# ФГАОУ ВО «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Отчет по курсовому проекту по дисциплине

«Инженерное проектирование»

Тема: «Проектирование веб-приложения на серверной стороне»

Выполнл студент 1 курса группы 231-322

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Супроткин Р.В.

Проверила

Старший преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Даньшина М.В.

**МОСКВА 2023**

# Содержание (****Критерии на оценку удовлетворительно****)

1. Реализовано API с помощью Django REST framework для минимум 2х моделей вашей предметной области
2. Есть минимум 1 запрос c Q. Запросы должны быть осмысленными и подходящими под вашу предметную область
3. Есть пагинация
4. Есть 1 вариант фильтрации
5. Есть сохранение истории изменений объектов
6. Экспорт данных из админки в эксель - реализовано только подключение экспорта данных в адмике, без дополнительных настроек
7. Написана 1 management команда
8. Реализован на выбор либо @action(methods=['GET'], detail=False), либо  @action(methods=['POST'], detail=True)
9. В admin.py есть list\_display, fields, list\_filter, есть Гиперссылки на страницы других объектов, short\_description, fieldsets, inlines. Можно заменить на другие настройки, должно быть минимум 5.
10. Ответить на минимум на 1 вопрос из 3х предложенных из списка на выбор преподавателя
11. **Реализовано API с помощью Django REST framework для минимум 2х моделей вашей предметной области**

**Модели:**

****

*Код models.py проекта*

### **Student (Студент)**

Эта модель описывает студентов и содержит следующие поля:

* first\_name и last\_name — имя и фамилия студента.
* university — название университета, где обучается студент.
* faculty — факультет университета.
* course\_number — номер курса студента.

Модель включает **HistoricalRecords** для отслеживания изменений данных студента. Поле **\_\_str\_\_** возвращает читаемое представление студента с указанием его фамилии, имени и университета.

### **Project (Проект)**

Модель описывает проекты, в которых участвуют студенты:

* title — название проекта.
* field — направление проекта (например, техническое, гуманитарное и т. д.).
* students — связь "многие ко многим" с моделью Student, что позволяет указать участников проекта.
* description — текстовое описание проекта.

Как и модель **Student**, она использует **HistoricalRecords** для истории изменений. Поле **\_\_str\_\_** возвращает название проекта.

### **Event (Мероприятие)**

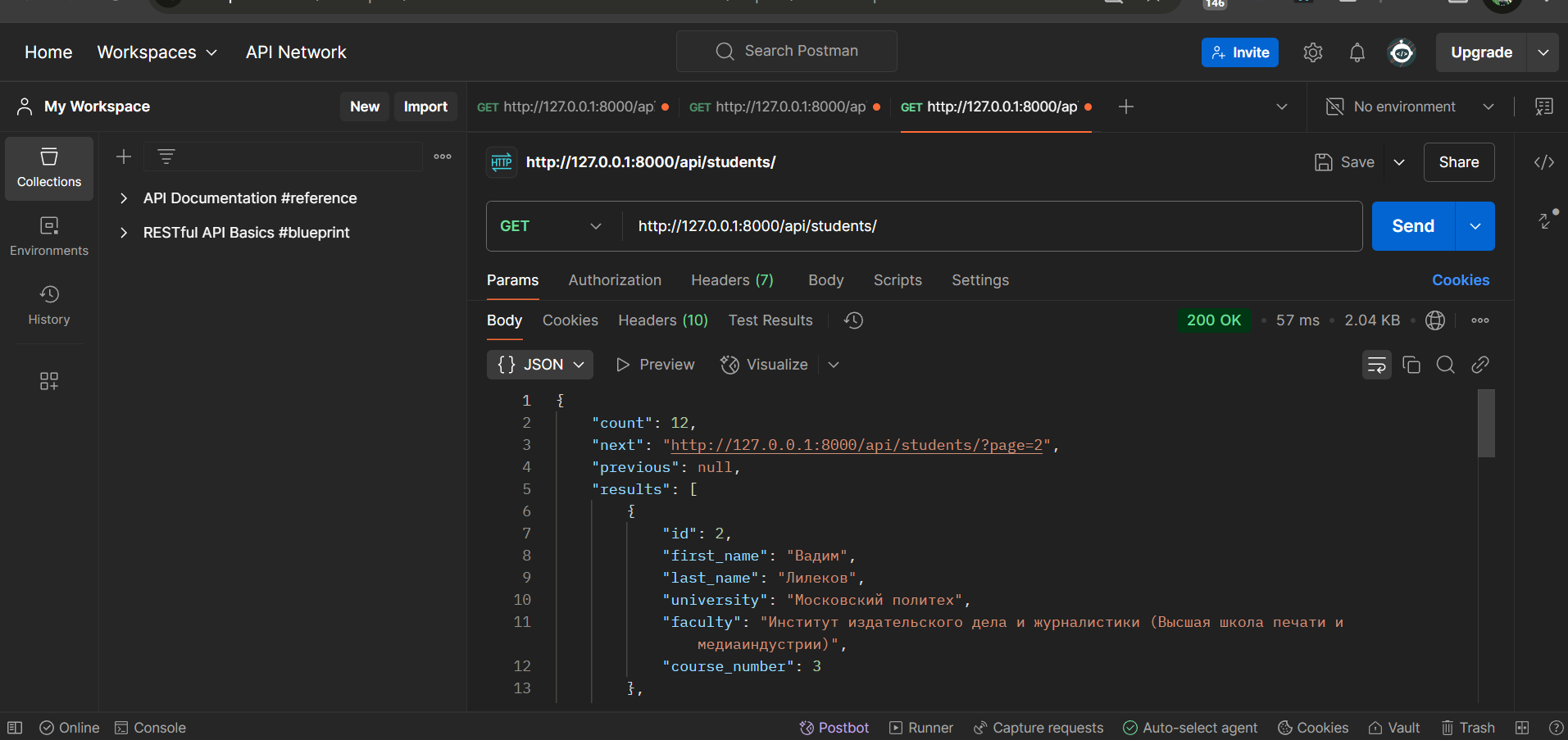
Модель для хранения информации о мероприятиях:

* title — название мероприятия.
* field — направление мероприятия.
* students — связь "многие ко многим" с моделью **Student**, позволяющая добавлять участников мероприятия.
* description — описание мероприятия.
* address — адрес проведения мероприятия.

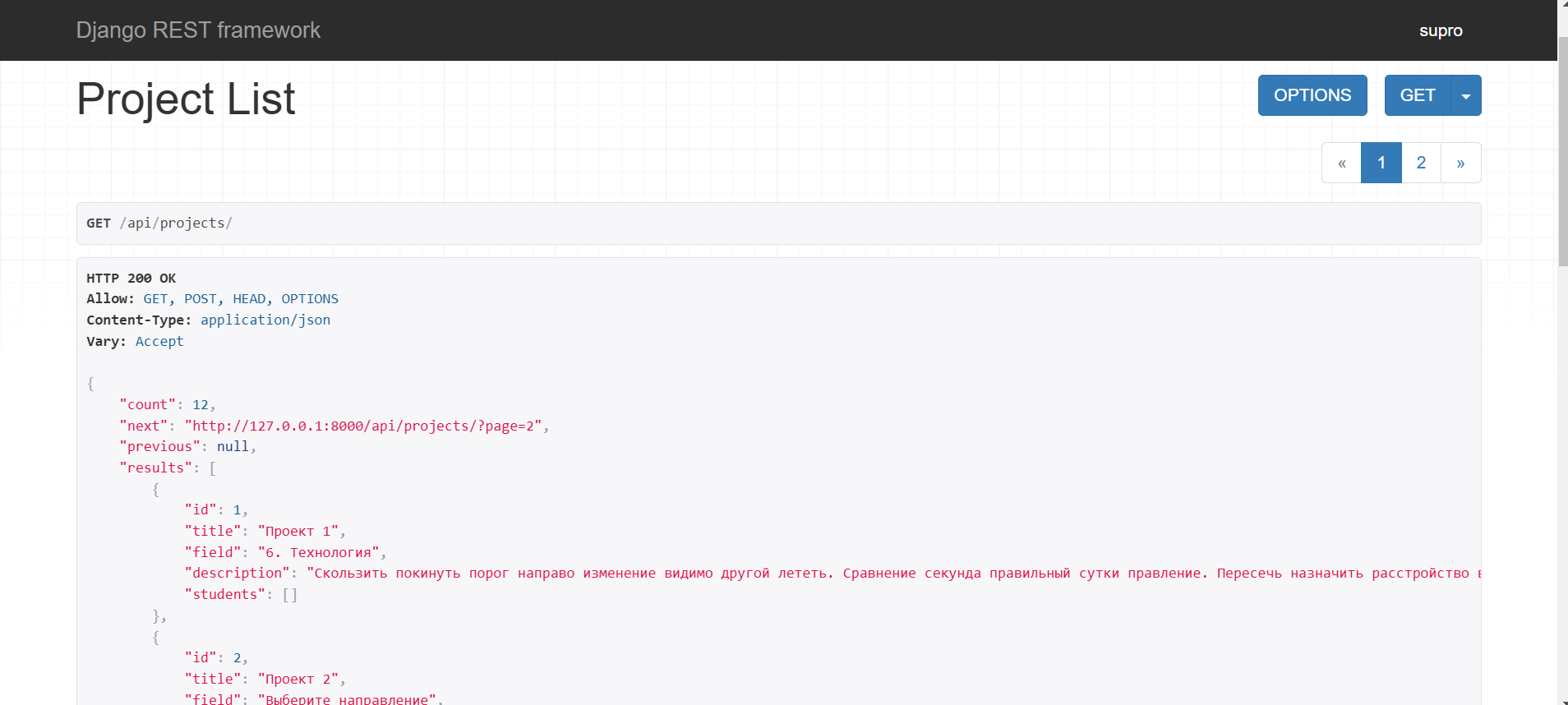
Также поддерживается история изменений с помощью **HistoricalRecords**, а метод **\_\_str\_\_** возвращает название мероприятия.

### **Особенности реализации**

* **Связи между моделями**: Модели Project и Event имеют связь "многие ко многим" с моделью Student, что позволяет гибко управлять участниками.
* **Исторические записи**: Все модели включают HistoricalRecords для автоматического ведения журнала изменений, что удобно для аудита.
* **Лаконичное отображение**: Методы \_\_str\_\_ делают работу с объектами в интерфейсе админки более удобной.

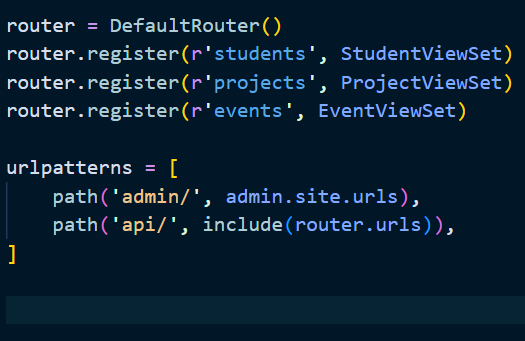
****

*Демонстрация работы API в Postman*

**

*Демонстрация работы API в браузере*

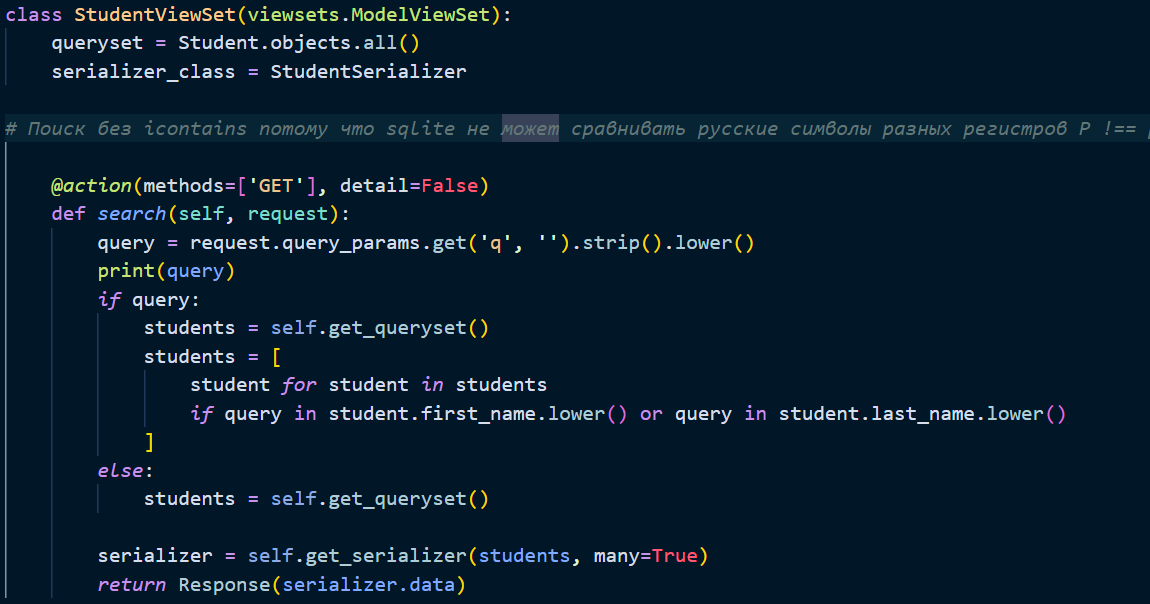
Ссылки реализованы через **DefaultRouter()**



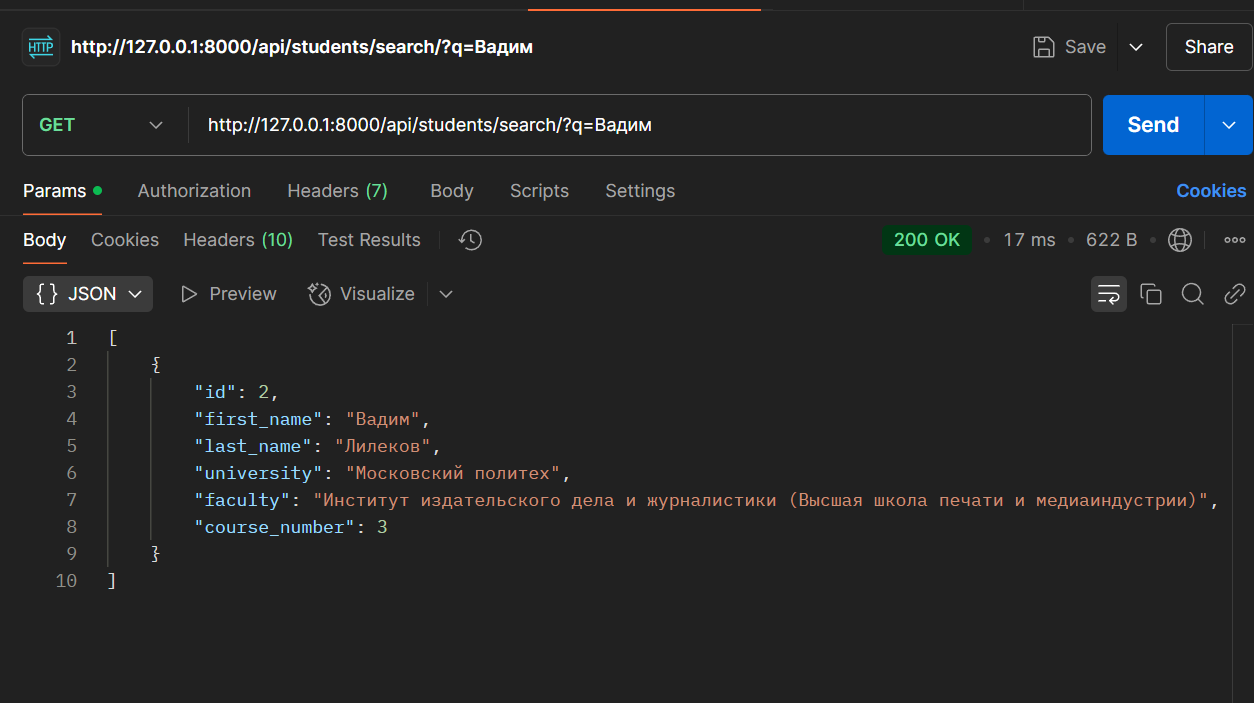
*Реализация ссылок в проекте*

1. **Есть минимум 1 запрос c Q. Запросы должны быть осмысленными и подходящими под вашу предметную область**

С использованием query параметров (q) реализована функция поиска по записям о студентах

****

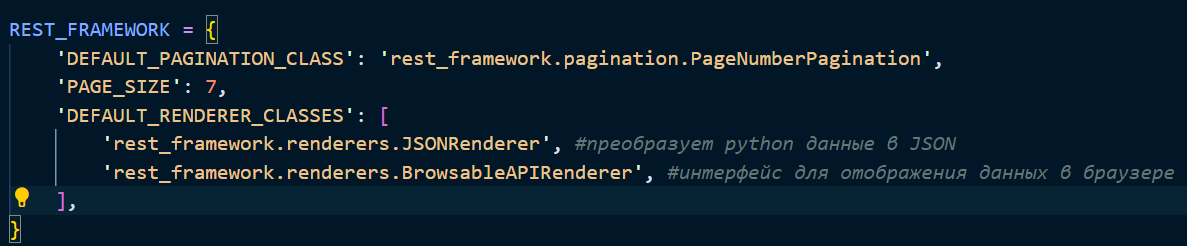
*Реализация q в запросах*



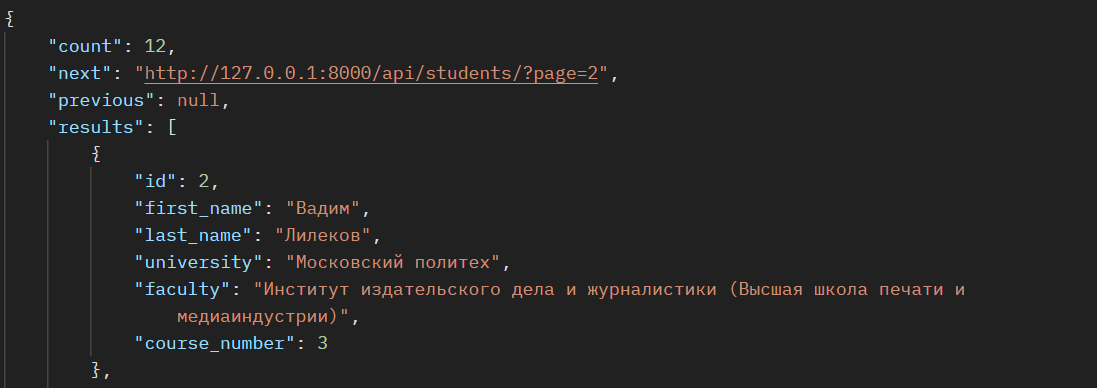
*Демонстрация q запросов в Postman*

1. **Есть пагинация**

Пагинация реализована следующим образом: в settings.py проекта была добавлена настройка



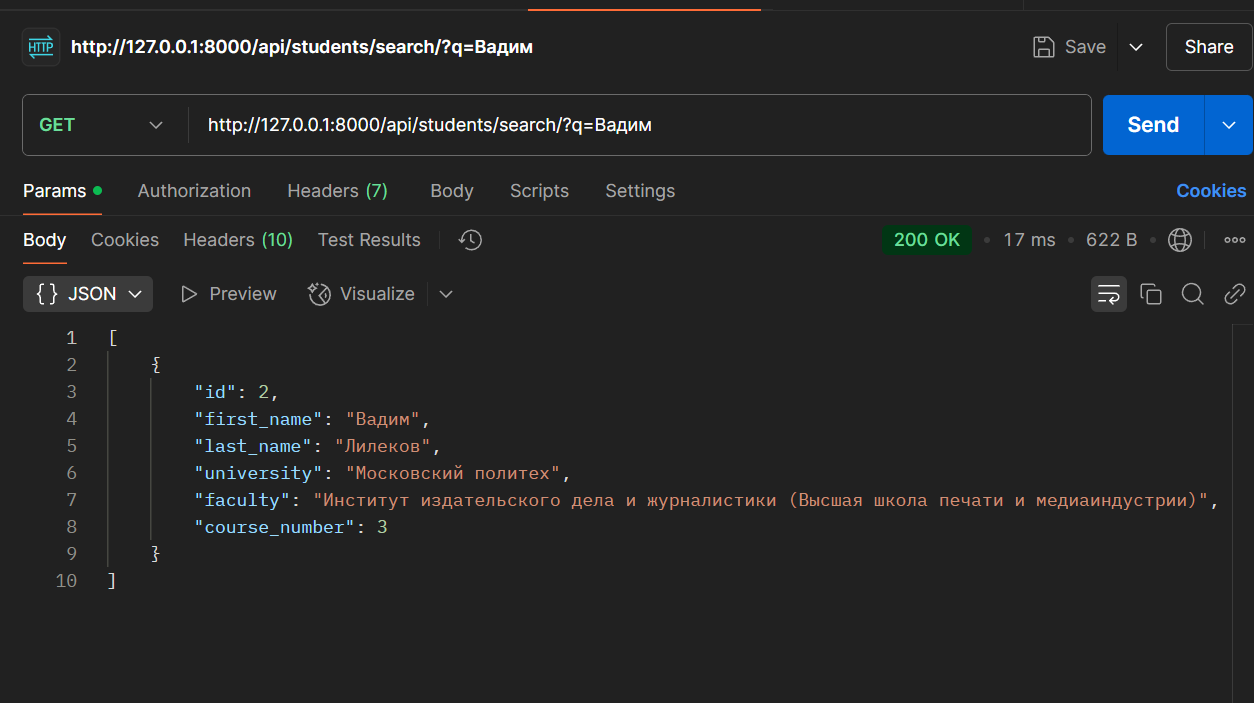
*Код реализации пагинации*



*Демонстрация работы системы пагинации в ответах API при помощи Postman*

1. **Есть 1 вариант фильтрации**

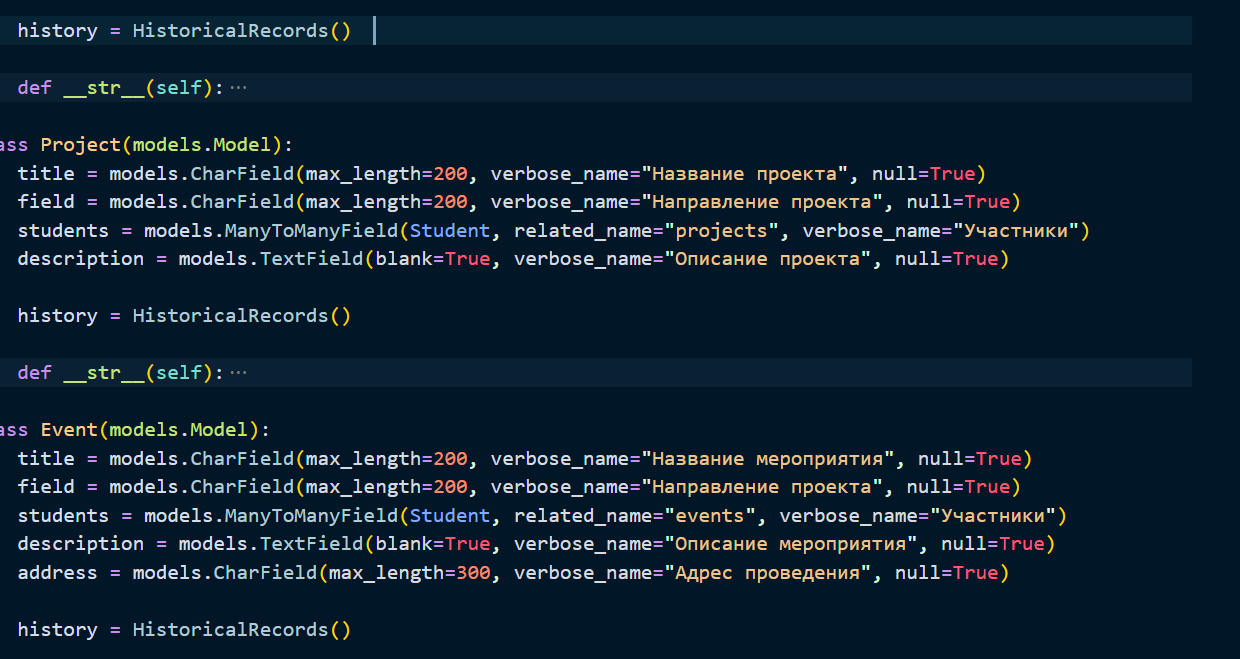
Фильтрация по имени и фамилии реализована при помощи q-параметров (их реализацию см. в соответствующем разделе)



*Демонстрация фильтрации по имени и фамилии в Postman*

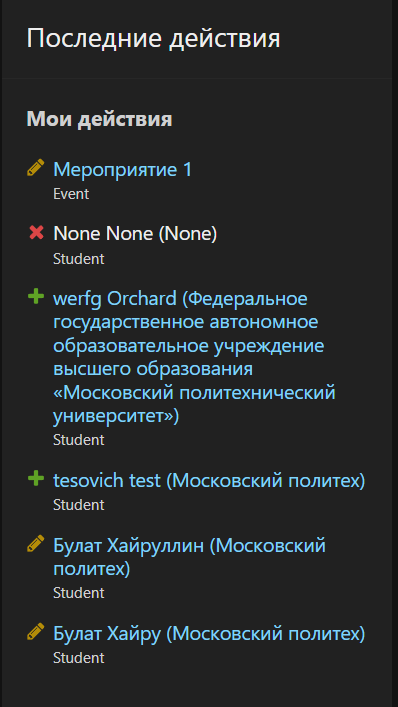
1. **Есть сохранение истории изменений объектов**

Запись истории ведётся при помощи **HistoricalRecords()**



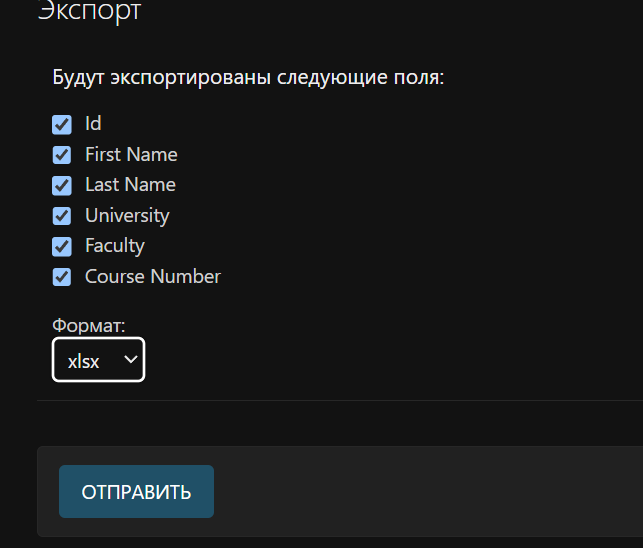
*Реализация сохранения истории изменений объектов*

Последние действия отображаются в админке

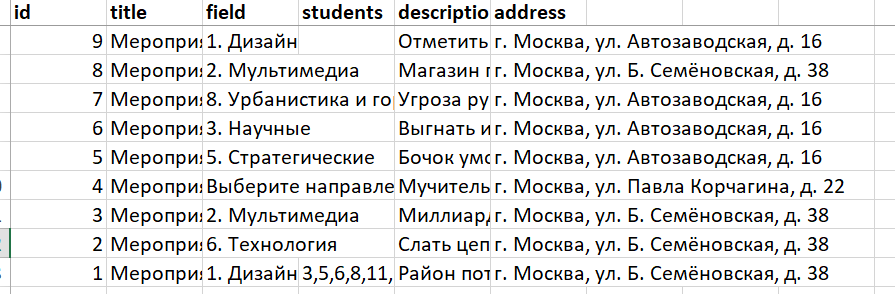


*Последние действия отображаются в админке*

1. **Экспорт данных из админки в эксель - реализовано только подключение экспорта данных в адмике, без дополнительных настроек**

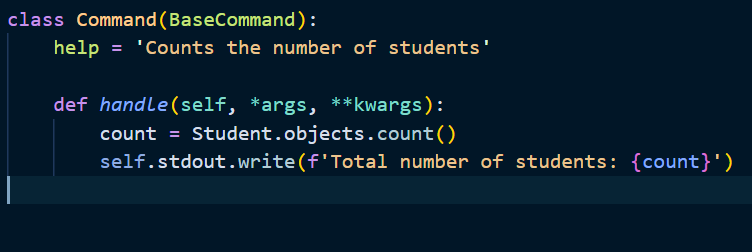


*Демонстрация экспорта данных админки в Excel*



*Демонстрация экспорта данных админки в Excel (2)*

1. **Написана 1 management команда**

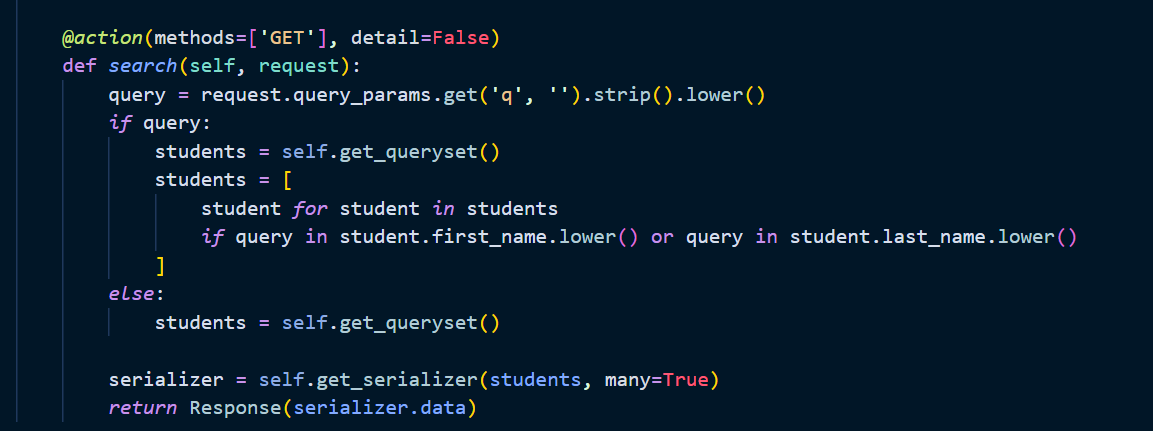
****

*Реализация management команды*

**Management-команды в Django** — это пользовательские скрипты, которые позволяют выполнять задачи через командную строку, используя инфраструктуру Django. Они создаются в папке management/commands приложения.

**Приведённая команда:**

1. Определяется в классе, наследующем **BaseCommand**.
2. **Назначение**: Подсчёт общего числа студентов в модели **Student** (из приложения **portfolio**).
3. Метод handle выполняет логику команды:
   * С помощью **Student.objects.count()** подсчитывается количество записей в таблице студентов.
   * Результат выводится в консоль с помощью **self.stdout.write**.
4. **Реализован на выбор либо @action(methods=['GET'], detail=False), либо  @action(methods=['POST'], detail=True)**



*Реализация @action(methods=['GET'], detail=False)*

Метод search, помеченный декоратором @action, позволяет выполнять поиск студентов по имени или фамилии. Он принимает GET-запрос с параметром q, который используется для фильтрации студентов. Если параметр q не указан, возвращаются все студенты.

Обратите внимание, что поиск осуществляется без использования icontains, так как SQLite не поддерживает сравнение русских символов с учетом регистра. Вместо этого используется метод lower() для приведения строк к нижнему регистру перед сравнением.

В конце метод сериализует найденные объекты и возвращает их в формате JSON.

1. **В admin.py есть list\_display, fields, list\_filter, есть Гиперссылки на страницы других объектов, short\_description, fieldsets, inlines. Можно заменить на другие настройки, должно быть минимум 5.**



*Реализация настроек админки Django REST Framework*



*Реализация настроек админки Django REST Framework (2)*