работа – это серьезное исследование, требующее глубокого анализа и разработки. В рамках этого проекта необходимо рассмотреть три популярных ORM-библиотеки: Django ORM, SQLAlchemy и Tortoise ORM. Целью является сравнение их производительности и удобства использования при разработке приложений для работы с базами данных.

* 1. **Актуальность темы**

ORM (Object-Relational Mapping) представляет собой инструмент, который позволяет разработчикам абстрагироваться от низкоуровневых операций с базой данных, предоставляя возможность работать с данными через объекты языка программирования. Если своими словами, то они действуют как посредники между объектами Python и таблицами базы данных, тем самым упрощая наше взаимодействие с данными. Это значительно упрощает разработку приложений, особенно в случае сложных бизнес-логик и больших объемов данных. Однако выбор подходящей ORM-библиотеки может существенно повлиять на производительность и удобство разработки.

* 1. **Цели и задачи исследования**

Цель данного дипломного проекта заключается в сравнительном анализе производительности и удобства использования трех популярных ORM-библиотек: Django ORM, SQLAlchemy и Tortoise ORM. Для достижения этой цели необходимо решить следующие задачи:

* Изучить теоретические основы ORM и исследовать архитектурные особенности выбранных библиотек.
* Разработать три аналогичных приложения для работы с базой данных с использованием каждой из исследуемых ORM-библиотек.
* Проанализировать удобство использования каждой библиотеки на основе личного опыта разработки и отзывов разработчиков.
* Сделать выводы о преимуществах и недостатках каждой библиотеки и дать рекомендации по выбору ORM в зависимости от конкретных требований проекта.
  1. **Основные понятия и определения**

Основные компоненты ORM включают:

* Модели: представляют таблицы базы данных в виде классов языка программирования.
* Отношения: определяют связи между различными моделями (например, один ко многим, многие ко многим).
* Запросы: позволяют выполнять операции чтения, записи, обновления и удаления данных.
* Миграции: автоматизируют процесс изменения структуры базы данных.

1. **Обзор исследуемых ORM-библиотек**

**2.1. Django**

Django ORM является частью фреймворка Django и предоставляет высокоуровневый интерфейс для работы с реляционными базами данных. Он следует шаблону Active Record, где каждая модель Django представляет таблицу базы данных, а экземпляры модели сопоставляются со строками таблицы. Эта тесная интеграция делает Django ORM невероятно удобным для разработчиков, особенно для тех, кто уже знаком с Django. Также фреймворк поддерживает автоматическое создание таблиц, миграцию схем базы данных и многие другие функции.

Основные возможности Django ORM:

* Автоматическая генерация схемы: определите свои модели на Python, а Django ORM займется созданием схемы базы данных (миграциями).
* Декларативные отношения: легко моделируйте отношения между моделями, используя такие поля, как ForeignKey и ManyToManyField .
* Интеграция с интерфейсом администратора: Django ORM легко интегрируется с интерфейсом администратора Django, позволяя легко выполнять операции CRUD (создание, чтение, обновление, удаление).
* Независимый от баз данных (в некоторой степени): хотя изначально он был разработан для PostgreSQL, MySQL и SQLite , дополнительная настройка позволит использовать Django ORM для поддержки других баз данных.
  1. **SQLAlchenemy**

SQLAlchemy: Мощная и гибкая библиотека для работы с базами данных, которая поддерживает как декларативную, так и императивную работу с объектами. Она принимает***шаблон Data Mapper*** , где классы Python сопоставляются с таблицами базы данных, но конфигурация сопоставления более явная. Эта гибкость позволяет разработчикам обрабатывать сложные взаимодействия с базами данных и интегрироваться с различными веб-фреймворками, а не только с Django, позволяет разработчику контролировать каждый аспект взаимодействия с базой данных.

Основные возможности SQLAlchemy:

* Независимая от баз данных: работает со многими типами баз данных, такими как *PostgreSQL, MySQL, SQLite и Oracle* .
* Конфигурация ORM: контролирует, как объекты Python отображаются в таблицах базы данных.
* Декларативный и императивный подход : поддерживает как декларативный (с использованием классов), так и императивный (с использованием функций) стили для определения объектно-реляционных отображений.
* Расширенные запросы: предлагает мощный механизм, позволяющий пользователям легко создавать сложные SQL-запросы.
  1. **Tortoise ORM**

Tortoise ORM – это асинхронный объектно-реляционный преобразователь (ORM), разработанный для работы с базами данных в Python. Он был создан с целью обеспечить удобный и эффективный способ взаимодействия с реляционными базами данных при использовании асинхронного программирования.

Основные возможности Tortoise ORM

* Поддержка асинхронных операций: Tortoise ORM полностью поддерживает работу с асинхронными операциями, что делает его идеальным выбором для приложений, написанных с использованием asyncio.
* Простота использования: Одним из главных преимуществ Tortoise является его простота и интуитивность. Создание моделей и работа с ними очень похожи на Django ORM, что облегчает переход для разработчиков, знакомых с этой библиотекой.
* Широкая поддержка баз данных: Tortoise поддерживает несколько популярных СУБД, включая PostgreSQL, MySQL и SQLite. Это позволяет использовать его в различных проектах без необходимости менять ORM в зависимости от выбранной базы данных.
* Интеграция с другими фреймворками: Tortoise легко интегрируется с популярными веб-фреймворками, такими как FastAPI и Starlette, благодаря своей поддержке асинхронной модели выполнения.

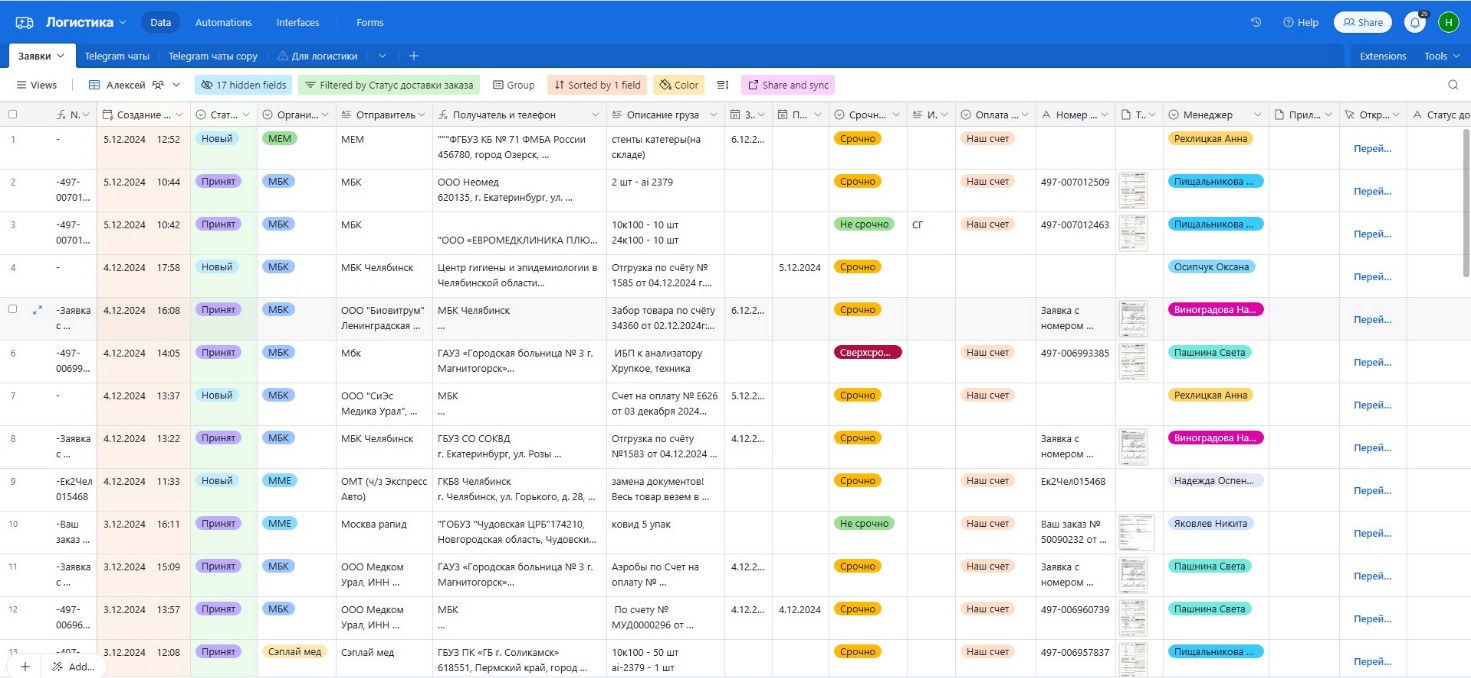
1. **Практическая реализация приложений**

**3.1. Описание задачи**

В практической части данной дипломной работы реализована разработка приложений на трех рассмотренных выше фреймворках.

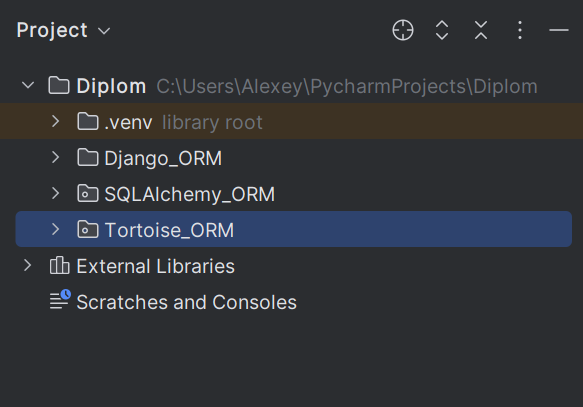
Каждое приложение представляет часть базы данных из CRM Airtable, которую используют в нашей организации, а именно блок «Логистика». В нем реализовано размещение заданий в отдел логистики и получение обратной связи о этапах их выполнения.

Рис. 3.1 CRM Airtable, блок «Логистика»



В базе данных создаются 2 таблицы: Manager – здесь формируется список сотрудников, которые могут создавать задания, и принадлежность сотрудников к отделам. Task – здесь происходит создание самой задачи с использованием необходимой информации: дата создания, плательщик, отправитель, получатель, содержание задачи, срочность выполнения задачи, накладная ТК (в случае необходимости) и менеджер, который создал задачу.

Рис. 3.1 Структура проекта



* 1. **Реализация приложения на Django ORM**

Установка Django.

Рис. 3.2.1 Установка необходимых библиотек для Django



Рис. 3.2.2 Создание нового проекта Django

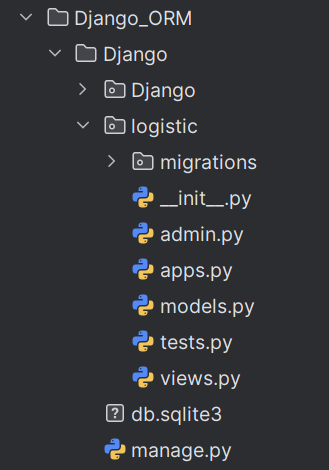


Рис. 3.2.3 Создание нового приложения внутри нашего проекта



После вышеуказанных команд создается определенная структура файлов и папок, необходимых для работы приложения.

Рис. 3.2.4 Структура проекта 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.

Файлы приложения logistic:

* admin.py – где через права администратора регистрируются изделия;
* models.py – где хранятся модели наших изделий;
* views.py – файл, в котором находятся логика работы приложения. Здесь запрашивается информация из модели, которая была создана ранее, и передаётся в шаблон.

Для удобства внесения данных, создаем доступ в панель администратора

Рис. 3.2.5 Создание панели администратора



Создаем миграции моделей в базу данных.

Рис. 3.2.6 Создание миграций в базу данных и применение их





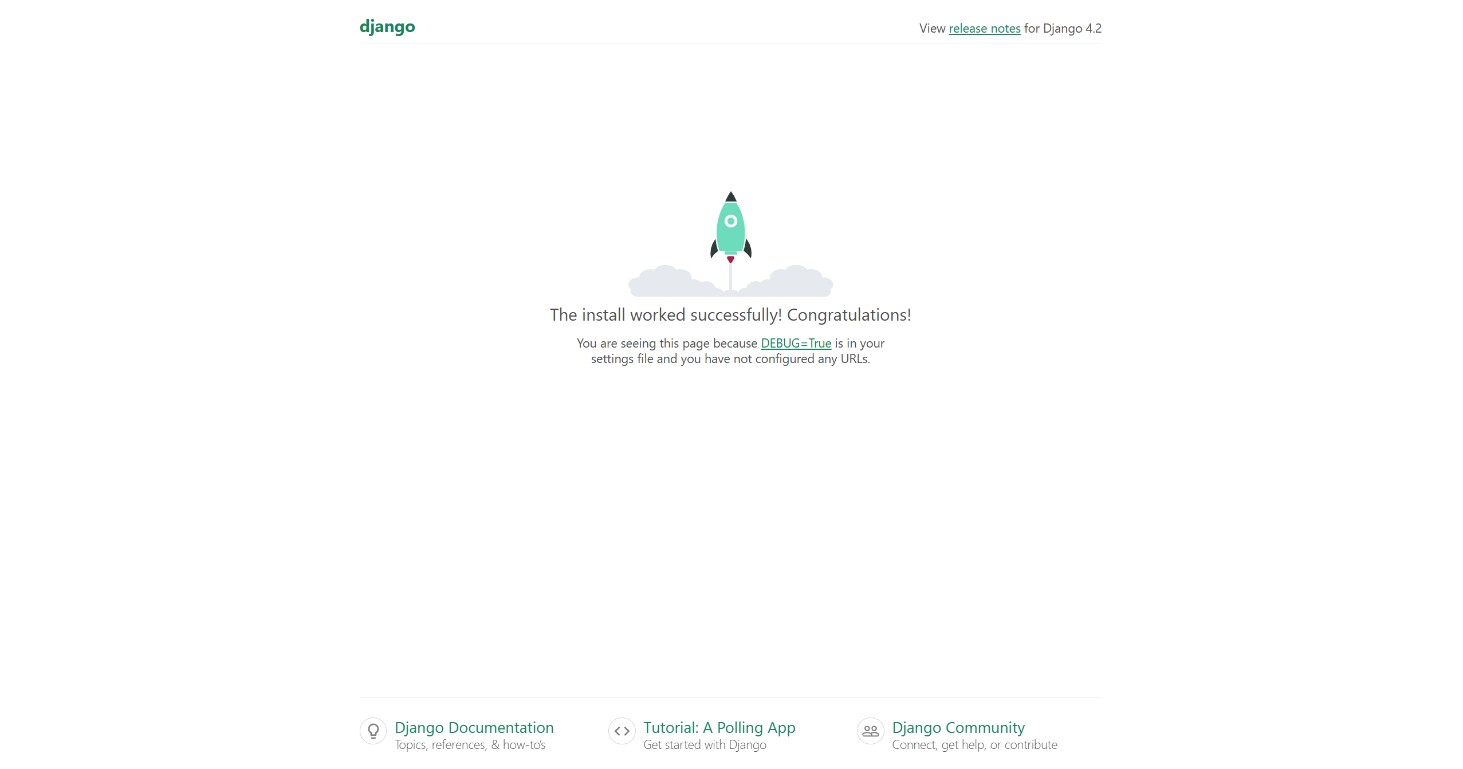
Все необходимое для запуска приложения выполнено. Запуск приложения.

Рис. 3.2.6 Запуск приложения



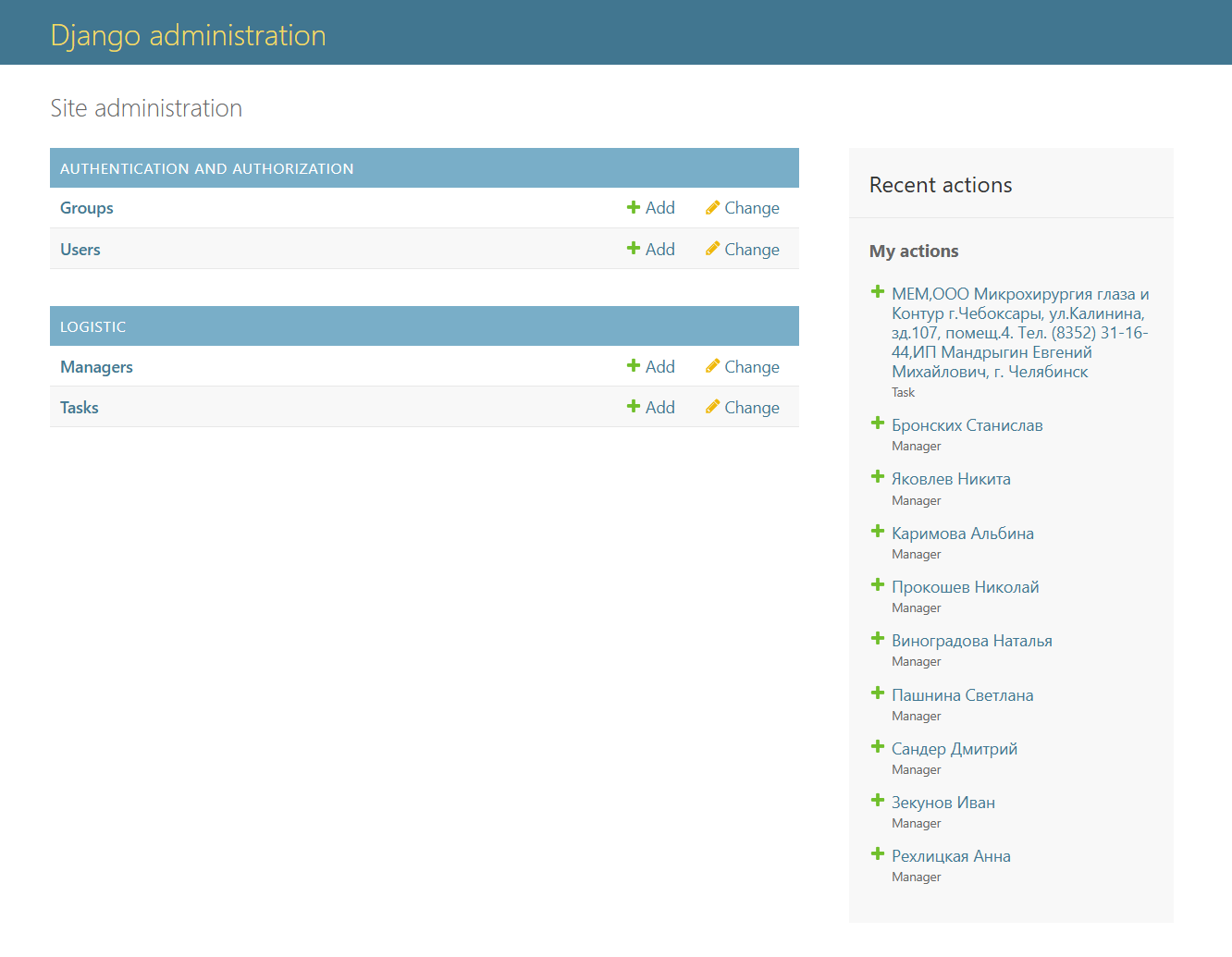
В терминале проходим по ссылке <http://127.0.0.1:8000>. Попадаем на страницу Django.

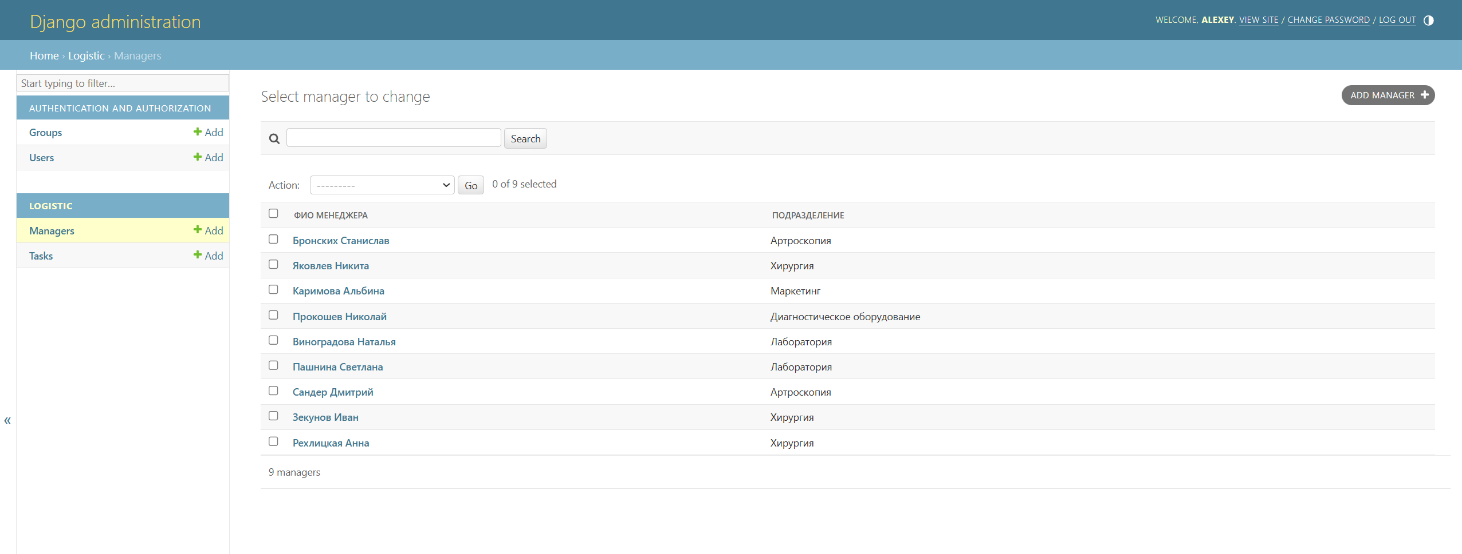
Рис. 3.2.7 Стартовая страница Django

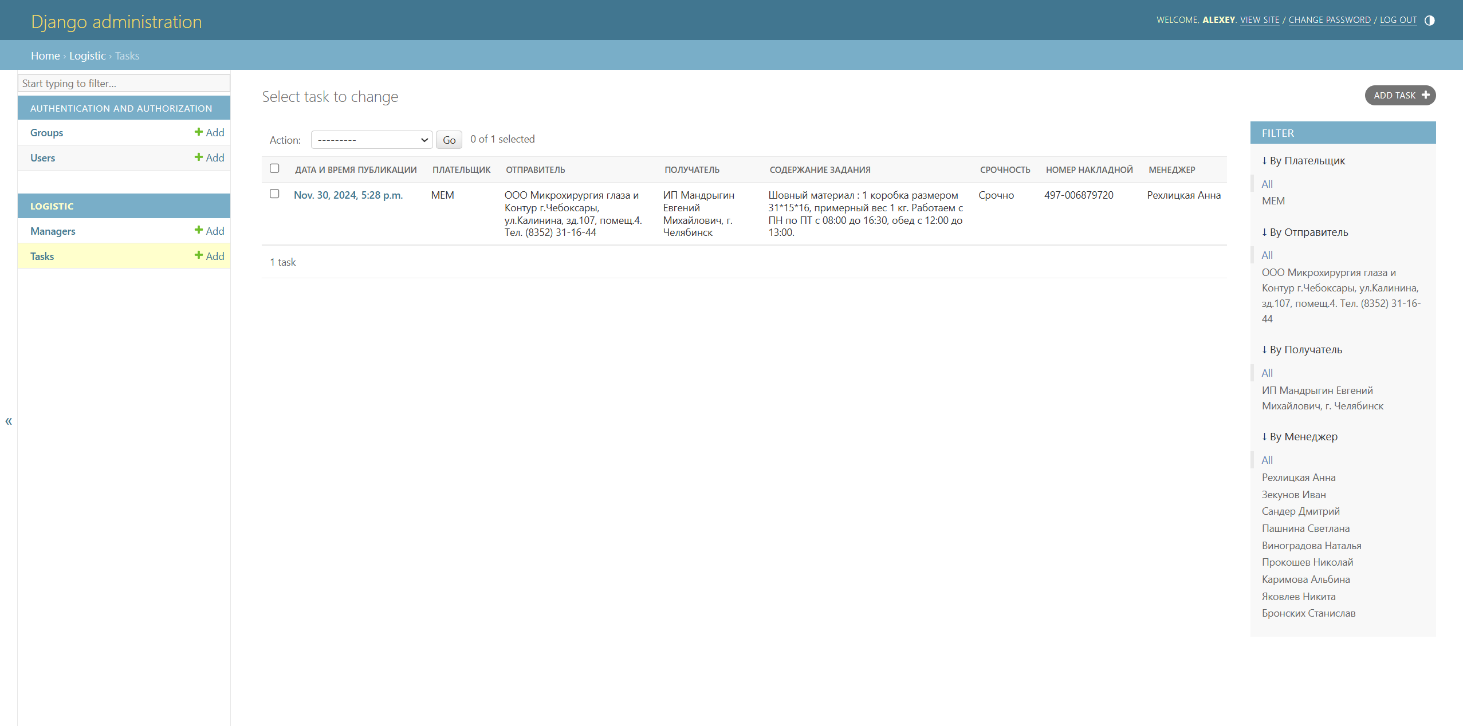


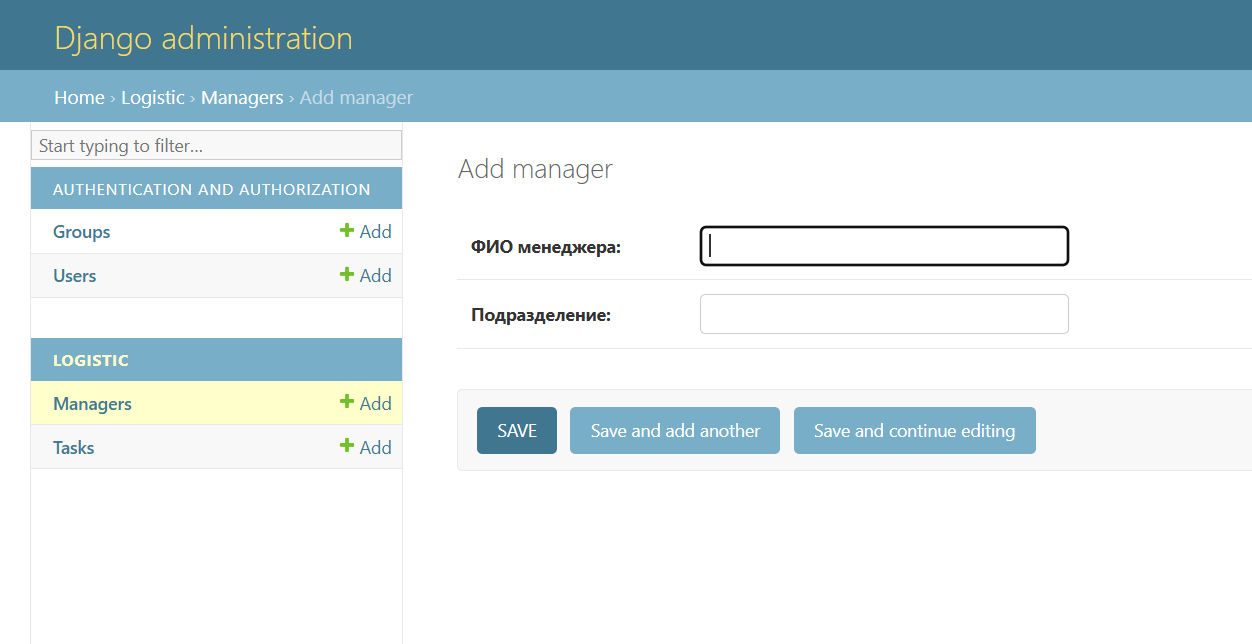
Заходим в панель администратора и создаем записи в соответствующих таблицах.

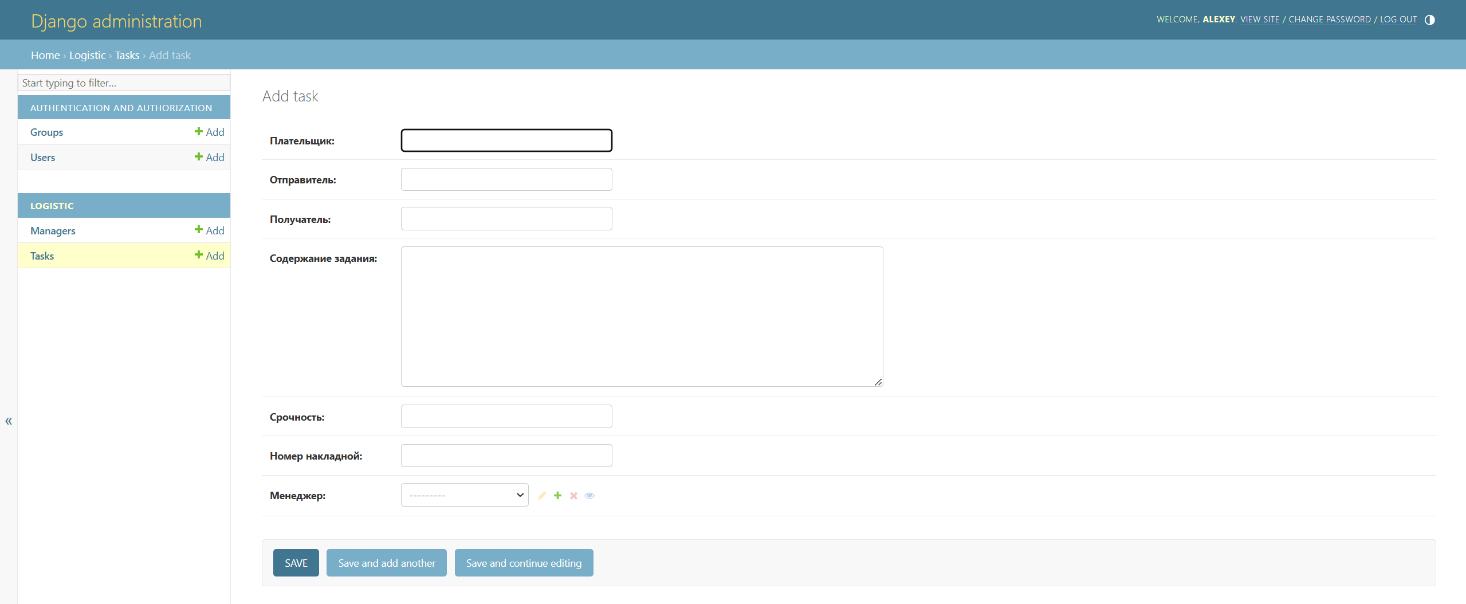
Рис. 3.2.8 Заполнение данными созданных таблиц manager и task











Кроме работы в панели администратора, в Django ORM есть возможность работать и в оболочке shell.

Рис. 3.2.8 Создание объектов с помощью оболочки shell









* 1. **Реализация приложения на SQLAlchemy ORM**

SQLAlchemy ORM работает внутри веб-фреймворка FastAPI.

Рис. 3.3.1 Установка SQLAlchemy ORM и FastAPI





Рис. 3.3.2 Вручную создаем структуру приложения

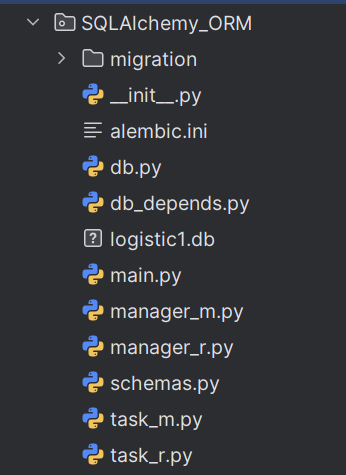
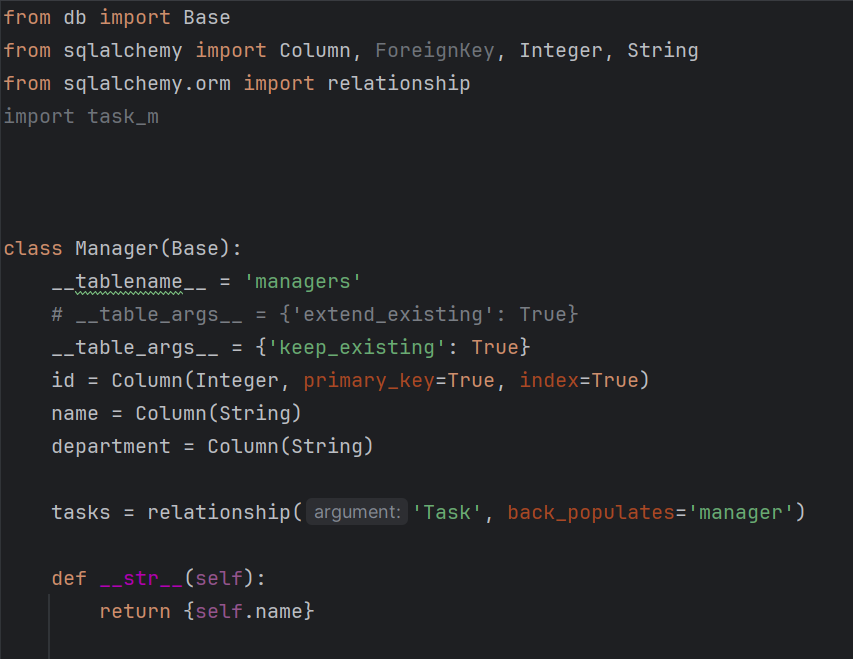
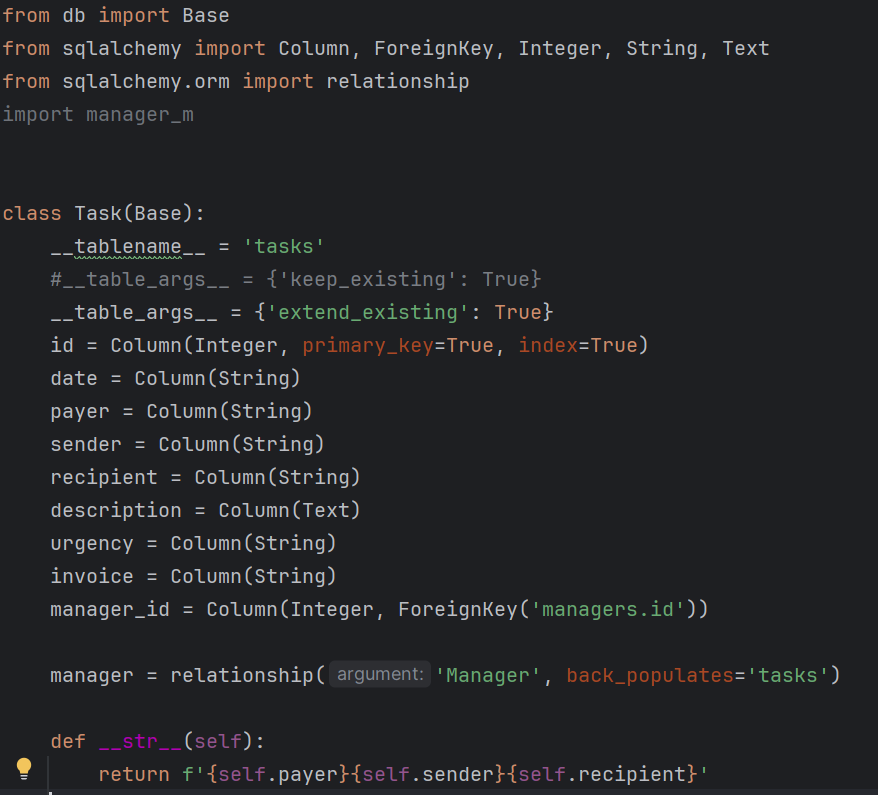


Рис. 3.3.3 Создание моделей для миграции таблиц





Устанавливаем библиотеку alembic для реализации миграций моделей данных в базу данных.

Рис. 3.3.4 Установка библиотеки alembic



С помощью alembic создаем базу данных.

Рис. 3.3.5 Создание базы данных по прописанным моделям







Запускаем приложение и переходим по пути /docs чтобы попасть в Swagger, в котором выполняются все основные CRUD процессы.

Рис. 3.3.6 Вэб-приложение с созданными таблицами

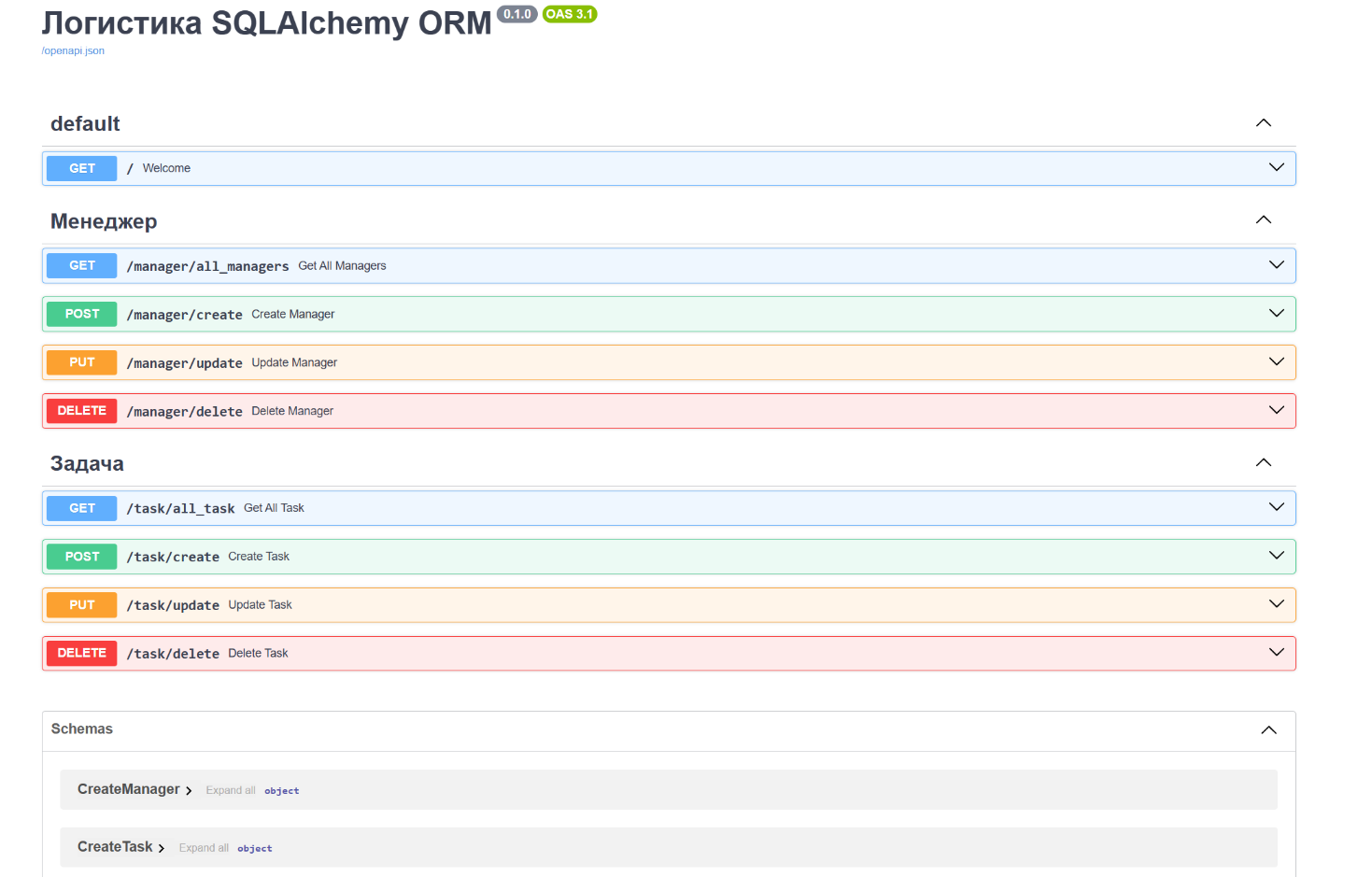
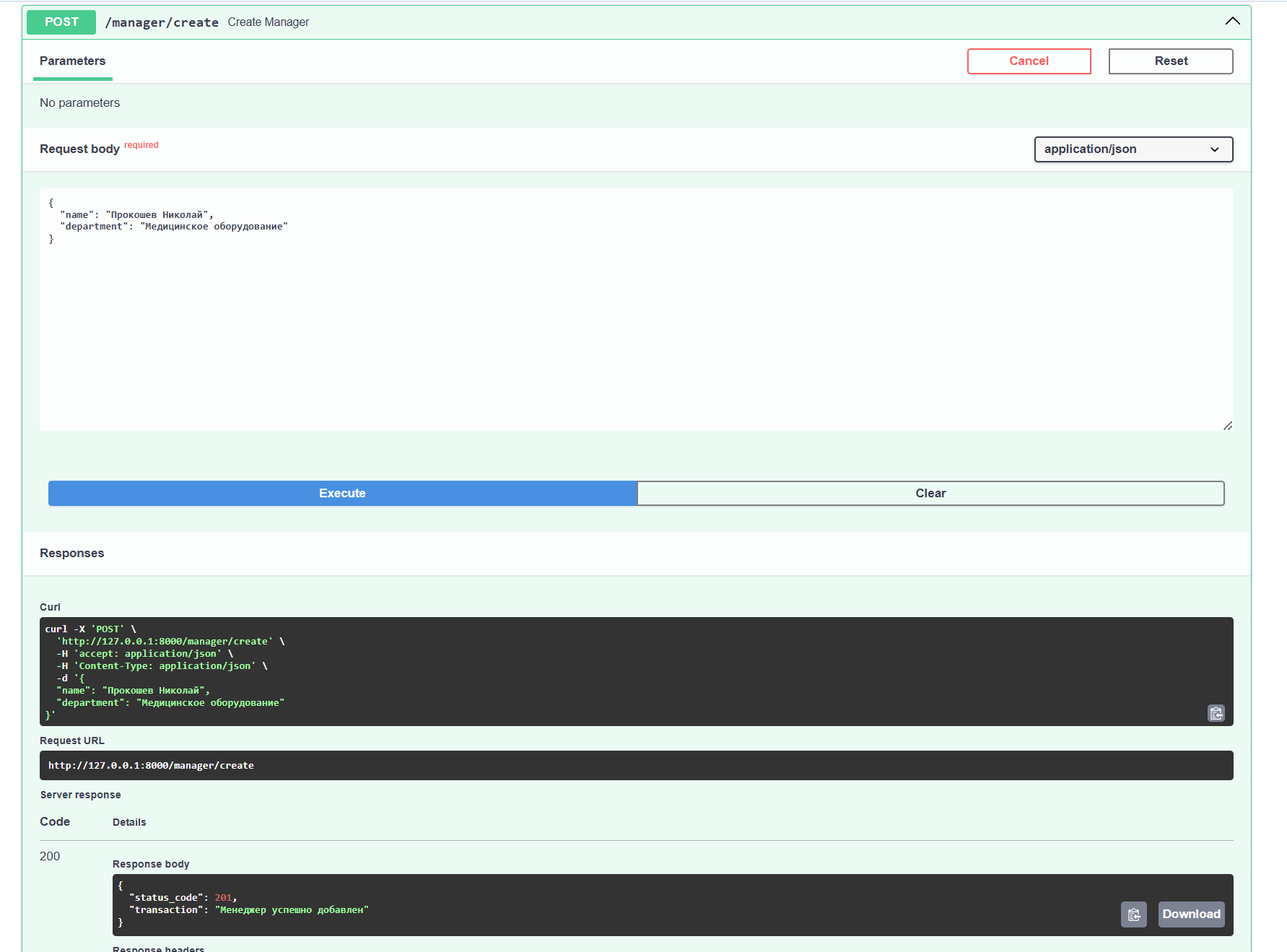


Рис. 3.3.7 Создание записи в таблицу



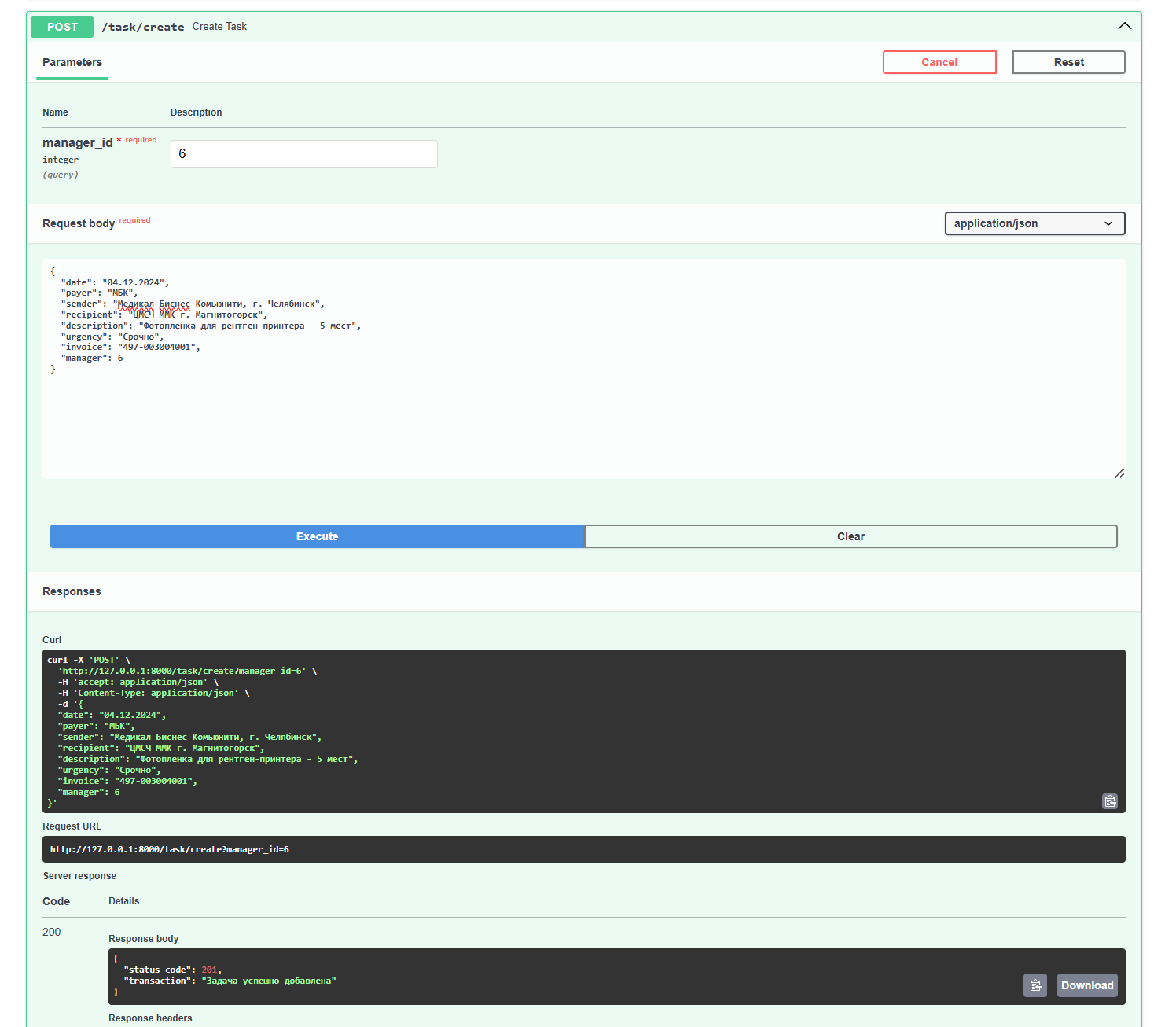
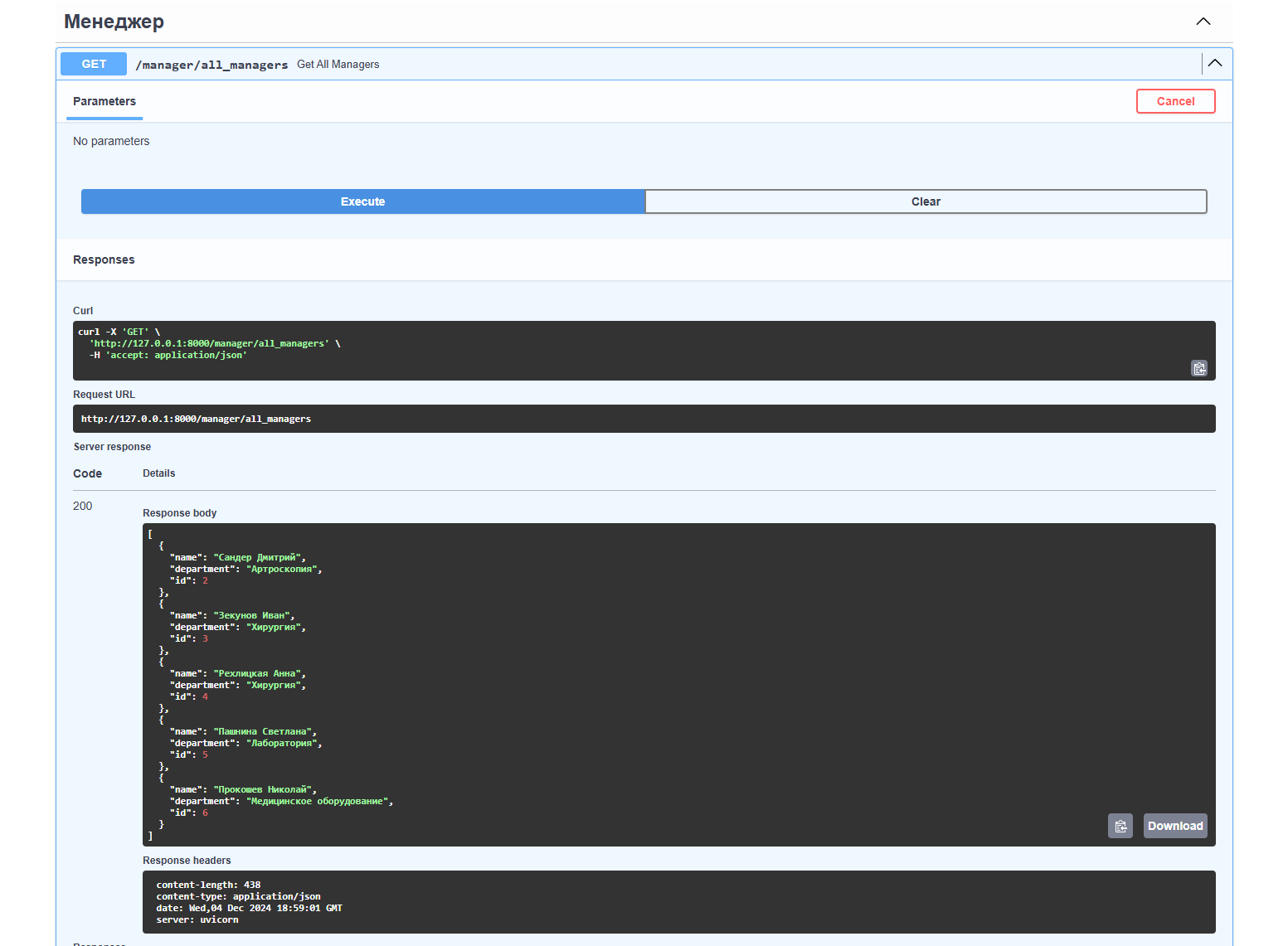


Рис. 3.3.8 Получение всех записей из таблицы



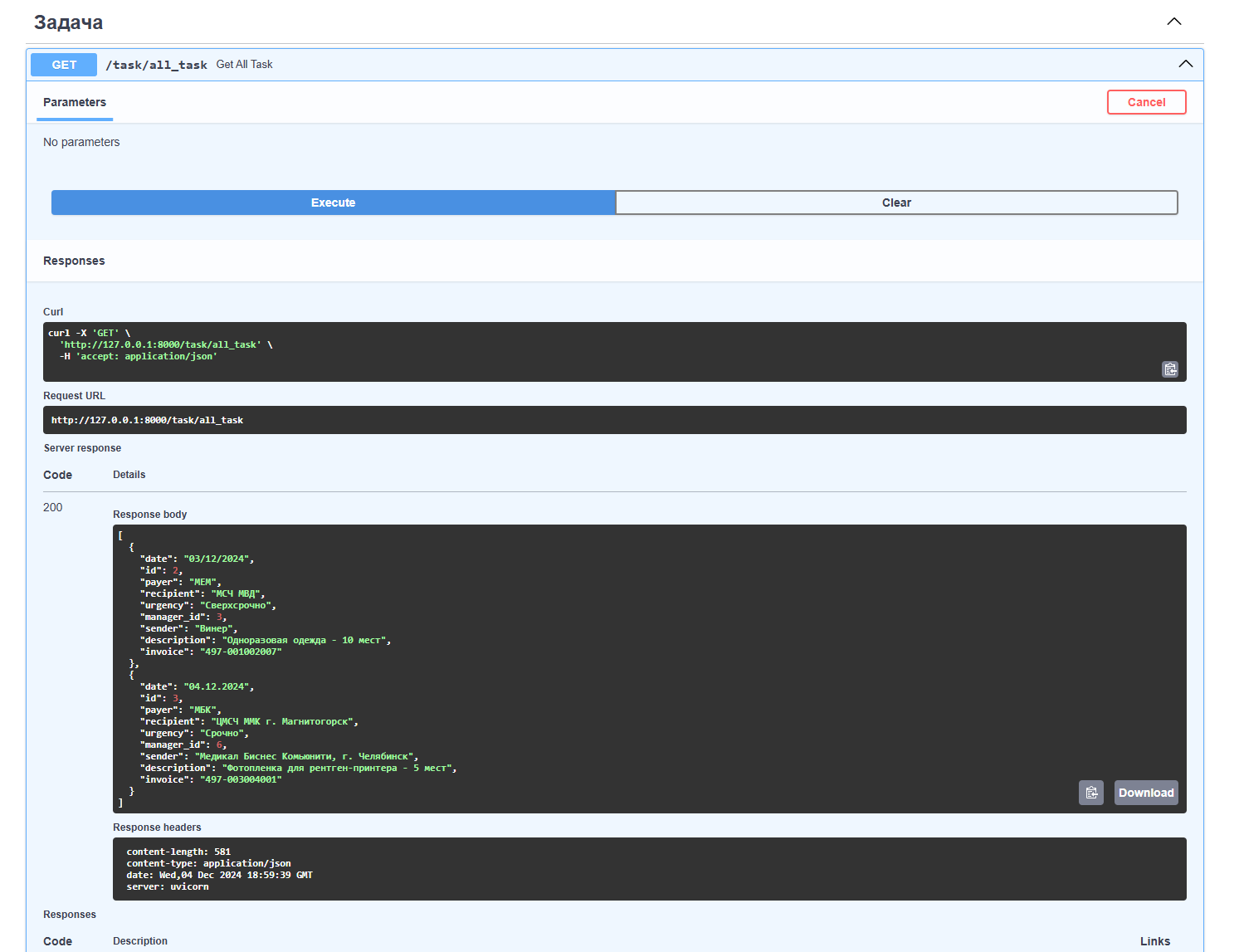
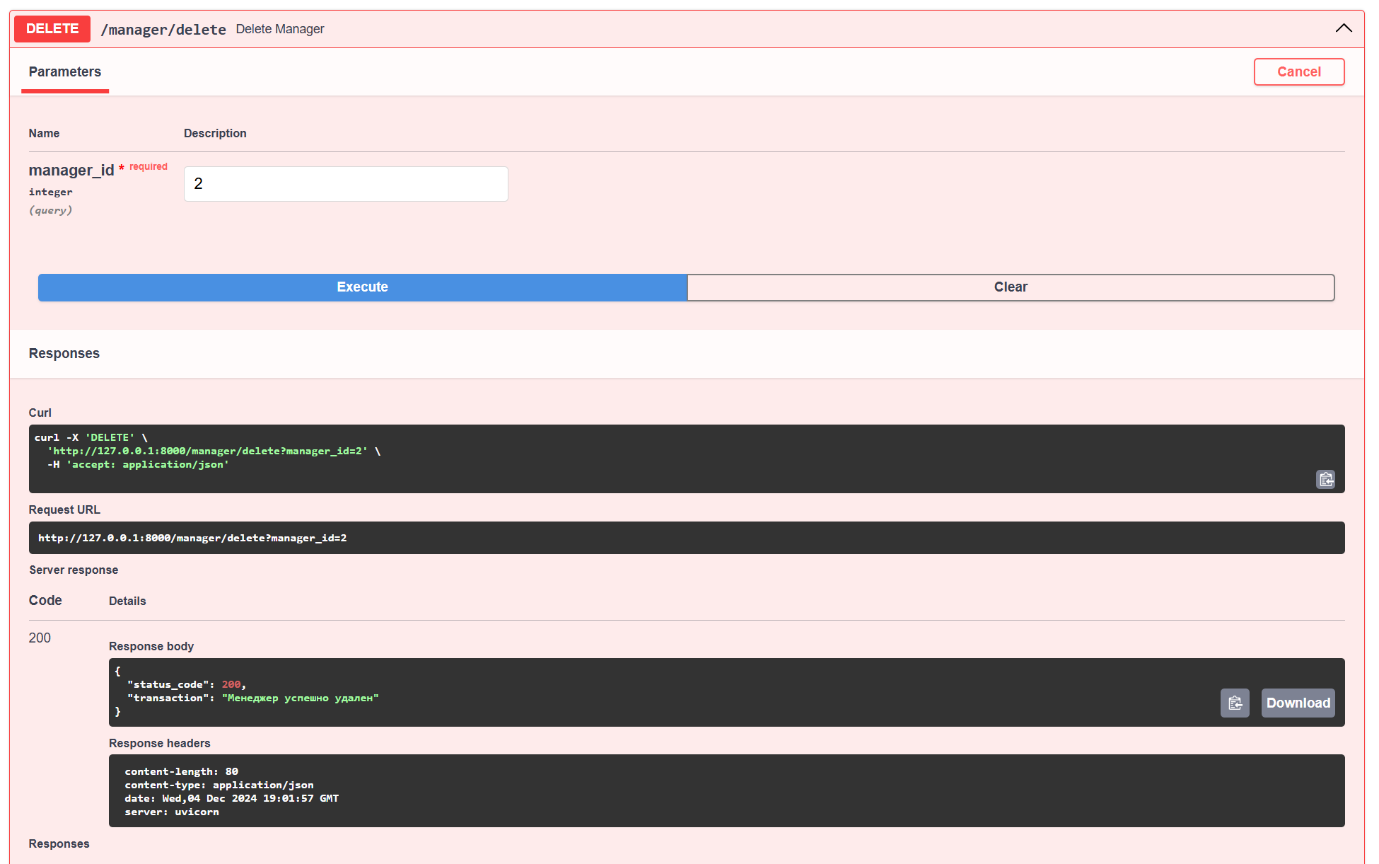


Рис. 3.3.9 Удаление записи из таблицы



* 1. **Реализация приложения на: Tortoise ORM**

Рис. 3.4.1 Установка библиотеки Tortoise ORM.



Также как и для SQLAlchemy ORM вручную создаем структуру приложения.

Рис. 3.4.2 Структура приложения для Tortoise ORM

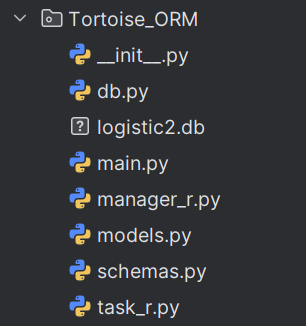


Рис. 3.4.3 Запуск приложения



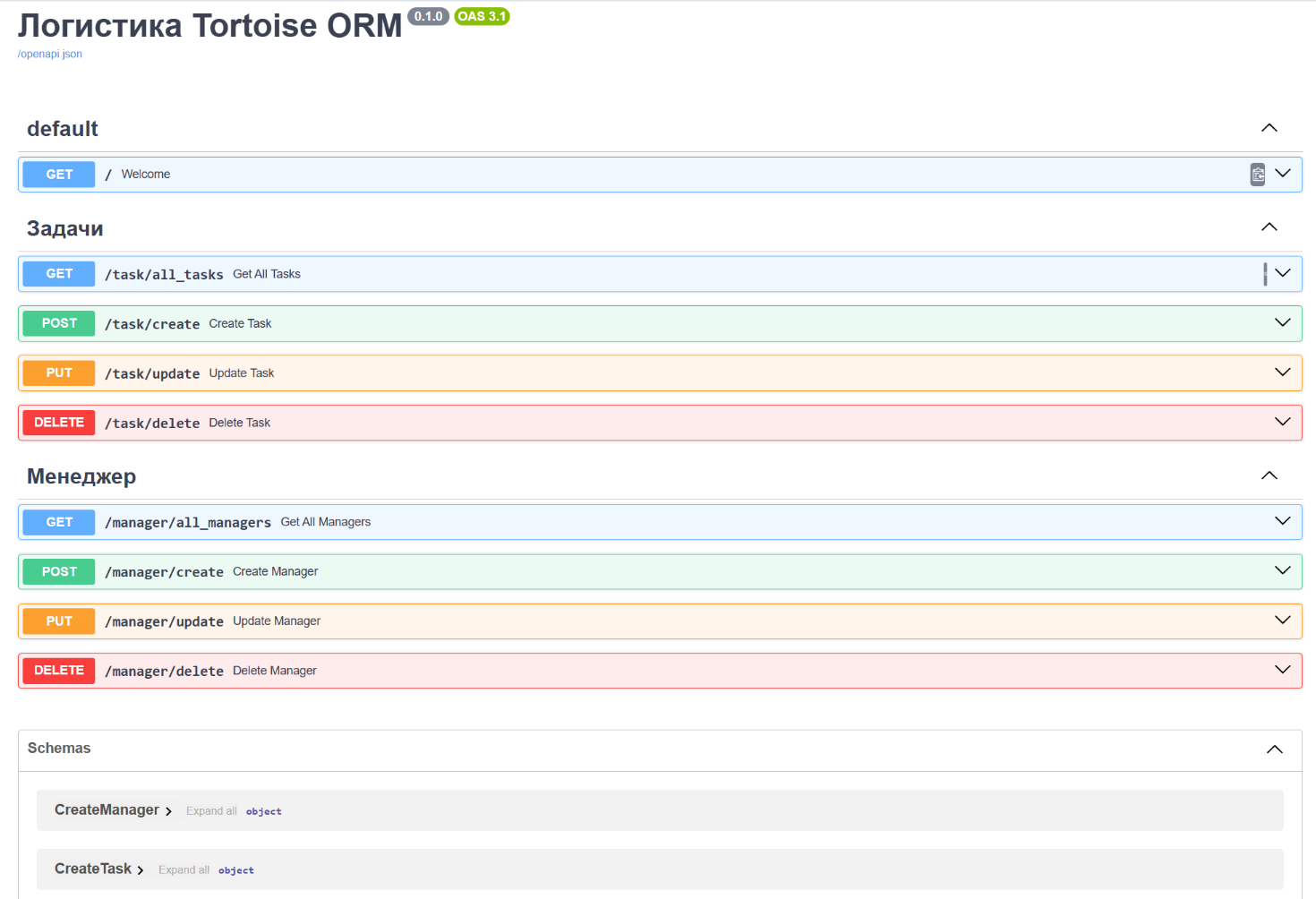
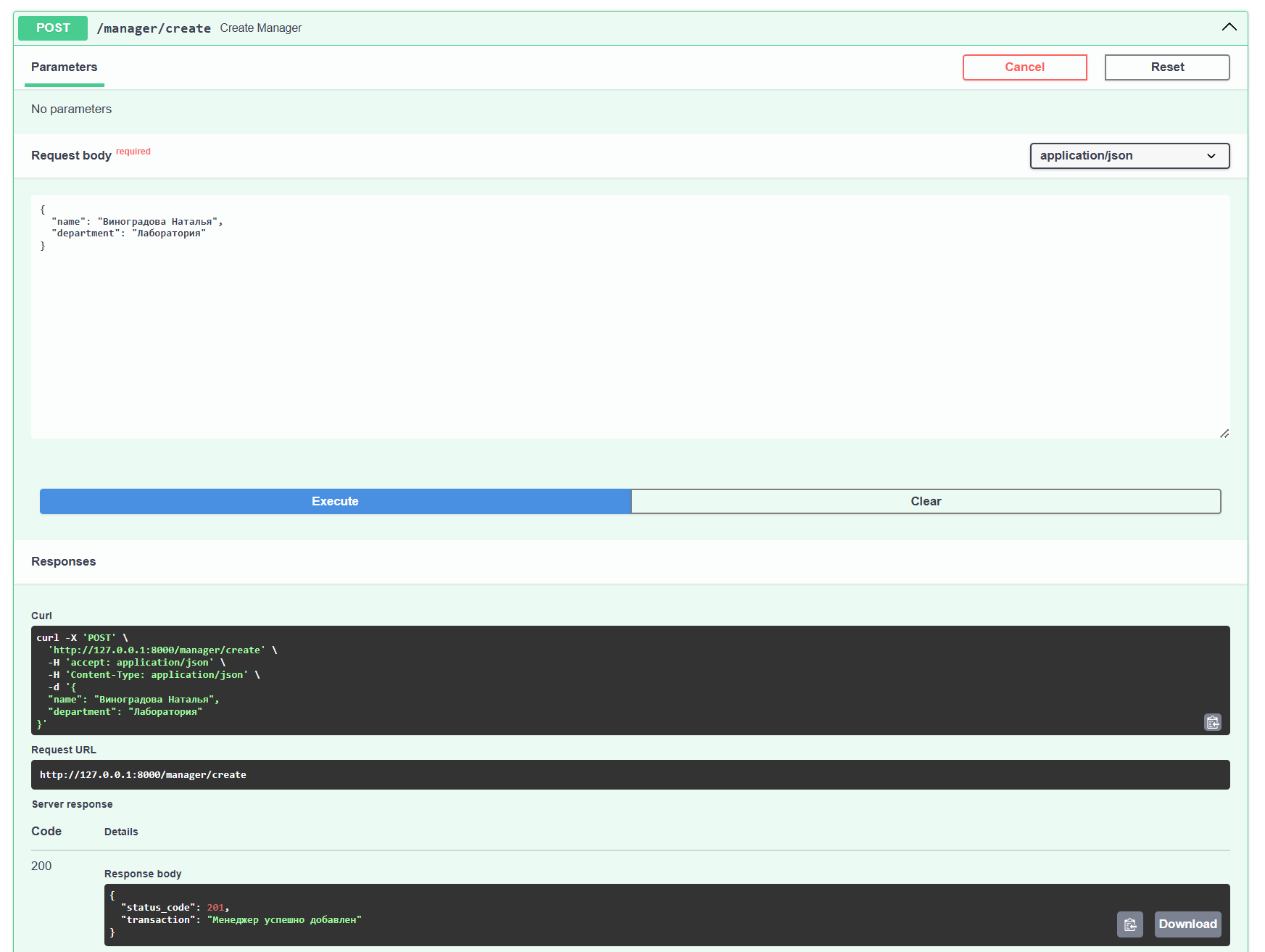


Рис. 3.4.4 Создание записи в таблицу



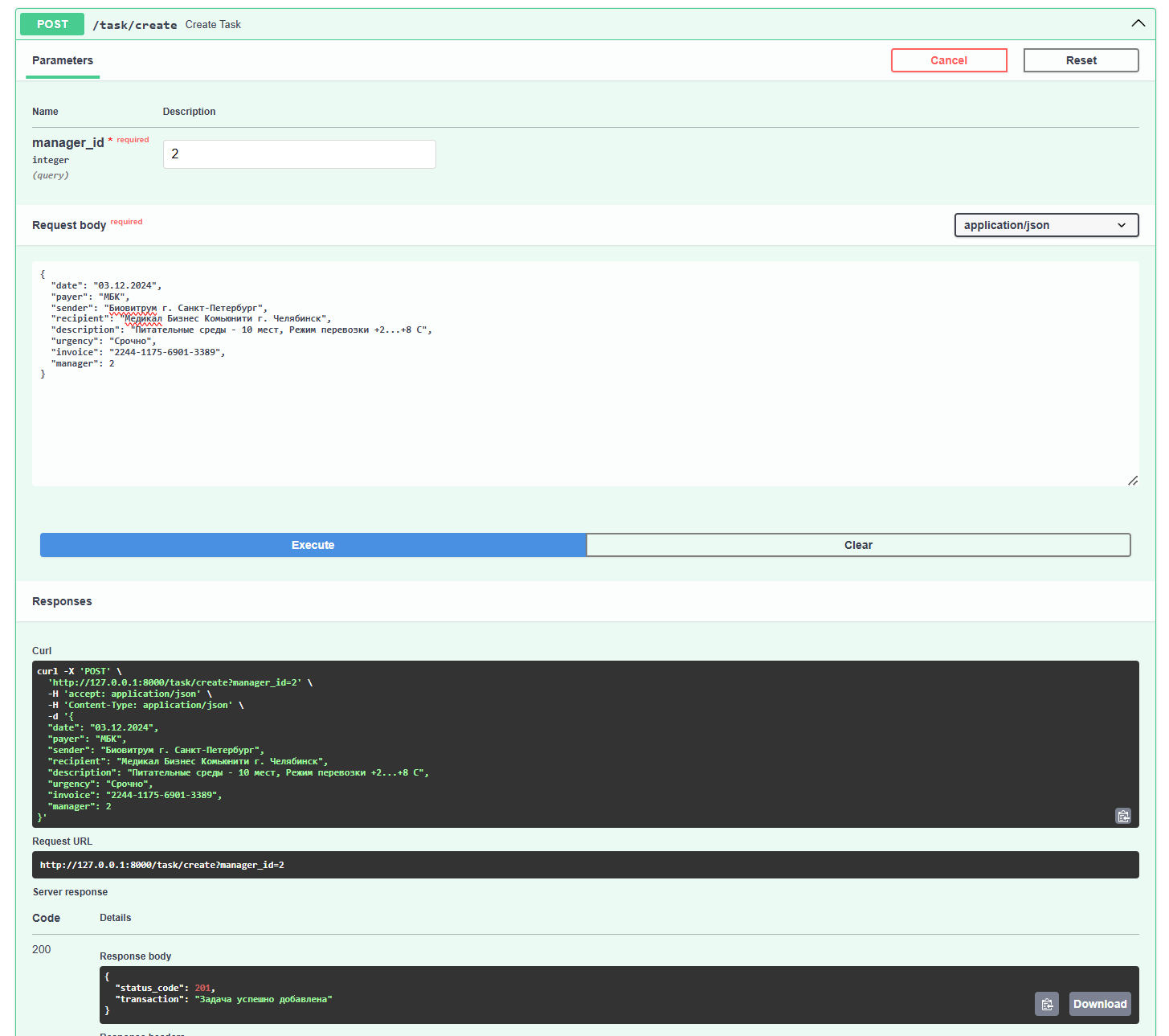


Рис. 3.4.5 Получение всех записей из таблицы

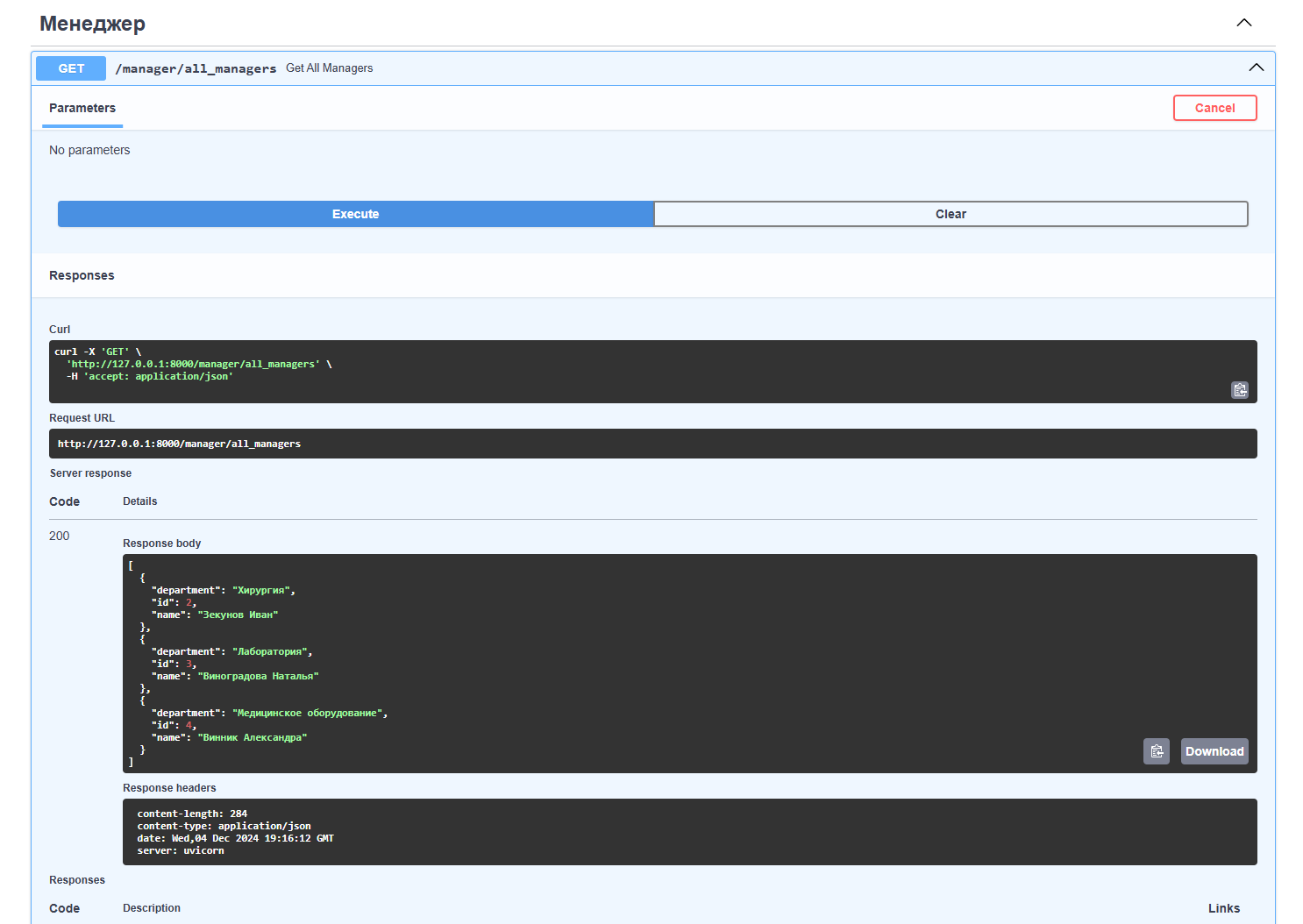
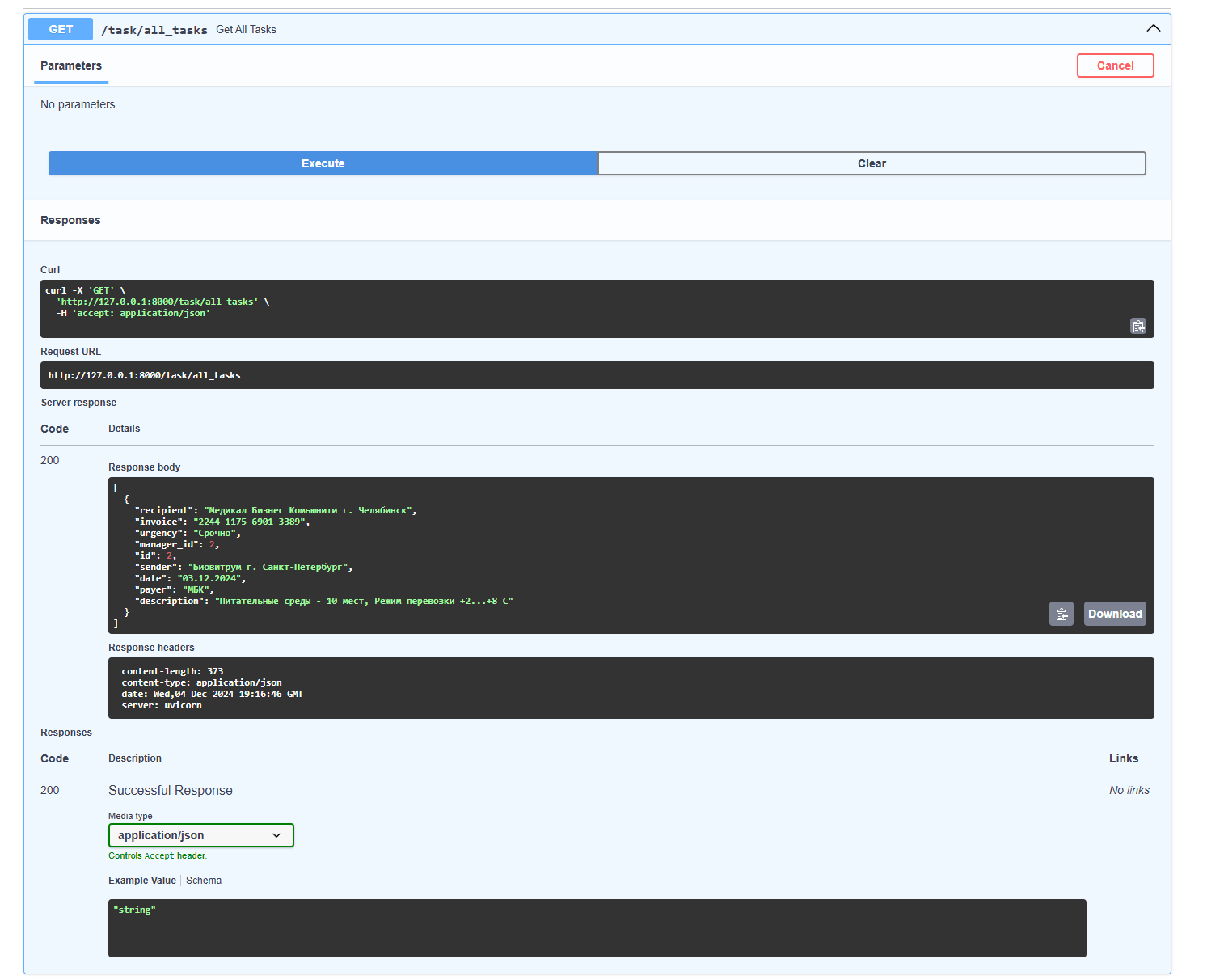
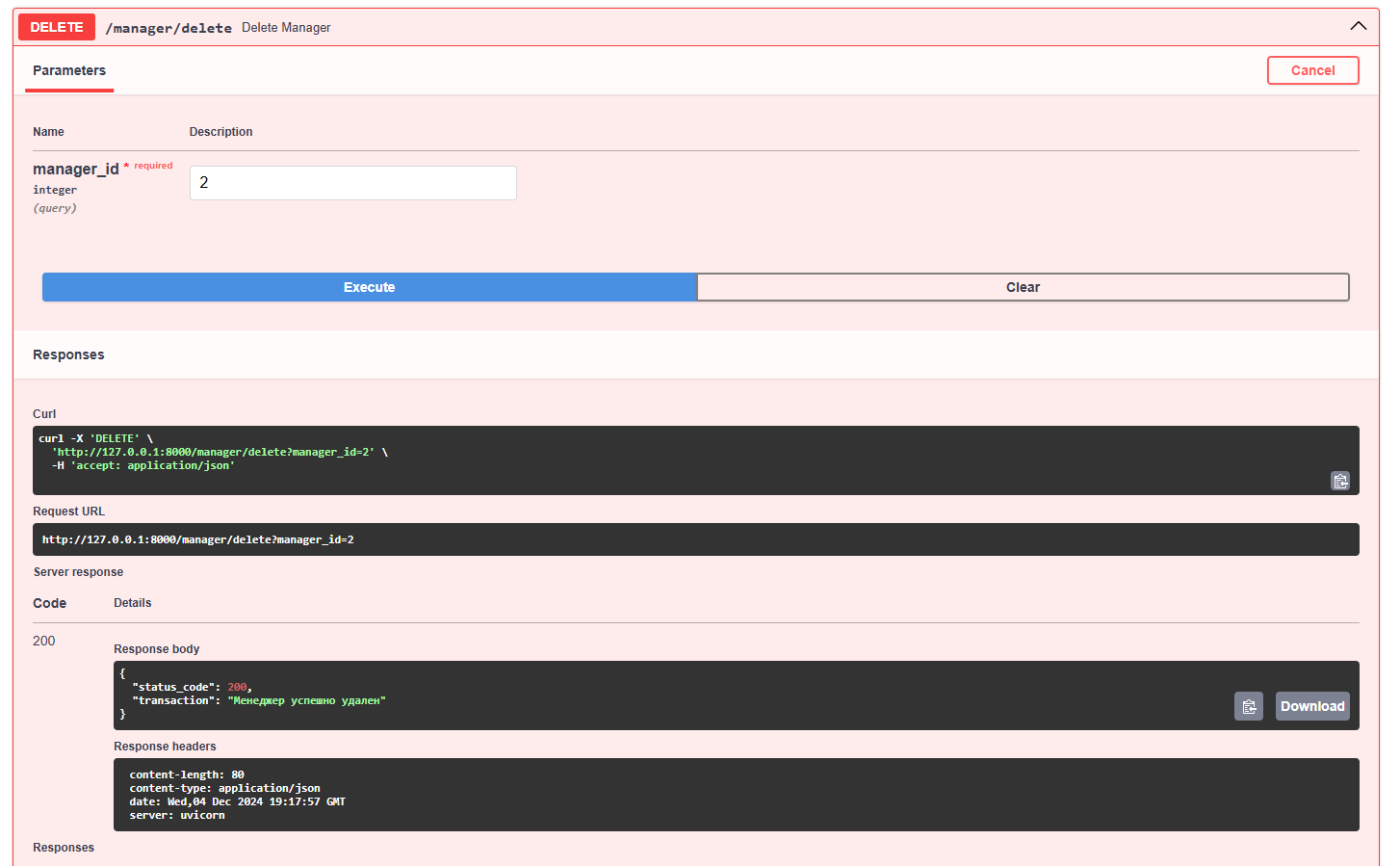


Рис. 3.4.6 Удаление записи из таблицы



1. **Сравнение и анализ результатов**

Django ORM, SQLAlchemy и Tortoise ORM – это три популярных инструмента для работы с базами данных в Python. Каждый из них имеет свои особенности, преимущества и недостатки.

**4.1. Django ORM**

Описание:

Django ORM является частью фреймворка Django и предоставляет высокоуровневый интерфейс для взаимодействия с базой данных через объектную модель. Это одна из самых известных ORM-систем среди разработчиков веб-приложений на Python.

Преимущества:

* Интеграция с Django: Django ORM идеально интегрируется с другими компонентами Django, такими как формы, админ-панель и система аутентификации.
* Простота использования: благодаря интуитивному синтаксису и встроенной поддержке множества типов полей, работа с моделями становится очень удобной.
* Широкая поддержка: Django ORM поддерживает множество СУБД, включая PostgreSQL, MySQL, SQLite и Oracle.
* Автоматическая миграция: Встроенная система миграции позволяет легко изменять схемы базы данных без необходимости ручного редактирования SQL-запросов.

Недостатки:

* Зависимость от Django: если вы не используете Django, то использование Django ORM может оказаться избыточным.
* Производительность: в некоторых случаях производительность Django ORM может уступать низкоуровневым решениям, таким как SQLAlchemy.
* Ограниченность гибкости: хотя Django ORM предлагает множество возможностей, она менее гибкая по сравнению с SQLAlchemy при работе с более сложными запросами.

**4.2. SQLAlchemy**

Описание:

SQLAlchemy – это мощный инструмент для работы с реляционными базами данных, который обеспечивает как низкоуровневое взаимодействие с базой данных (через SQL), так и высокоуровневую работу с объектами (через ORM).

Преимущества:

* Гибкость: SQLAlchemy предоставляет полный контроль над SQL-запросами, позволяя разработчику писать сложные запросы вручную.
* Поддержка нескольких уровней абстракции: Вы можете использовать SQLAlchemy как на уровне ORM, так и на уровне SQL-выражений, что делает его универсальным инструментом.
* Расширяемость: SQLAlchemy легко расширяется за счет плагинов и дополнительных модулей.
* Высокая производительность: SQLAlchemy оптимизирован для высокой производительности и масштабируемости.

Недостатки:

* Из-за своей гибкости и мощи SQLAlchemy требует больше времени на освоение по сравнению с Django ORM.
* Отсутствие встроенных инструментов: В отличие от Django ORM, SQLAlchemy не включает в себя такие инструменты, как админ-панель или систему аутентификации.
* Более сложный синтаксис: Синтаксис SQLAlchemy может показаться сложным для новичков, особенно при написании сложных запросов.

**4.3. Tortoise ORM**

Описание:

Tortoise ORM – это асинхронная ORM-библиотека для Python, которая ориентирована на работу с асинхронными фреймворками, такими как FastAPI и Starlette. Она вдохновлена Django ORM и предоставляет аналогичный высокоуровневый интерфейс для работы с базами данных.

Преимущества:

* Асинхронность: Tortoise ORM специально разработана для работы в асинхронных приложениях, что делает её идеальным выбором для высоконагруженных систем.
* Простота использования: как и Django ORM, Tortoise ORM имеет простой и понятный синтаксис, что облегчает начало работы.
* Совместимость с популярными фреймворками: Tortoise ORM хорошо интегрируется с такими фреймворками, как FastAPI и Starlette.
* Поддержка миграции: Tortoise ORM включает в себя систему миграции, аналогичную той, что используется в Django.

Недостатки:

* Ограниченная поддержка СУБД: на данный момент Tortoise ORM поддерживает ограниченное количество СУБД, таких как PostgreSQL, MySQL и SQLite.
* Молодость проекта: Tortoise ORM относительно новый проект, поэтому могут возникать проблемы с документацией и поддержкой.
* Меньшая популярность: по сравнению с Django ORM и SQLAlchemy, Tortoise ORM менее популярен и имеет меньшее сообщество пользователей.

1. **Заключение**

Выбор ORM зависит от потребностей и требований проекта. Django ORM отлично подходит для проектов, где требуется быстрый старт с фреймворком и готовыми решениями. SQLAlchemy предоставляет большую гибкость и мощность, что позволяет использовать его в более сложных сценариях. Tortoise ORM хорош для асинхронных приложений и имеет простой интерфейс.