*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети

**Отчет**

**по лабораторной работе №2**

**Разработка модели backend для приложения “Интернет-магазин компьютеров”**

Студент гр. ИУ6-13М\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Макаров А.В.

(Подпись, дата) (Фамилия И.О.)

Студент гр. ИУ6-12М\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Разуваев К.А.

(Подпись, дата) (Фамилия И.О.)

Студент гр. ИУ6-12М\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Колесников В.С.

(Подпись, дата) (Фамилия И.О.)

Студент гр. ИУ6-12М\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Пирмамедов М.Э.

(Подпись, дата) (Фамилия И.О.)

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Еремин О.Ю.

(Подпись, дата) (Фамилия И.О.)

Москва, 2019

**Задание:** разработать модель представления данных, ER-диаграмму, а также базу данных проекта.

Модель представления данных представляет собой промежуточное звено в модели MVC, выполняющее непосредственно работу с данными приложения.

Данная модель представлена на рисунке 1:

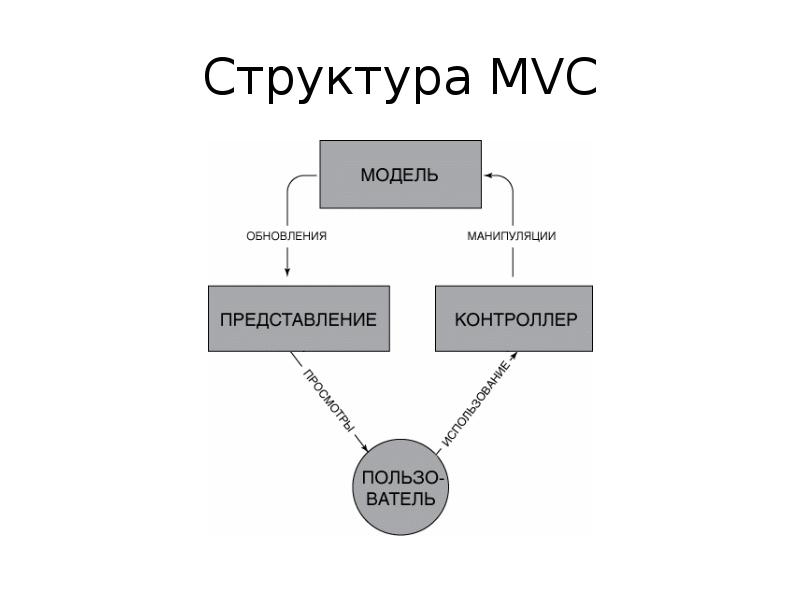


Рисунок 1 - Модель MVC.

**Контроллер (controller)** представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

**Представление (view)** - это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.

**Модель (model)** представляет класс, описывающий логику используемых данных.

В этой схеме модель является независимым компонентом - любые изменения контроллера или представления не затрагивают модель. Контроллер и представление являются относительно независимыми компонентами, и нередко их можно изменять независимо друг от друга.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью. И если нам, допустим, важна визуальная часть или фронтэнд, то мы можем тестировать представление независимо от контроллера. Либо мы можем сосредоточиться на бэкэнде и тестировать контроллер.

Для разработки приложения был выбран фреймворк Django.

Django — [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA" \o "Фреймворк) для [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на языке [Python](https://ru.wikipedia.org/wiki/Python" \o "Python), использующий шаблон проектирования [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Проект поддерживается организацией [Django Software Foundation](https://ru.wikipedia.org/wiki/Django_Software_Foundation" \o "Django Software Foundation).

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других (например, [Ruby on Rails](https://ru.wikipedia.org/wiki/Ruby_on_Rails" \o "Ruby on Rails)). Один из основных принципов фреймворка — [DRY](https://ru.wikipedia.org/wiki/DRY) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Don't repeat yourself).

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление, а презентационная логика Представления реализуется в Django уровнем Шаблонов. Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Разработанная ER-диаграмма представлена на следующем рисунке (рис. 2):

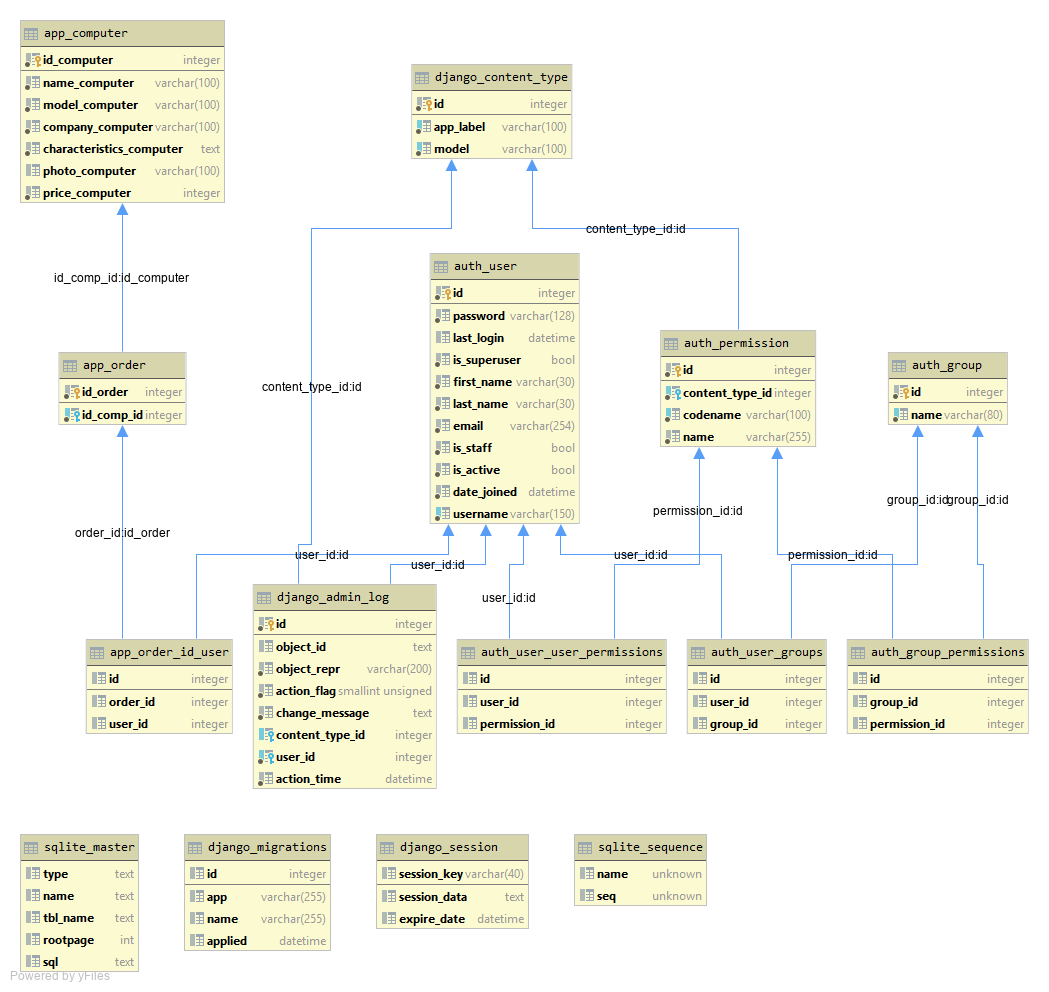


Рисунок 2 - ER-диаграмма.

База данных, разработанная для приложения, работает под управлением СУБД SQLite.

Модель данного приложения реализована в файле models.py, код которого приведён ниже:

**from** django.db **import** models  
**from** django.contrib.auth.models **import** User  
  
**class** Computer(models.Model):  
 id\_computer = models.AutoField(primary\_key=**True**)  
 name\_computer = models.CharField(max\_length=100, verbose\_name=**'Компьютер'**)  
 model\_computer = models.CharField(max\_length=100, verbose\_name=**'Модель'**)  
 company\_computer = models.CharField(max\_length=100, verbose\_name=**'Производитель'**)  
 price\_computer = models.IntegerField(verbose\_name=**'Цена'**)  
 characteristics\_computer = models.TextField(verbose\_name=**'Характеристики'**)  
 photo\_computer = models.ImageField(null=**True**, blank=**True**, verbose\_name=**'Фото'**)  
  
 **def** \_\_str\_\_(self):  
 **return** self.name\_computer  
  
 **class** Meta:  
 verbose\_name\_plural = **"Компьютеры"** verbose\_name = **"Компьютер"  
  
  
class** Order(models.Model):  
 id\_order = models.AutoField(primary\_key=**True**, verbose\_name=**'ID Заказа'**)  
 id\_comp = models.ForeignKey(Computer, on\_delete=models.CASCADE, verbose\_name=**'Название компьютера'**)  
 id\_user = models.ManyToManyField(User, verbose\_name=**'ID Пользователя'**)  
  
 **def** get\_users(self):  
 **return** [u.username **for** u **in** self.id\_user.all()]  
 get\_users.short\_description = **'Заказы пользователей'  
  
 class** Meta:  
 verbose\_name\_plural = **"Заказы"** verbose\_name = **"Заказ"**

В данной модели реализованы два класса – Computer и Order. Класс Computer необходим для работы с информацией о продаваемых продуктах. Содержит в себе следующие поля:

* id\_computer – уникальный идентификатор товара;
* name\_computer – имя товара;
* model\_computer – модель товара;
* company\_computer – производитель;
* price\_computer – цена;
* characteristics\_computer – характеристики;
* photo\_computer – фотография товара.

Класс Order необходим для работы с заказами пользователей. Содержит в себе следующие поля:

* id\_order – уникальный идентификатор заказа;
* id\_comp – уникальный идентификатор компьютера;
* id\_user – уникальный идентификатор пользователя.

Метод get\_users возвращает имя пользователя, сделавшего заказ.

Реализованная модель работает корректно и отвечает всем требования функционала приложения.

**Вывод:** в данной работе реализована модель backend-приложения для интернет-магазина компьютеров, построена ER-диаграмма, разработана база данных приложения. Также, получены навыки работы с фреймворком Django и языком программирования Python.