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 «Курсовое проектирование»

Тема: «Конструктор фотоальбомов»

Выполнил:

Студент группы 211-322

Хасанов М. С.

_____

подпись, дата

Принял:

Старший преподаватель

Даньшина М.В.

_____

подпись, дата

Москва 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Моим курсовым проектом являлось создать конструктор фотоальбомов. Я поставил себе как задачу создать некий аналог Pinterest, только без социальной составляющей, альбомы, которые собирает пользователь, видны только ему. Собственно в самом проекте реализовано создание фотоальбомов, добавление в них картинок, а также редактирование информации о них (добавление описания, автора, категорий и подкатегорий) для дальнейшей возможной реализации поиска по сайту, а также добавление собственно самих категорий и подкатегорий (индивидуальных для каждого пользователя), добавление авторов (список которых общий для всех пользователей) и система авторизации.

Аналоги

Pinterest

Интернет-сервис для хостинга изображений, позволяющий пользователям добавлять изображения в свои альбомы (которые тут называются досками). Имеет достаточный функционал, позволяющий пользователям находить изображения на определенные темы благодаря категориям, а авторам делиться своим творчеством с остальным миром. Карта пути клиента весьма проста — пользователь ищет изображения на интересующую его тему, затем добавляет их в свой альбом, все. Сам сайт и мобильное приложение выполнены интуитивно понятно. Однако в отличии от моего проекта, Pinterest используется больше для того, чтобы искать что-то, что выложили другие пользователи, как правило идей для различных сфер жизни, мой же проект используется конкретно для добавления своих изображений и только.

Ссылка: https://www.pinterest.ru/

Juxtapost

Имеет весьма схожий с Pinterest функционал и карту пути клиента, однако гораздо менее user-friendly дизайн, по которому не так удобно ориентироваться, да и сам сайт оптимизирован гораздо хуже чем Pinterest.

Ссылка: http://www.juxtapost.com/

Photo-Pick

Наверно наиболее похожий на мой проект сайт, представленный в аналогах. В отличии от остальных здесь он не предоставляет возможности редактировать информацию о изображениях, а только добававлять их, однако он используется не для поиска идей, а именно что для создания фотоальбомов. Предоставляет достаточно обширный функционал (любое количество фотоальбомов, возможность ими поделиться, статистику альбомов, преобразование форматов и многое другое), интуитивно понятный и красивый дизайн, а также еще более простую карту пути клиента, состоящую лишь из того, чтобы добавить необходимые фотографии в альбом, после чего по необходимости поделиться им.

Ссылка: https://www.photo-pick.com/

Joomag

Интересный интернет-сервис, позволяющий создавать фотоальбомы, выглядящие как классические фотоальбомы, однако представленные в онлайн-формате, с возможностью добавлять в них такие интерактивные элементы как аудио и видео, уже не присущие классическим альбомам. Однако по сути карта пути клиента не сильно отличается от Photo-Pick, просто добавляются новые возможности, например вручную водить пользователей по своему альбому, как на презентации.

Ссылка: https://www.joomag.com/

Google Диск

Редко рассматривается как конструктор фотоальбомов, однако предоставляет для этого достаточный функционал. В него можно грузить фотографии, сортировать их по альбомам и даже более того у каждой фотографии будет информация о ней (время загрузки, описание и прочее). Альбомами (которые здесь правда будут являться папками) можно поделиться с другими пользователями и даже дать им доступ к их редактированию. На самом деле на практике возможно это самый популярный конструктор фотоальбомов, который просто не принято называть так.

Ссылка: https://drive.google.com/

ОСНОВНАЯ ЧАСТЬ

Проектирование

Функционал

За основу я взял функционал уже существующего онлайн-сервиса для создания фотоальбомов Pinterest. Для меня было важным реализовать такие же категории и подкатегории как и там. Но в отличии от Pinterest, у меня категории напрямую прописаны в описании изображений, а не являются своего рода еще одним альбомом для всех таких изображений с такой же категорией. К тому же у меня подкатегории зависят от категории, а не просто добавляются как набор слов. Идею с авторами я придумал сам, ведь на Pinterest и многих других подобных сайтах автором указан тот, кто выкладывает изображение, а не тот, кто его действительно сделал. Будь то фотограф или художник, если он не поставил вотемарку, он все равно может быть указан и не только в описании, которое по идее не предназначено для указания автора.

Позже, будь у меня больше времени, я планировал использовать информацию о изображении для реализации поиска.

Передо мной также стоял выбор как создавать альбомы, сразу заполненные или по нажатию кнопки создавать пустой альбом, а после заполнять его? В конце концов я посчитал, что второй вариант будет удобнее, хотя некоторые сервисы, в том числе встроенная в телефон галерея (по крайней мере на моем телефоне) автоматически удаляет пустые альбомы, что меня неоднократно напрягало, в связи с чем я и решил, что второй вариант будет удобнее.

Также я добавил возможность редактировать альбомы, назначая им имя, обложку и описание.

О пользовательской системе я думал сделать ли ее больше на подобии Pinterest с большой социальной составляющей, где ваши альбомы как правило делаются не только для вас или же сделать альбомы больше личными и все таки решил сделать это скорее галерей, где фотоальбомы создаются в первую очередь для пользователя. В будущем я планировал возможность давать к ним доступ, как в Google Диск-е, но увы мне не хватило времени.

Проектирование базы данных

Для проектирования базы данных я учел необходимый мне функционал и в итоге сделал базу данных из шести основных таблиц: альбом, изображение, автор, категория, подкатегория и пользователь. А также из некоторых побочных таблиц, необходимых для связи вида Many-to-Many. Саму схему можно посмотреть в приложениях (Рисунок 1 - Схема базы данных).

Дизайн

В дизайне я сильно не заморачивался, для меня важнее был не красивый дизайн, а понятный и минималистично приятный, поэтому он гораздо проще чем у аналогов и без излишеств. В свою очередь для описания изображения позаимствовал некоторые фишки у Pinterest, как например категории, выделенные овалом.

Ссылка на макет Figma: Макет

Разработка

Структура

Подготовившись к разработке, установив через Poetry (аналог pip) необходимые библиотеки в виртуальное окружение, я создал корневую папку проекта и стартовые файлы, прописав django-admin startproject mycourse. После чего для создания структуры для каждой отдельной таблицы базы данных создал отдельное приложение, прописав команду py manage.py startapp <название приложения>.

Модели

О создании моделей расскажу на примере альбомов. Для создания модели в файле models.py я создал одноименный класс Album, унаследованный от класса models.Model, предоставляемого самим Django. Создал переменные, приравнивая их соответствующим полям (пример: title = models.CharField, cover = models.ImageField и так далее). Всего были использованы следующие поля: CharField (короткий текст), TextField (длинный текст), DateField (дата), ImageField (изображение), ManyToManyField (поле, позволяющее Django реализовать соответствующий тип связи) и ForeignKey (поле, связанное с полем из другой таблицы как Many-to-One). Каждому полю был назначен параметр verbose_name, собственно имя поля, а также соответствующие типу поля параметры (max_length для CharField, upload_to для ImageField и т. д.).

Также модели была прописана функция __str__ и класс Meta, описывающие как модель будет отображаться в административной панели.

Представления

Представления прописывались в файле views.py, для каждого отдельного представления была создана отдельная функция.

Чтобы получить объекты какой либо модели, мне нужно было импортировать модели в соответствующий файл views, где для получения всех объектов данной модели необходимо было прописать (на примере модели Album) — Album.objects.all(), для получения какого либо определенного объекта – Album.objects.get(<параметр>), для получения некоторого набора объектов — Album.objects.filter(<параметр>). Однако я импортировал функцию get_object_or_404 из django.shortcuts и получал некоторые объекты так же с помощью нее, потому что если используя эту функцию, в случае, если объект не был получен, возвращается 404 ошибка.

Также я импортировал декоратор @login_required из django.contrib.auth.decorators и использовал его для многих функций-представлений. Этот декоратор не дает доступ к соответствующим функциям неавторизованным пользователям, перекидывая их на страницу авторизации в случае попытки получения доступа.

Для возвращения шаблонов я использовал две функции — render и HttpResponseRedirect. Первая просто рендерит необходимый шаблон, а вторая перекидывает нас на другой URL-адрес в другую функцию-представление, которая уже сама рендерит необходимый шаблон.

В большинстве моделей было возможно создавать новые объекты, а также удалять или редактировать уже существующие. Создание объекта модели я реализовывал либо через функцию create() (прописывая, например, Album.objects.create()), либо через формы, о которых скажу позже. Редактирование всегда реализовалось через формы, а удаление функцией delete().

Шаблоны

Шаблоны Django представляют собой HTML-код с вставками из Django. В моем проекте я использовал наследование шаблонов командой {% extends '<название>' %}, загрузку static файлов и их непосредственное применение, используя, соответственно, команды {% load static %} и {% static '<папка>/<файл>' %}.

Также я использовал шаблонный тег {{ <поле> }} для извлечения передаваемых данных, инструкцию {% for <элемент> in <элементы> %} … {% endfor %} для прохода по массиву объектов и условный тег {% if <условие> %} … {% else %} … {% endif %} для проверки условий.

Одним из самых частых шаблонных тегов был {% url '<имя>' <args> %}, благодаря которому и происходит переключение между различными страницами. Сам тег по имени находит нужный URL и переходит на него, после чего применяется соответствующее представление.

Адреса

Все адреса прописаны в различных файлах urls.py, также у проекта есть один такой файл, в который включаются все остальные файлы благодаря команде include().

В остальных же таких файлах прописаны адреса специально для приложения, в котором этот файл находится. Все адреса помещены в список urlpatterns.

У каждого адреса есть собственно сама строка адреса, которая отображается в браузере, представление, которое используется, когда мы переходим по этому адресу и имя, благодаря которому мы можем, например, перейти на этот адрес по тегу {% url %}.

Формы

Все изменения в существующих объектах, а также создание некоторых новых у меня зависят от форм. Формы в Django создаются довольно легко, все что надо это из django.forms импортировать ModelForm и унаследовать свой класс формы от этого класса. После чего либо вручную прописать поля формы, либо, как в моем случае, унаследовать эти поля от модели, а потом прописать поля, которые необходимо отображать данной форме. Это реализуется добавлением в класс формы класса Meta, в котором выбирается модель, из которой наследуется форма (model = <модель>) и поля (fields = [<поля>]).

После создания формы нам необходимо отобразить ее в шаблоне, для этого импортируем эту форму в файл с представлениями views.py и назначаем ее переменной (form=<форма>), после чего передаем в context-е шаблону и там уже используем шаблонный тег Django {{ <переменная с формой> }} для извлечения формы. Для получения данных из формы в файлах представлений views.py я использовал конструкцию:

If request.method==”POST”:

form.save()

else: …

Доступность

Для реализации доступности я подобрал достаточно констрастные цвета, а также назначил всем изображениям параметр alt, отображаемый в случае если изображение не прогрузилось. Для того, чтобы скрин-ридеры могли прочитать ссылки, им был назначен параметр aria-label.

Верстка

Так как шаблоны Django основываются на HTML, мне нужно было написать много HTM и CSS кода, который я писал в соответствии с БЭМ-методологией.

Современные CSS-решения:

1. Grid – сетка для списка альбомов и самого альбома.
2. Использование Flexbox для многих задач (например в списке навигации).
3. Использование препроцессора.
4. Использование псевдоэлементов и псевдоклассов.
5. Использование БЭМ-методологии.

ЗАКЛЮЧЕНИЕ

В результате я реализовал почти все что хотел сделать, у меня есть вполне успешно работающий конструктор фотоальбомов, в который можно загружать свои фотографии и назначать им описание.

Ссылка на GitHub: https://github.com/Namisami/mycourse

Ссылка на сервер: http://mycourse.std-1702.ist.mospolytech.ru/albums/

Скрины результатов будут также представлены в приложениях.

СПИСОК ИСТОЧНИКОВ

1. Веб-фреймворк Django (Python) - Изучение веб-разработки | MDN [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django (дата обращения: 07.07.2022).

2. Все публикации подряд / Хабр [Электронный ресурс]. – URL: https://habr.com/ru/all/ (дата обращения: 07.07.2022).

3. Хабр Q&A — вопросы и ответы [Электронный ресурс]. – URL: https://qna.habr.com/ (дата обращения: 07.07.2022).

4. Stack Overflow - Where Developers Learn, Share, & Build Careers [Электронный ресурс]. – URL: https://stackoverflow.com/ (дата обращения: 07.07.2022).

5. Django documentation | Django documentation | Django [Электронный ресурс]. – URL: https://docs.djangoproject.com/en/4.0/ (дата обращения: 07.07.2022).

6. Django.fun | Все о фреймворке Джанго и его библиотеках [Электронный ресурс]. – URL: https://django.fun/ (дата обращения: 07.07.2022).

ПРИЛОЖЕНИЯ

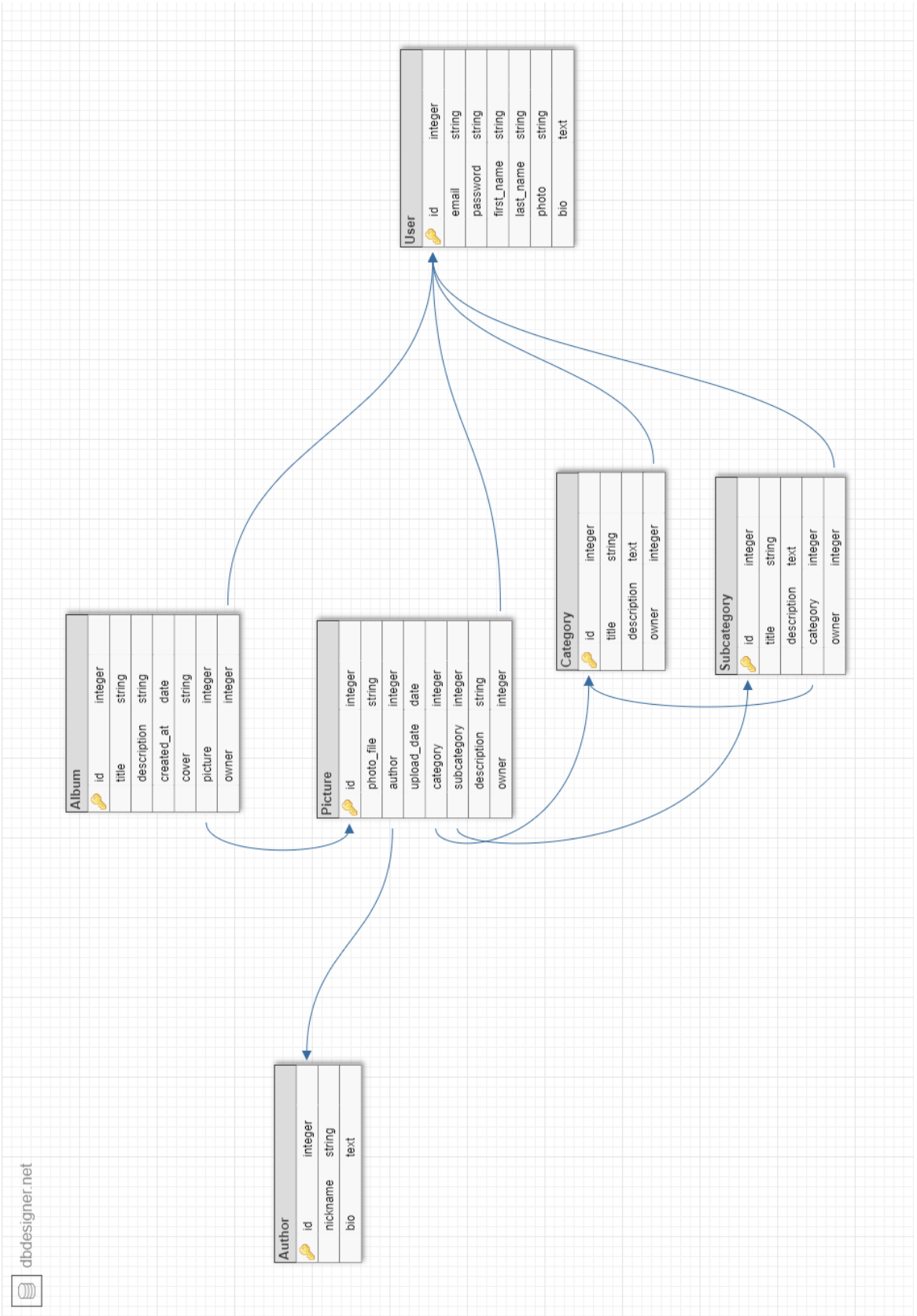

Рисунок 1 - Схема базы данных

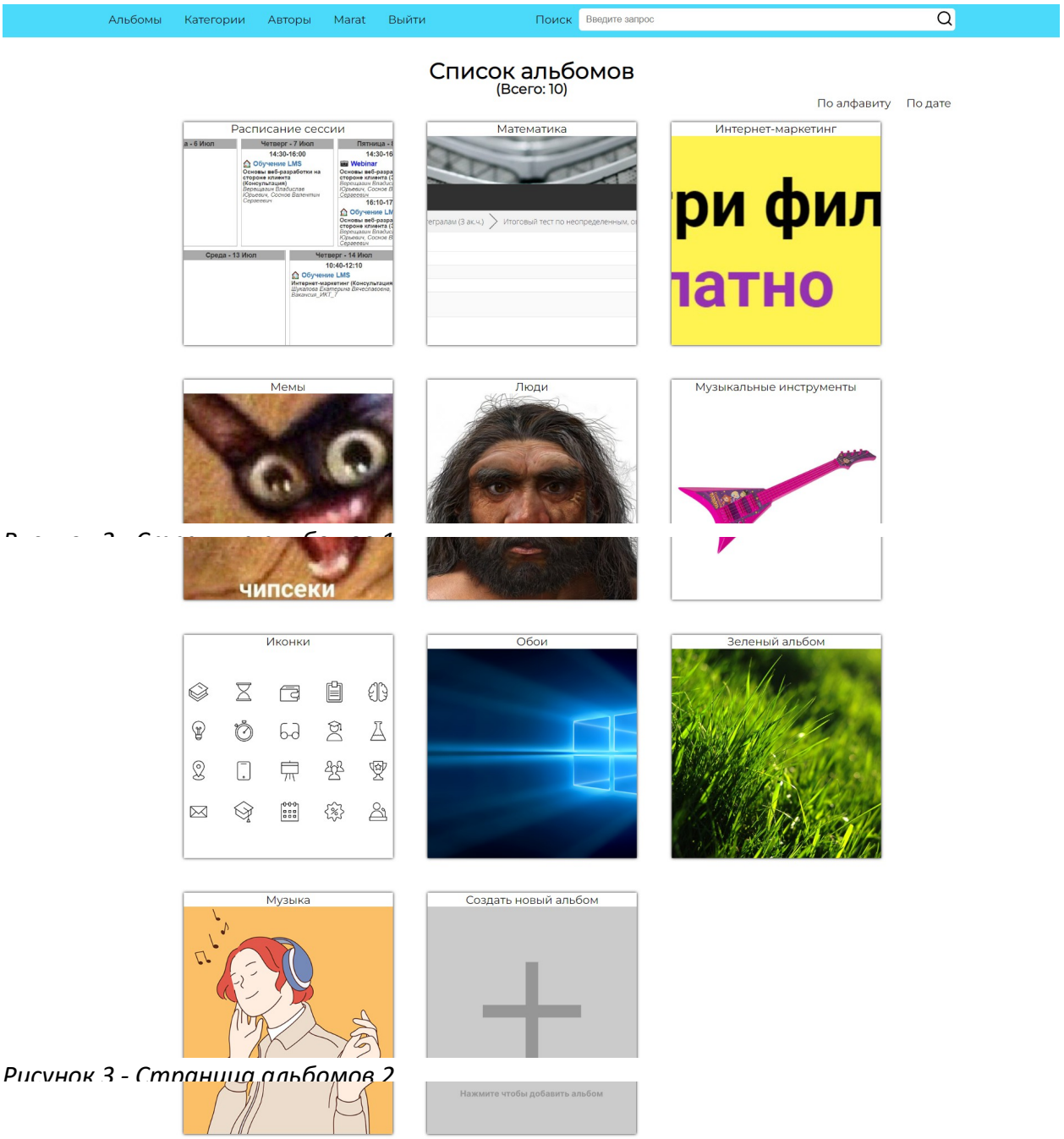

Рисунок 3 - Страница альбомов 2

Рисунок 4 - Страница альбомов 3

Рисунок 5 - Профиль

Рисунок 6 - Категории

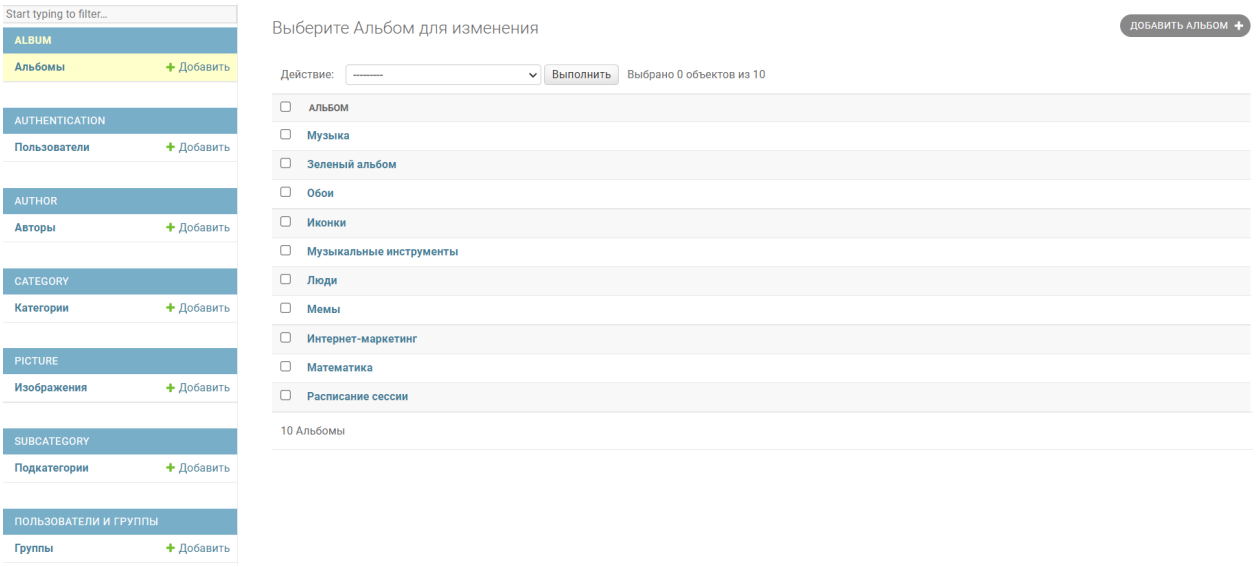

Рисунок 7 - Панель администратора

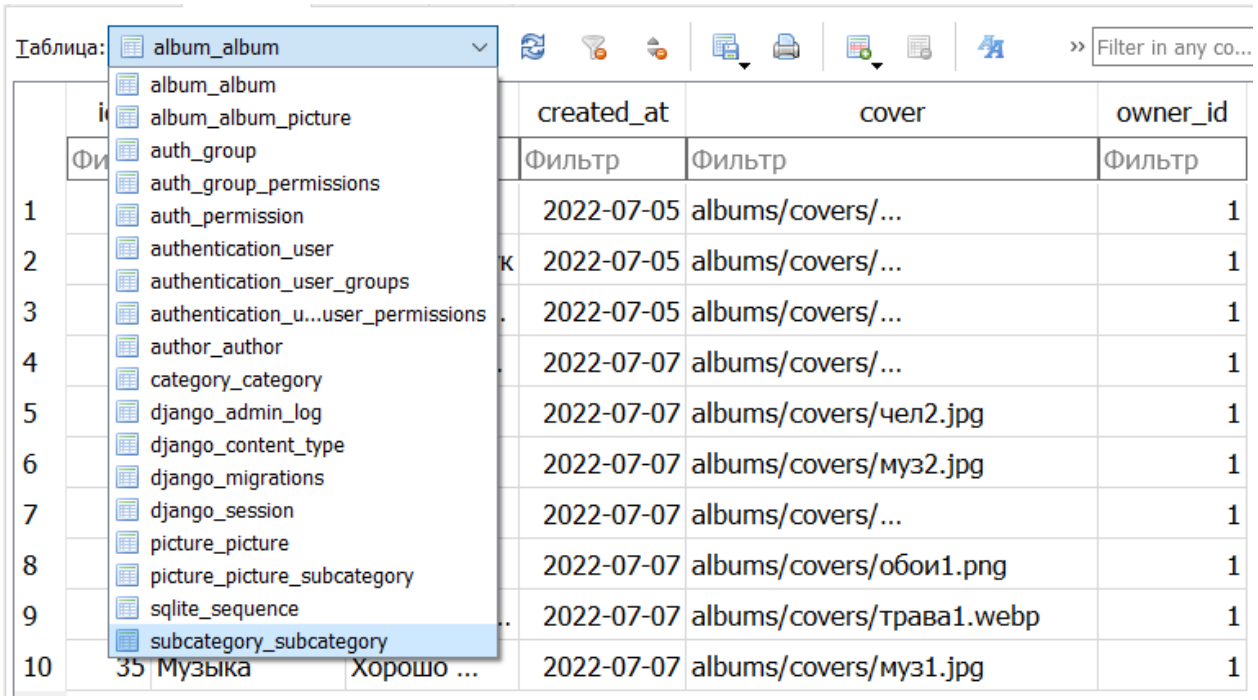

Рисунок 8 - База данных