

Министерство науки и высшего образования Российской Федерации Федеральное
государственное бюджетное образовательное учреждение высшего образования
«Московский политехнический университет»

Кафедра «Инфокогнитивные технологии»
Образовательная программа «Веб-технологии»

Отчет по курсовому проекту
по дисциплине «Инженерное проектирование»

Тема: «Спортивные мероприятия»

Выполнил:

Студент группы 181-322

Ахмедов Д.К.

подпись, дата

Принял:

Старший преподаватель

Даньшина М.В.

подпись, дата

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках дисциплины “Инженерное проектирование” в Московском политехническом университете мною было разработано приложение для спортивных мероприятий и новостей.

В качестве аналогов были выбраны:

1. <https://www.sports.ru>
2. <https://sportbox.ru>

Благодаря разработке можно быстро управлять новостями, мероприятиями, подкастами, турнирами и другим с помощью автоматизированной системы.

АНАЛИЗ АНАЛОГОВ

Первый из найденных мною аналогов был <https://www.sports.ru>. На данном сайте присутствуют такие важные разделы: Новости, Подкасты, События по разными видам спорта.

Второй аналог - <https://sportbox.ru>. На нем также представлены эти разделы.

По функционалу эти сайты очень похожи. Есть возможность просматривать, создавать, редактировать и удалять события, новости, мероприятия, подкасты и тд. Представлены статусы матчей и их результат, информация о спонсорах и турнирах.

По структуре эти сайты тоже очень похожи. Есть разделы с определёнными видами спорта, страница с новостями, подкастами, видео и тд.

Проведя анализ этих сайтов я выделил функционал создаваемого приложения: CRUD новостей, мероприятий, подкастов, турниров, команд, игроков, спонсоров, авторов, категорий.

ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

После анализа конкурентов я приступил к созданию базы данных. Для этого я выделил основные сущности:

1. Новость
2. Подкаст
3. Пользователь
4. Категория
5. Турнир
6. Спонсор
7. Команда
8. Игрок
9. Тип команды
10. Тип пользователя

Определив основные сущности я начал прописывать параметры для этих сущностей:

1. Новость
 - 1.1. Заголовок
 - 1.2. Контент
 - 1.3. Дата создания
 - 1.4. Дата обновления
 - 1.5. Фотография
 - 1.6. Статус публикации
 - 1.7. Категория
 - 1.8. Автор
2. Подкаст
 - 2.1. Заголовок
 - 2.2. Описание
 - 2.3. Дата создания
 - 2.4. Аудио файл
 - 2.5. Статус публикации
 - 2.6. Категория
 - 2.7. Автор
3. Пользователь

- 3.1. Email
- 3.2. Пароль
- 3.3. Имя
- 3.4. Фамилия
- 3.5. Тип пользователя
- 4. Категория
 - 4.1. Название
- 5. Турнир
 - 5.1. Название
 - 5.2. Призовой фонд
 - 5.3. Спонсор
- 6. Спонсор
 - 6.1. Название
 - 6.2. Логотип
- 7. Команда
 - 7.1. Название
 - 7.2. Логотип
 - 7.3. Игроки
 - 7.4. Тип команды
- 8. Игрок
 - 8.1. Имя
 - 8.2. Фамилия
 - 8.3. Номер
- 9. Тип команды
 - 9.1. Название
- 10. Тип пользователя
 - 10.1. Название

Выделив основные сущности и их параметры я спроектировал ERD для наглядного отображения связей между таблицами (рис. 1):

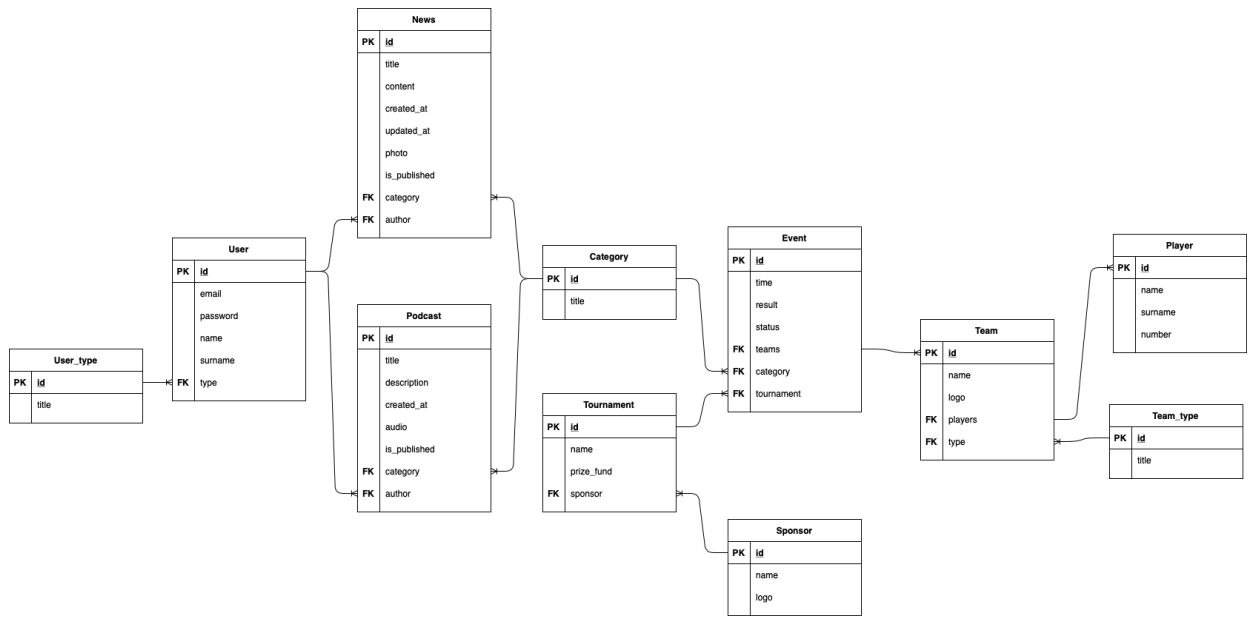


рис. 1. ERD

СОЗДАНИЕ ПРИЛОЖЕНИЯ

Для решения задачи был выбран язык программирования Python и его фреймворк для создания web-приложения Django.

Так как я не знаком с языком и фреймворком, я начал учить его по видео на YouTube и документации.

Изучив основы я приступил к созданию приложения. Оно началось с создания репозитория на <https://github.com>. Далее я подключил виртуально окружение, установил Django и генерировал проект под названием sports.

После создания проекта я разделил проект на несколько приложений (рис. 2):

1. Новости
2. Мероприятия
3. Команды
4. Категории
5. Подкасты

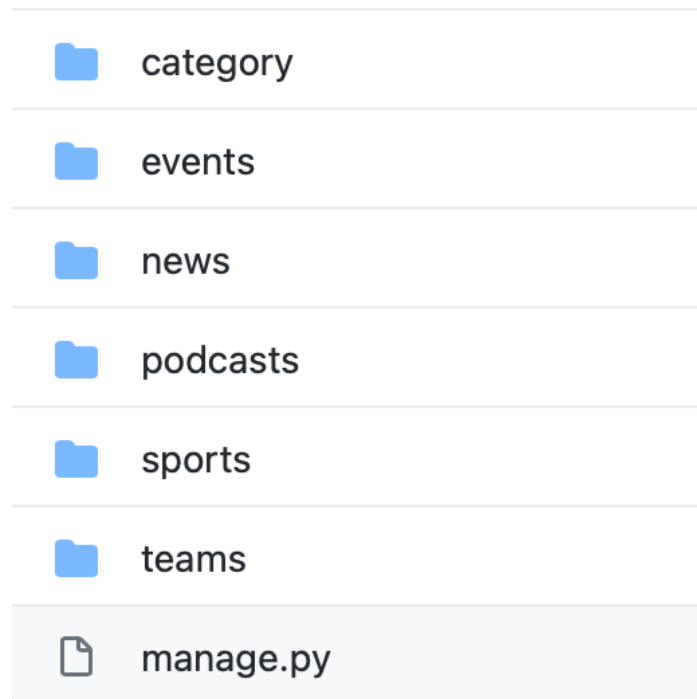


рис. 2. Структура проекта

Создав приложения я начал переносить спроектированную базу данных в код, в файлы models.py в приложениях (рис. 3).


```
class News(models.Model):
    title = models.CharField(max_length=64, verbose_name='Название')
    content = models.TextField(verbose_name='Текст')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата создания')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Дата обновления')
    photo = models.ImageField(upload_to='photos/%Y/%m/%d/', verbose_name='Фотография', blank=True)
    is_published = models.BooleanField(default=True, verbose_name='Опубликовано')
    category = models.ForeignKey(Category, on_delete=models.CASCADE, verbose_name='Категория')
    author = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Автор')
```

рис. 3. Код models.py

После создания всех моделей я выполнил миграции и подключил приложения к проекту. В следствии чего в базе данных создались таблицы в соответствии написанному коду, а в адмнике проекта появились приложения (рис. 4).

КАТЕГОРИИ	
Категории	+ Добавить
КОМАНДЫ	
Игрок	+ Добавить
Команды	+ Добавить
Тип команд	+ Добавить
МЕРОПРИЯТИЯ	
Мероприятия	+ Добавить
Спонсоры	+ Добавить
Турниры	+ Добавить
НОВОСТИ	
Новости	+ Добавить
ПОДКАСТЫ	
Подкасты	+ Добавить

рис. 4. Админка

НАСТРОЙКА АДМИНИСТРАТИВНОГО ИНТЕРФЕЙСА

Изначально административный интерфейс не локализован, названия полей генерируется машиной, нет фильтрации, сортировки, поиска, импорта, экспорта. Это может вызвать некоторые проблемы при его использовании.

После выявления этих проблем я приступил к их решению.

Локализация интерфейса и корректные название полей достиглись путем смены языка в настройках проекта (`LANGUAGE_CODE = 'ru'`), прописывания `verbose_name` у параметров моделей, определений функции `srt` и класса `Meta` (рис. 5 и рис. 6).

```
class News(models.Model):
    title = models.CharField(max_length=64, verbose_name='Название')
    content = models.TextField(verbose_name='Текст')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата создания')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Дата обновления')
    photo = models.ImageField(upload_to='photos/%Y/%m/%d/', verbose_name='Фотография', blank=True)
    is_published = models.BooleanField(default=True, verbose_name='Опубликовано')
    category = models.ForeignKey(Category, on_delete=models.CASCADE, verbose_name='Категория')
    author = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Автор')

    def __str__(self):
        return self.title

class Meta:
    verbose_name = 'Новость'
    verbose_name_plural = 'Новости'
    ordering = ['created_at']
```

рис. 5. Модель новости

Добавить Новость

Название:

Текст:

Фотография: no file selected

☒ Опубликовано

Категория:

Автор:

рис. 6. Добавление новости в админке

Фильтрация, поиск и сортировка были добавлены с помощью вспомогательного класса, которые настраивать админка для определенной модели (рис. 7):

```
class NewsAdmin(ImportExportModelAdmin):
    list_display = ('title', 'created_at', 'updated_at', 'is_published', 'category', 'author')
    search_fields = ('title', 'content')
    list_filter = ('created_at', 'is_published', 'category', 'author')
    actions = [make_published]
```

рис. 7. Класс настройки отображения

Благодаря этому классу мы получаем фильтрацию, поиск и сортировку по заданным нам параметрам (рис. 8).

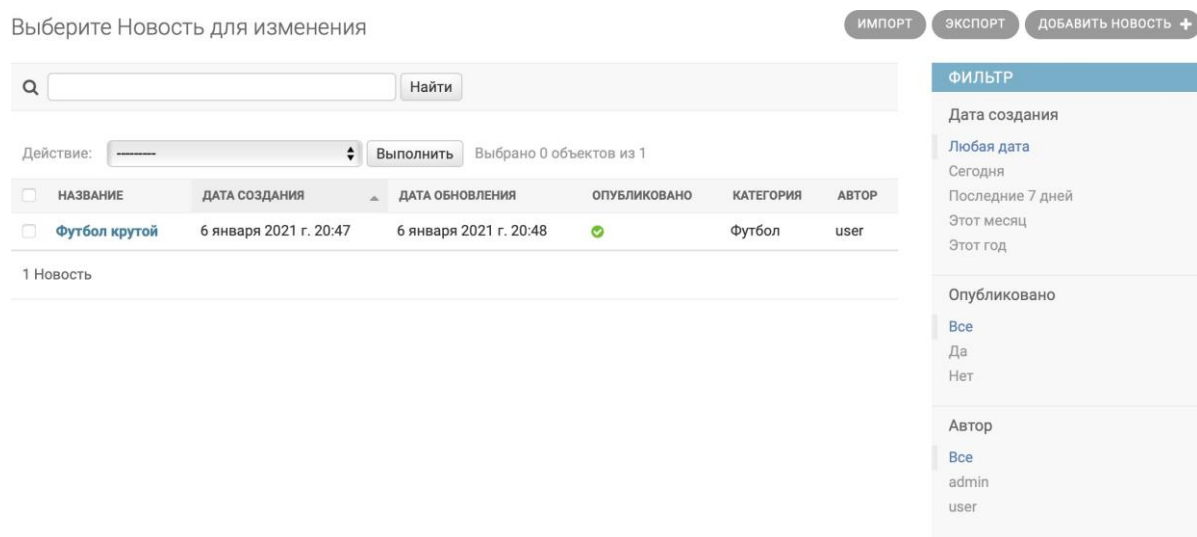


рис. 8. Поиск, фильтрация, сортировка

Для импорта и экспорта данных я использовал библиотеку `django-import-export`, которая добавляет возможность импорта и экспорта данных в различных форматах (рис. 9).

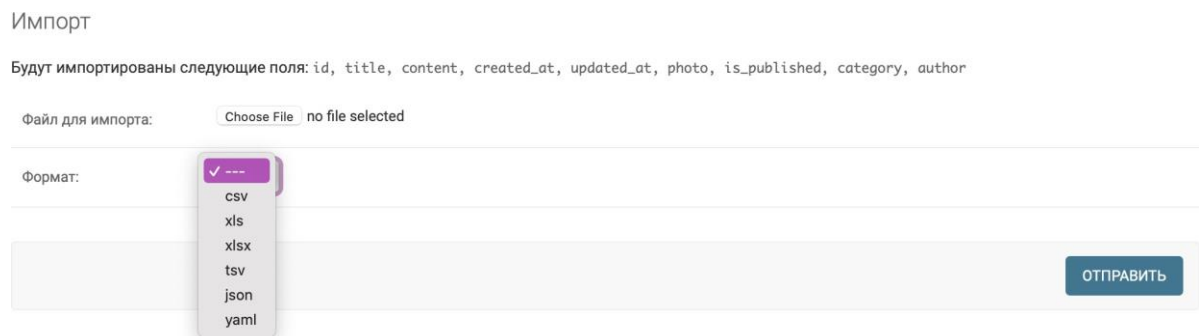


рис. 9. Импорт и экспорт данных

Также были реализованных admin-actions для более удобного взаимодействия с данными. Я реализовал кейс для публикации сразу нескольких новостей. Это достигалось путем написания простой функции (рис. 10):

```
def make_published(modeladmin, request, queryset):
    queryset.update(is_published=True)

make_published.short_description = "Опубликовать несколько"
```

рис. 10. Функция admin-actions

После добавления этой функции в поле actions в админке стало доступно действие с помощью которого можно выделить несколько новостей и опубликовать их (рис. 11).

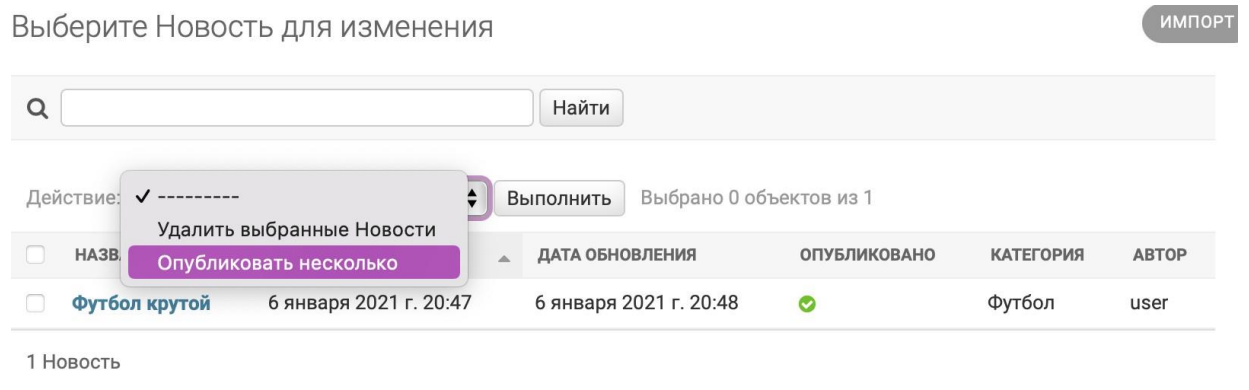


рис. 11. Admin-actions

РЕАЛИЗАЦИЯ REST-API

Для реализации rest-api я воспользовался фреймворком Django-rest-framework. Я выбрал 2 кейса для его изучения: просмотр всех новостей и добавление новости.

В файле views.py я создал класс NewsView, который будет отвечать за отправку и добавление новостей с помощью http запросов (рис. 12).

```
from rest_framework.response import Response
from rest_framework.views import APIView

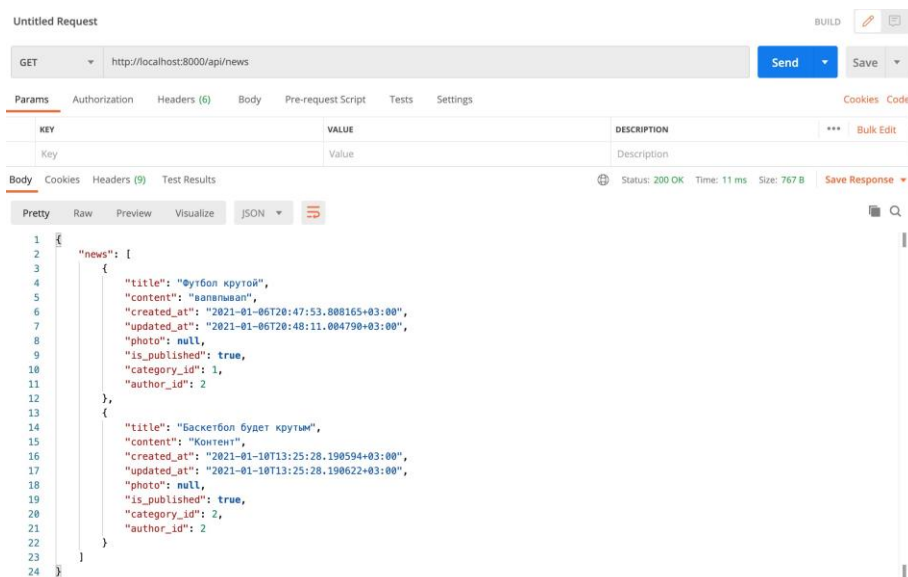
from .models import News
from .serializers import NewsSerializer

class NewsView(APIView):
    def get(self, request):
        news = News.objects.all()
        serializer = NewsSerializer(news, many=True)
        return Response({"news": serializer.data})

    def post(self, request):
        news = request.data.get('news')
        serializer = NewsSerializer(data=news)
        if serializer.is_valid(raise_exception=True):
            news_saved = serializer.save()
            return Response({"success": "News '{}' created successfully".format(news_saved.title)})
```

рис. 12. Класс для работы через rest-api

Для тестирования REST-API я воспользовался программой Postman. Написав запрос к серверу я получил ответ, который содержал все новости (рис. 13).



The screenshot shows the Postman interface with an 'Untitled Request' set to a GET method at the URL 'http://localhost:8000/api/news'. The 'Send' button is visible. Below the request details, the 'Body' tab is selected, showing the JSON response in 'Pretty' format. The response is a JSON object with a 'news' key containing a list of two news items. The status bar at the bottom indicates a successful 200 OK response with a time of 11 ms and a size of 767 B.

```
{
  "news": [
    {
      "title": "Футбол крутой",
      "content": "Анализ",
      "created_at": "2021-01-06T20:47:53.000165+03:00",
      "updated_at": "2021-01-06T20:48:11.004790+03:00",
      "photo": null,
      "is_published": true,
      "category_id": 1,
      "author_id": 2
    },
    {
      "title": "Баскетбол будет крутым",
      "content": "Коммент",
      "created_at": "2021-01-10T13:25:28.190594+03:00",
      "updated_at": "2021-01-10T13:25:28.190622+03:00",
      "photo": null,
      "is_published": true,
      "category_id": 2,
      "author_id": 2
    }
  ]
}
```

рис. 13. Ответ в Postman

Для тестирования я написал 2 теста, которые проверяю, что статус ответа сервера 200 и то, что ответ содержит массив с новостями (рис. 14).

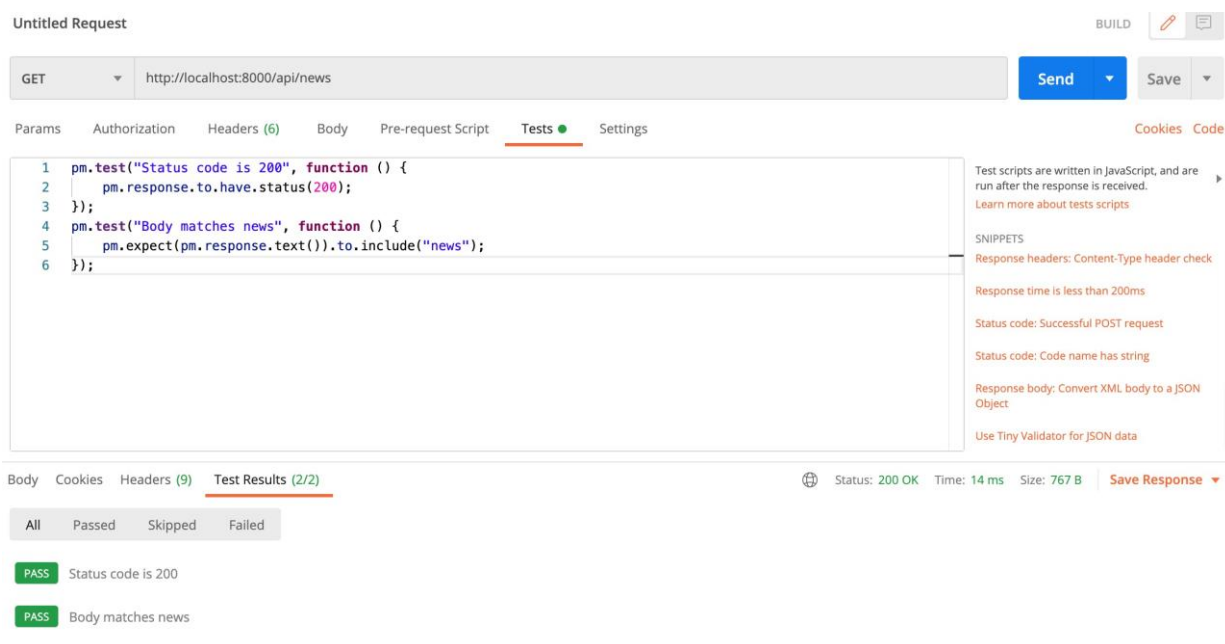


рис. 14. Тестирование в Postman

КЕЙСЫ ИСПОЛЬЗОВАНИЯ

С помощью созданного приложения пользователям можно:

1. Добавлять новости и подкасты
2. Редактировать новости и подкасты
3. Удалять новости и подкасты
4. Просматривать новости и подкасты

Для администраторов доступны такие кейсы:

1. Добавлять новости, подкасты, категории, команды, события, турнир, спонсоров
2. Удалять новости, подкасты, категории, команды, события, турнир, спонсоров
3. Редактировать новости, подкасты, категории, команды, события, турнир, спонсоров
4. Просматривать новости, подкасты, категории, команды, события, турнир, спонсоров
5. Добавлять, удалять, редактировать, просматривать пользователей и группы

Для операторов это приложения пригодится для:

1. Добавлять команды, события, турнир, спонсоров
2. Удалять команды, события, турнир, спонсоров
3. Редактировать команды, события, турнир, спонсоров
4. Просматривать команды, события, турнир, спонсоров

Все эти кейсы использования были реализованы путем создания групп и выдачи им прав на управление данными (рис. 15).

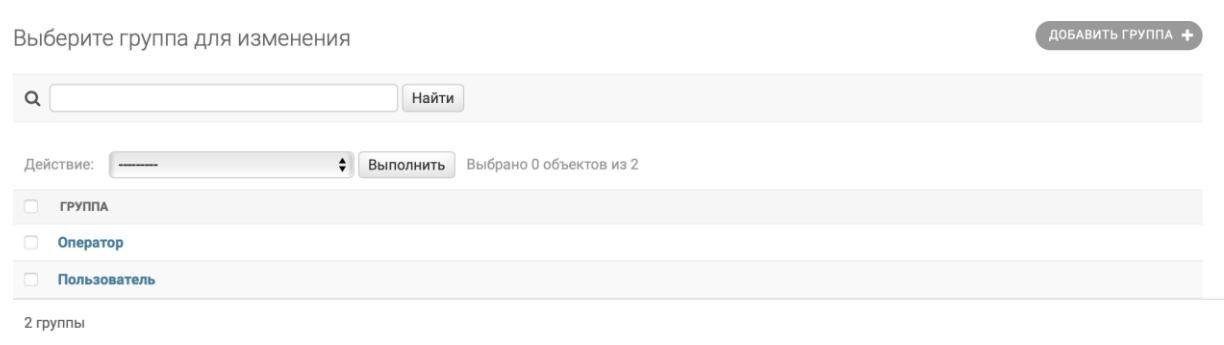


рис. 15. Группы в админке

ОПИСАНИЕ ТИПОВЫХ ЗАПРОСОВ К БД

Простые запросы:

1. `SELECT * FROM news_news` - получение всех новостей
2. `SELECT * FROM news_news WHERE id = 1` - получение новости с id равное 1
3. `INSERT INTO category_category (title) VALUES ('Футбол')` - добавление категории

Сложные запросы:

1. `SELECT * FROM news_news INNER JOIN category_catygory ON news_news.category = category_catygory.id` - вывод новостей с категориями
2. `SELECT * FROM news_news GROUP BY category` - вывод новостей с группировкой по категориям
3. `SELECT title FROM podcasts_podcasts INNER JOIN category_catygory ON news_news.category = category_catygory.id WHERE category_catygory.id = 1` - вывод названий подкастов у которых id категории равно 1
4. `UPDATE news_news SET content = 'Новый контент' WHERE id=1` - обновление контента новости
5. `DELETE FROM news_news WHERE category=1` - удалить все новости с категорией 1

РАЗРАБОТКА ВИЗУАЛЬНОЙ ЧАСТИ

Sports-news [Новости](#) [Мероприятия](#)

Категория ▾

Футбол

Футбол крутой
вапывап

Баскетбол

Баскетбол будет крутым
Контент

Футбол

Footbool
Footbool

Было разработано несколько страниц с помощью bootstrap. Реализован поиск и фильтрация.

ЗАКЛЮЧЕНИЕ

В рамках дисциплины “Инженерное проектирование” было реализованное приложение для управления спортивными мероприятиями и новости.

Репозиторий проекта - <https://github.com/mkhotsevich/sports-news>

Пароль от админки admin/admin

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

<https://docs.djangoproject.com/en/3.1/> - официальная документация Django

<https://www.django-rest-framework.org> - сайт фреймворка для REST-API

<https://www.postman.com> - сайт программы Postman

<https://webformyself.com> - сайт с различными туториалами по Django и python