

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему

«Интернет-магазин музыкальных инструментов»

Студентка гр. 753503
К.А Ярохович

Руководитель
Удовин И.А.

Минск 2020

Содержание

Введение	3
1. Анализ предметной области	4
1.1 Технологии и средства разработки	4
1.2 Платформа Django	4
1.3 Язык Python	5
1.4 SQL	6
1.5 MySql и MySql Workbench	7
1.6 PyCharm	8
2. Интернет-магазин	10
2.1 Основные задачи приложения	10
3. Разработка приложения	11
3.1 Структура программы	11
3.2 Разработка функциональной модели	12
3.3 Разработка модели данных	14
3.4 Работа с базой данных	15
3.5 Результаты работы программы	16
Заключение	22
Список использованных источников	23
Приложение 1. Текст программы	24

Введение

В современном мире интернет-технологии играют огромную роль не только в жизни отдельных людей, но и жизненном цикле множества компаний. Во многом это стало возможно благодаря распространению и улучшению качества программного обеспечения.

Django – фреймворк для создания веб-приложений, позволяет относительно легко и быстро разработать приложение, соответствующее высоким ожиданиям заказчика.

Веб-приложения, в особенности интернет-магазины, призваны не только помочь предпринимателям в организации и управлении бизнесом, но и помочь покупателям получить доступ к огромному ассортименту товаров из любой точки мира.

1. Анализ предметной области

1.1 Технологии и средства разработки

Для реализации данного веб-приложения были использованы такие средства разработки как язык программирования Python в сочетании с веб-фреймворком Django, интегрированная среда разработки PyCharm.

1.2 Платформа Django

Django это свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC (в случае Django корректнее будет говорить MVT). Проект поддерживается организацией Django Software Foundation. Сайт на Django строится из одного или нескольких приложений, это одно из существенных архитектурных отличий этого фреймворка от некоторых других. Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных. Веб-фреймворк Django используется в сайтах, Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google и др.

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

Некоторые возможности Django:

- ORM, API доступа к БД с поддержкой транзакций;
- встроенный интерфейс администратора, с уже имеющимися переводами на многие языки;
- диспетчер URL на основе регулярных выражений;
- расширяемая система шаблонов с тегами и наследованием;
- система кеширования;
- интернационализация;
- подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
- «generic views» — шаблоны функций контроллеров;
- авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и проч.;

- система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
- библиотека для работы с формами (наследование, построение форм по существующей модели БД);
- встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение;

1.3 Язык Python

Python - высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой набор полезных функций. Python поддерживает структурное, обобщенное, объектно-ориентированное, функциональное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты. Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года.

Язык обладает чётким и последовательным синтаксисом, продуманной модульностью и масштабируемостью, благодаря чему исходный код написанных на Python программ легко читаем. Одной из интересных синтаксических особенностей языка является выделение блоков кода с помощью отступов (пробелов или табуляций), поэтому в Python отсутствуют операторные скобки begin/end, как в языке Pascal, или фигурные скобки, как в C. Дизайн языка Python построен вокруг объектно-ориентированной модели программирования. Реализация ООП в Python является элегантной, мощной и хорошо продуманной, но вместе с тем достаточно специфической по сравнению с другими объектно-ориентированными языками.

Богатая стандартная библиотека является одной из привлекательных сторон Python. Здесь имеются средства для работы со многими сетевыми протоколами и форматами Интернета, например, модули для написания HTTP-серверов и клиентов, для разбора и создания почтовых сообщений, для работы с XML и т. п. Набор модулей для работы с операционной системой позволяет писать кроссплатформенные приложения. Существуют модули для

работы с регулярными выражениями, текстовыми кодировками, мультимедийными форматами, криптографическими протоколами, архивами сериализации данных, поддержка юнит-тестирования и др.

1.4 SQL

SQL – это декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL считается языком программирования, в общем случае (без ряда современных расширений) не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

- создание в базе данных новой таблицы;
- добавление в таблицу новых записей;
- изменение записей;
- удаление записей;
- выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
- изменение структур таблиц.

Со временем SQL усложнился — обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) — и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

Язык SQL представляет собой совокупность операторов, инструкций, вычисляемых функций. Согласно общепринятому стилю программирования, операторы (и другие зарезервированные слова) в SQL обычно рекомендуется писать прописными буквами.

1.5 MySql и MySql Workbench

MySQL свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle. Изначальным разработчиком данной СУБД была шведская компания MySQL AB. В 1995 году она выпустила первый релиз MySQL. В 2008 году компания MySQL AB была куплена компанией Sun Microsystems, а в 2010 году уже компания Oracle поглотила Sun и тем самым приобрела права на торговую марку MySQL. Поэтому MySQL на сегодняшний день развивается под эгидой Oracle.

Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Для упрощения работы с сервером MySQL в базовый комплект установки входит такой инструмент как MySQL Workbench.

MySQL Workbench — инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL.

Возможности программы:

- Позволяет наглядно представить модель базы данных в графическом виде.
- Наглядный и функциональный механизм установки связей между таблицами, в том числе «многие ко многим» с созданием таблицы связей.
- Reverse Engineering — восстановление структуры таблиц из уже существующей на сервере БД (связи восстанавливаются в InnoDB, при использовании MyISAM — связи необходимо устанавливать вручную).
- Удобный редактор SQL запросов, позволяющий сразу же отправлять их серверу и получать ответ в виде таблицы.

- Возможность редактирования данных в таблице в визуальном режиме.

При разработке данного приложения была использована редакция MySQL Workbench под названием «Community Edition».

1.6 PyCharm

PyCharm — интегрированная среда разработки для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией JetBrains на основе IntelliJ IDEA. PyCharm — это кросс-платформенная среда разработки, которая совместима с Windows, MacOS, Linux. Эта среда динамично развивается, постоянно обновляется.

Основные преимущества PyCharm:

1. Понятный git:

Понятный интерфейс работы с git, история комментариев к коммитам, удобный экран решения конфликтов, отдельная панель Version control, аннотация строчек по их автору.

2. Простая организация проектов:

В данной среде разработки очень просто создавать проекты и открывать уже существующие: PyCharm буквально в два клика позволяет приступить к редактированию кода.

3. Удобный автокомплит:

Автокомплит работает мгновенно, не нужно вызывать его хоткеем. Работает не только в привычных местах, но и в шаблонах Django, для подстановки путей к файлам.

4. Тесная интеграция с Django:

В PyCharm присутствует удобная навигация между шаблонами. В последнем релизе также появилась отладка шаблонов Django.

5. Приятный интерфейс:

PyCharm вобрал в себя множество приятных характеристик: двустрочные вкладки, вертикальные направляющие для выравнивания текста, быстрый кодфолдинг, логичное расположение элементов интерфейса, благодаря чему он не выглядит перегруженным.

Для разработки данного приложения была использована версия PyCharm «Community Edition».

2. Интернет-магазин

Интернет-магазин — сайт, торгующий товарами посредством сети Интернет. Позволяет пользователям онлайн, в своём браузере или через мобильное приложение, сформировать заказ на покупку, выбрать способ оплаты и доставки заказа, оплатить заказ. При этом продажа товаров осуществляется дистанционным способом, и она накладывает ограничения на продаваемые товары.

Интернет-магазины обычно позволяют покупателям использовать функции «поиска», чтобы найти конкретные модели, бренды или предметы.

2.1 Основные задачи приложения

Рассмотрим задачи, которые представляется возможным решить благодаря данному веб-приложению.

Приложение представляет собой реализацию интернет-магазина по продаже музыкальных инструментов различных типов.

Приложение позволяет осуществлять демонстрацию товаров, оформление заказов, управление пользователями, поставщиками и товарами в принципе.

Данный магазин позволяет клиенту просматривать ассортимент продуктов и услуг фирмы, просматривать фотографии или изображения продуктов, а также информацию о технических характеристиках продукта и ценах.

Основной функционал реализован в системе регистрации пользователя, изменение регистрационных данных, возможности просматривать каталог товаров, отдельную страницу товара, сортировка товаров по различным критериям (тип, производитель, страна производства, стоимость), добавление/удаление товаров из корзины, оформление и просмотр активных заказов, удаление заказов, просмотр завершённых и отменённых заказов.

3. Разработка приложения

Для написания веб-приложения была использована среда разработки PyCharm и язык программирования Python.

Описание процесса создания приложения начнем со знакомства со структурой программы и её основными модулями.

3.1 Структура программы

Для начала рассмотрим структуру проекта, написанного с использованием Django. Эта структура является стандартной для проектов данного фреймворка.

Внешний каталог:

Внешний каталог содержит внутри себя все данные проекта по сути являясь просто контейнером для проекта, в котором содержатся общие файлы для настройки и взаимодействия с проектом, Python пакет проекта и папки с приложениями проекта.

manage.py:

Это утилита командной строки Django для административных задач. Кроме того, в каждом проекте Django автоматически создается manage.py. manage.py - это обертка вокруг django-admin.py

settings.py:

Файл настроек Django содержит всю конфигурацию установки Django. В файле настроек Django нет необходимости определять какие-либо настройки, если это не нужно. У каждой настройки есть разумное значение по умолчанию. Эти значения по умолчанию находятся в модуле `django / conf / global_settings.py`.

urls.py:

Чистая, элегантная схема URL-ов – это важная часть качественного приложения. Django позволяет проектировать URL-адреса различными способами, без ограничений фреймворка. Django также предоставляет метод для перевода URL на текущий язык.

За работу приложения отвечает 1 класс для взаимодействия с базой данных, 10 классов представляющих модели данных, 7 файлов представлений, в которых и реализована основная функциональность приложения и 18 файлов шаблонов + 1 файл стилей, которые определяют всю внешнюю составляющую приложения.

Связь с базой данных MySQL осуществляется при помощи внутреннего ORM Django. Взаимодействие происходит при помощи классов, представляющих модели данных.

Проектирование велось на основе стандартного шаблона проектирования для Django приложений паттерн MVT (Model-View-Template).

Основные элементы паттерна:

- URL dispatcher: при получении запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.
- View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).
- Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
- Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

3.2 Разработка функциональной модели

Функциональную модель предметной области представим в виде диаграммы вариантов использования в нотации UML, представляющей

систему в виде набора варианта использования и актеров, взаимодействующих с ними.

В рамках предметной области можно выделить двух актеров:

- администратор;
- пользователь;

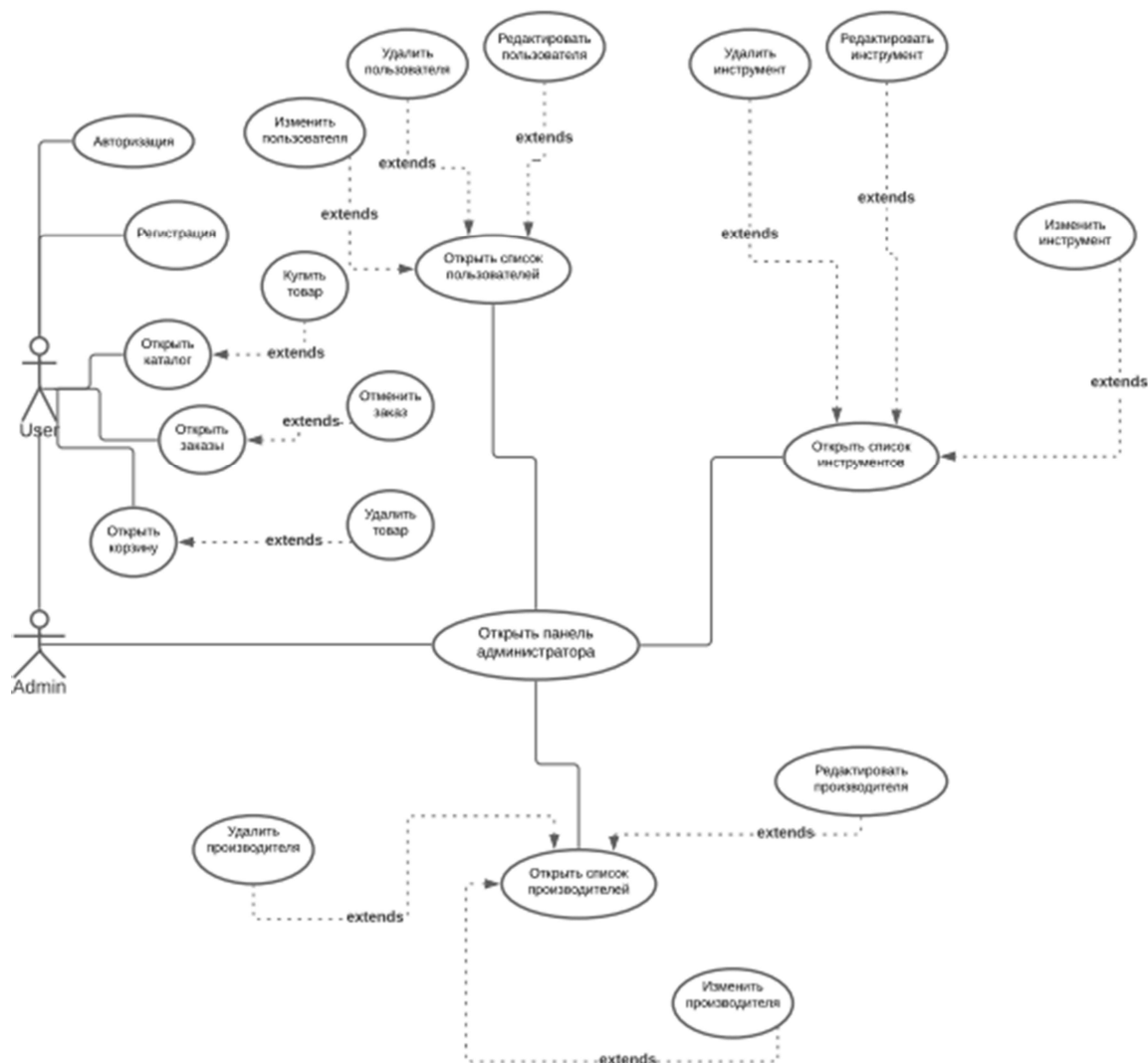


Рисунок 3.1. Диаграмма вариантов использования

Согласно приведенной диаграмме (рисунок 1), пользователь применяет следующие варианты использования:

- авторизация, восстановление пароля, регистрация;
- открыть инструмент, добавить в корзину, удалить из корзины, купить товар;
- открыть список заказов, просмотреть заказ, отменить заказ;

Администратор применяет варианты использования, доступные пользователю, а также дополнительные варианты использования – добавить, изменить, удалить пользователя; добавить, изменить, удалить инструмент, добавить, изменить, удалить поставщика, производителя и т.д.

3.3 Разработка модели данных

Разработка модели данных включает разработку классов, обеспечивающих взаимодействие с базой данных (рисунок 3.2).

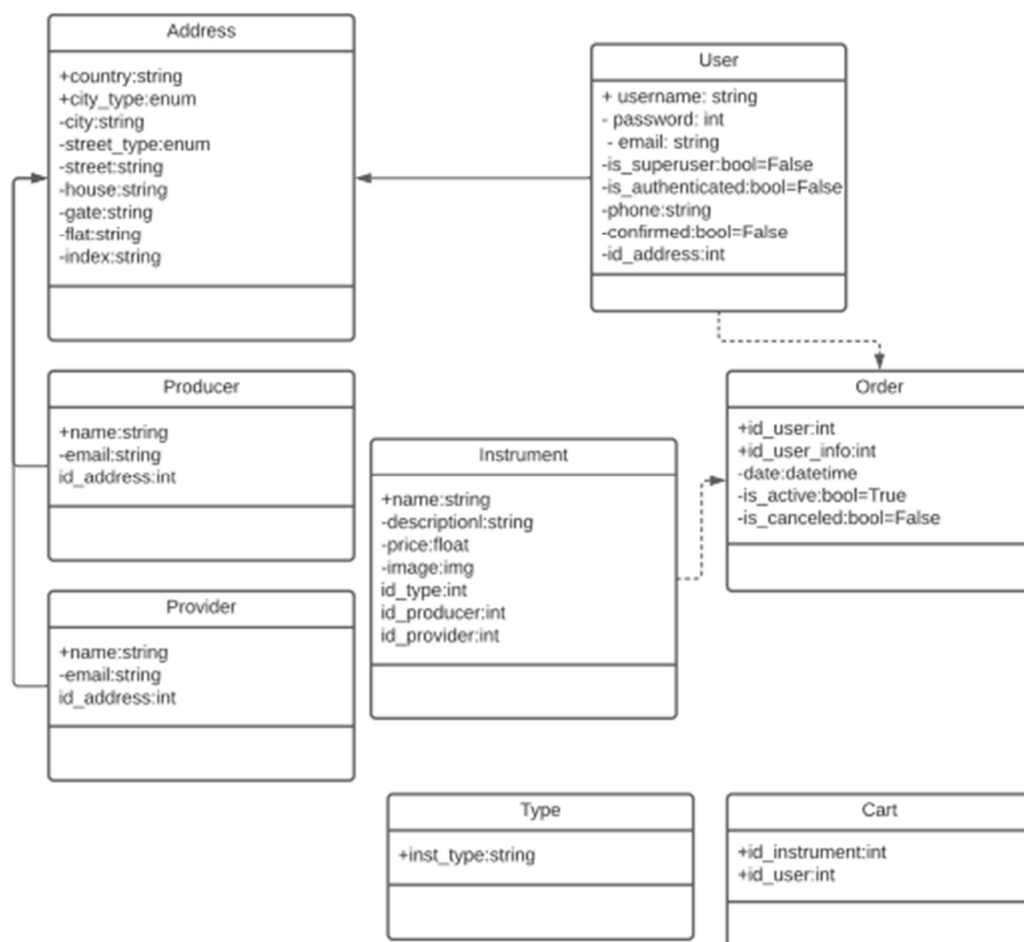


Рисунок 3.2. Диаграмма классов модели данных

Другие методы работы с БД реализованы в стандартной библиотеке `django.models`.

3.4 Работа с базой данных

Для хранения данных приложения была использована база данных MySQL. Было создано 10 таблиц для организации хранения данных. Ниже на рисунке 3.3 представлен полный список разработанных и реализованных таблиц + автоматически созданные сервисом аутентификации Django таблицы.

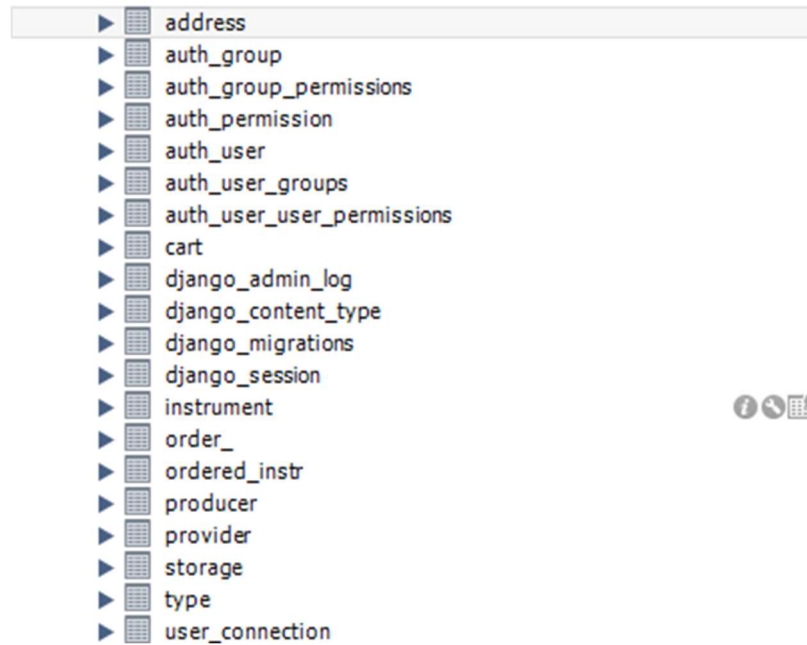


Рисунок 3.3 Список таблиц базы данных приложения musicalStore

На рисунке 3.4 представлена диаграмма базы данных приложения.

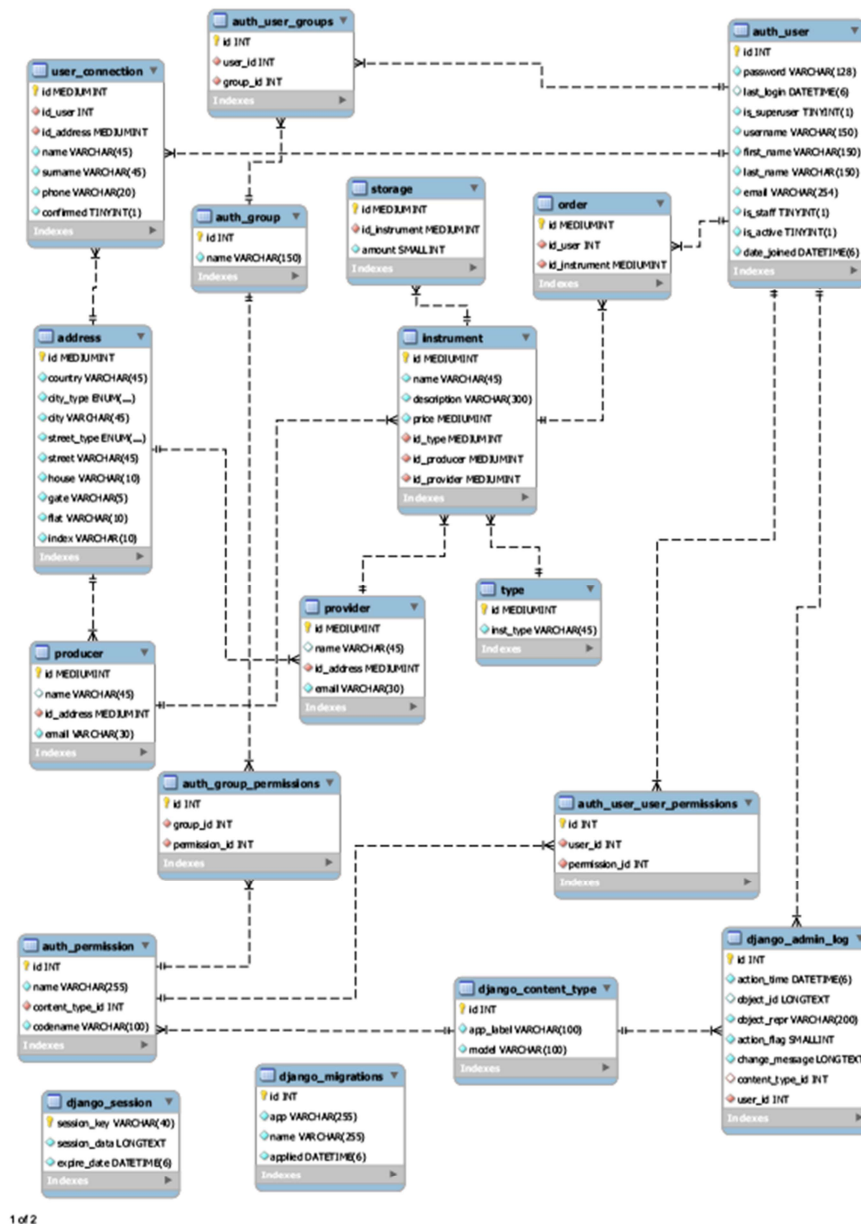


Рисунок 3.4 Диаграмма базы данных

3.5 Результаты работы программы

Структура файлов проекта, разработанная в соответствии с классами, выявленными в ходе проектирования, приведена на рисунке 3.5.

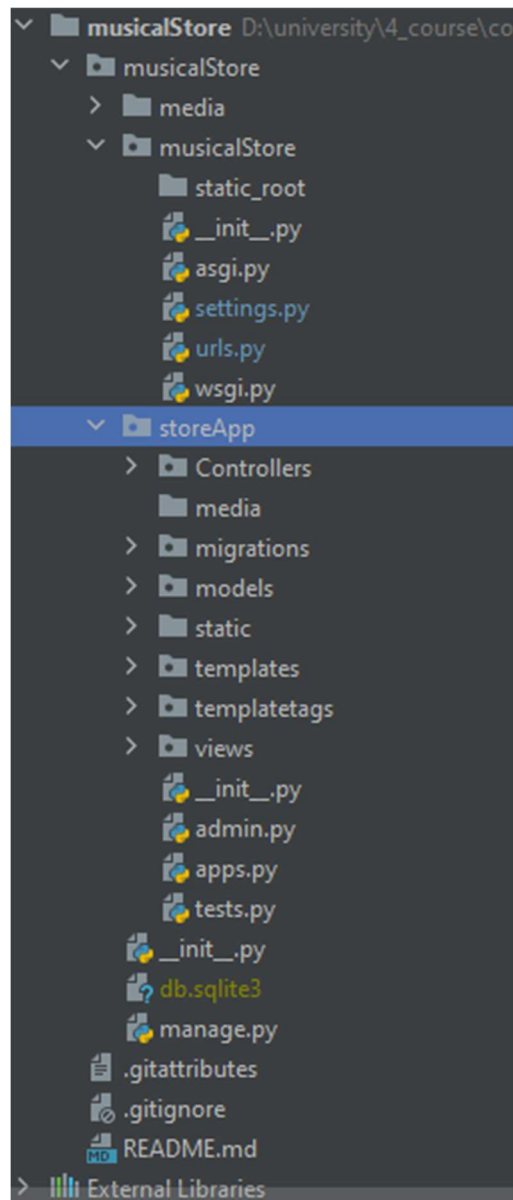


Рисунок 3.5. Проект приложения

Далее на скриншотах будут представлены интерфейс и основные функции приложения.

Начальный экран (стартовой страницей является каталог музыкальных инструментов) (рисунки 3.6-3.7).

Отдельная страница товара (рисунок 3.8).

Страница регистрации пользователя (рисунок 3.9). При регистрации пользователя происходит так же подтверждение по электронной почте (для отправки сообщений был использован сервис G-mail). На рисунке 3.11 представлен процесс подтверждения почты.

LOGIN CATALOG

Create Account

User info:

Login

First name

Last name

Phone

E-mail

Password

Рисунок 3.9 Страница регистрации пользователя

Страница авторизации (рисунок 3.10)

LOGIN CATALOG

Welcome Back!

Username

Password

Lost password?
Don't have account yet?

LOG IN

Рисунок 3.10 Страница авторизации

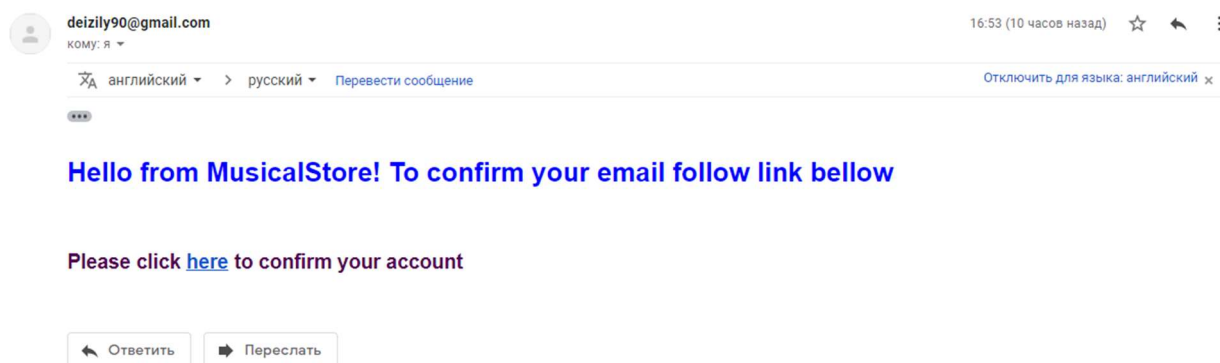


Рисунок 3.11 Письмо подтверждения

В личном профиле пользователя есть возможность просмотреть корзину с выбранными товарами, просмотреть активные заказы, а также отмененные и завершенные (рисунки 3.12 - 3.13).



Рисунок 3.12 Список заказов

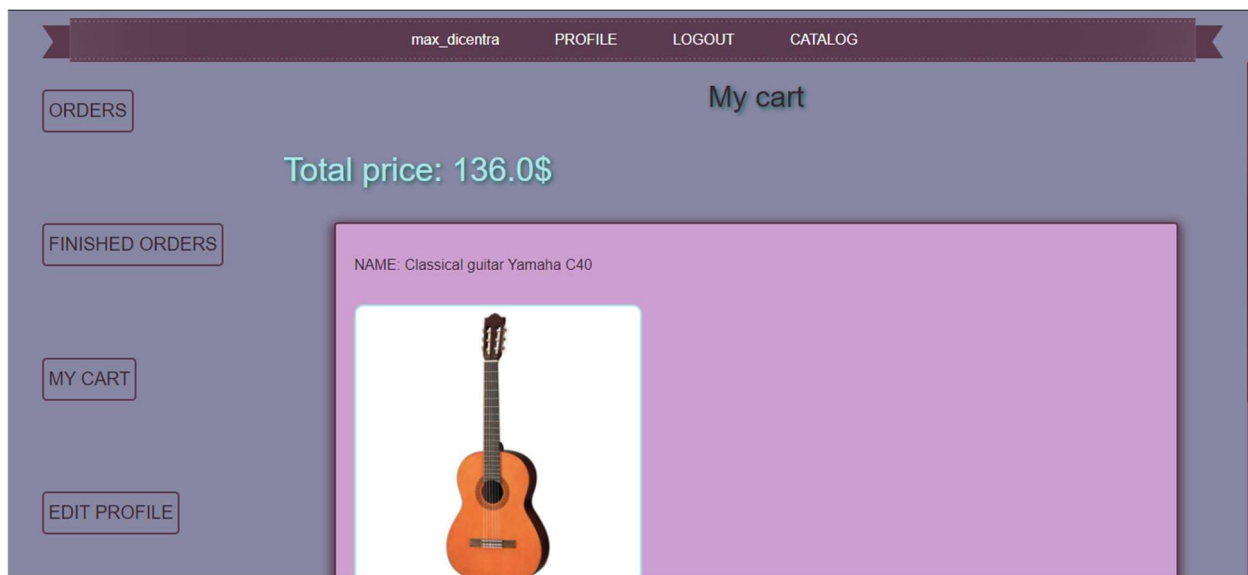


Рисунок 3.13 Корзина выбранных товаров

Помимо этого, пользователь имеет возможность редактирования данных, введенных при регистрации (рисунок 3.14)

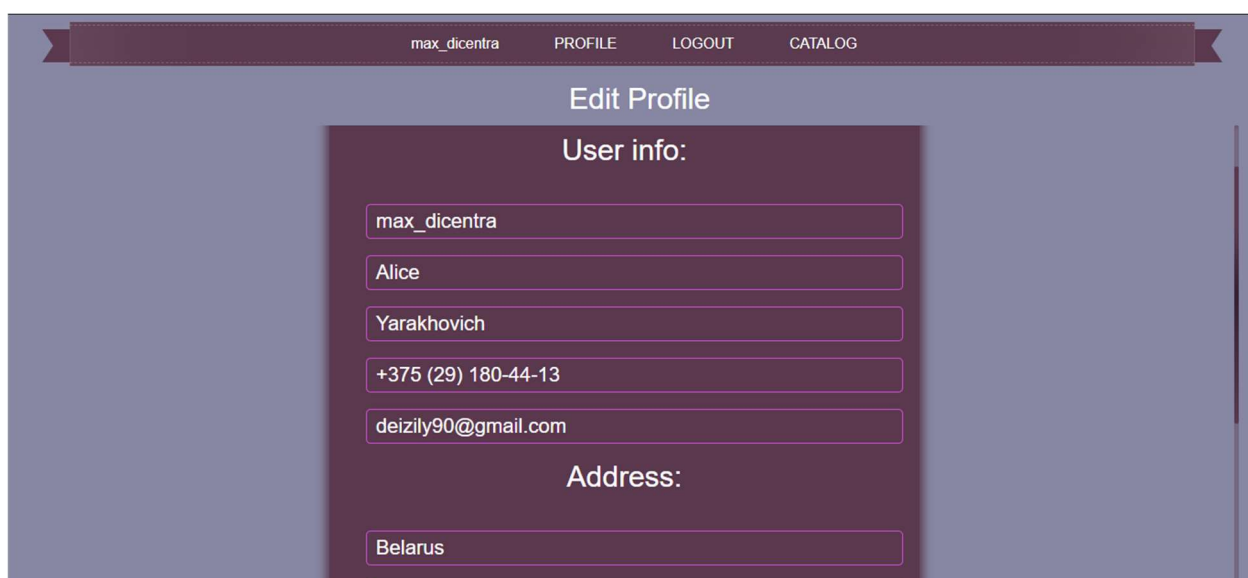
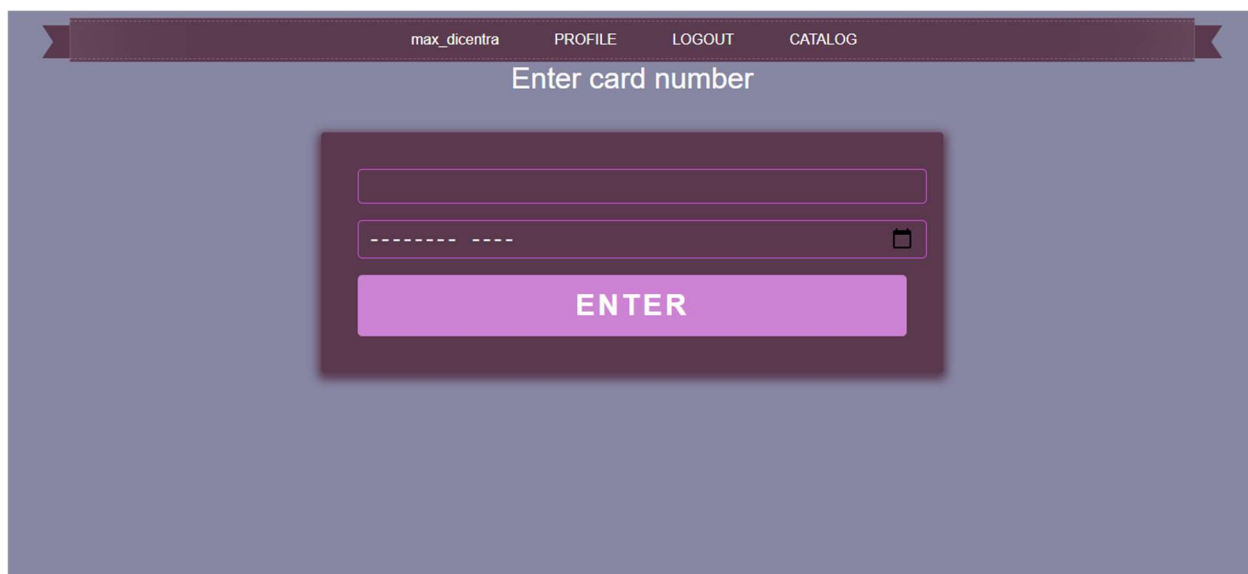


Рисунок 3.14 Страница редактирования личных данных

При оформлении заказа пользователю нужно ввести данные банковской карты, которые проверяются на валидность (рисунок 3.15).



The screenshot shows a web interface with a dark purple header bar containing links: max_dicentra, PROFILE, LOGOUT, and CATALOG. Below the header, the text "Enter card number" is displayed. The main content area features a dark purple box with a light purple border. Inside this box, there are two input fields: the first is for the card number, and the second is for the expiration date, indicated by a dashed line and a calendar icon. Below these fields is a large, light purple button labeled "ENTER".

Рисунок 3.15. Ввод и проверка данных карты

Для администратора доступна стандартная admin-панель предоставляемая Django (рисунок 3.16).

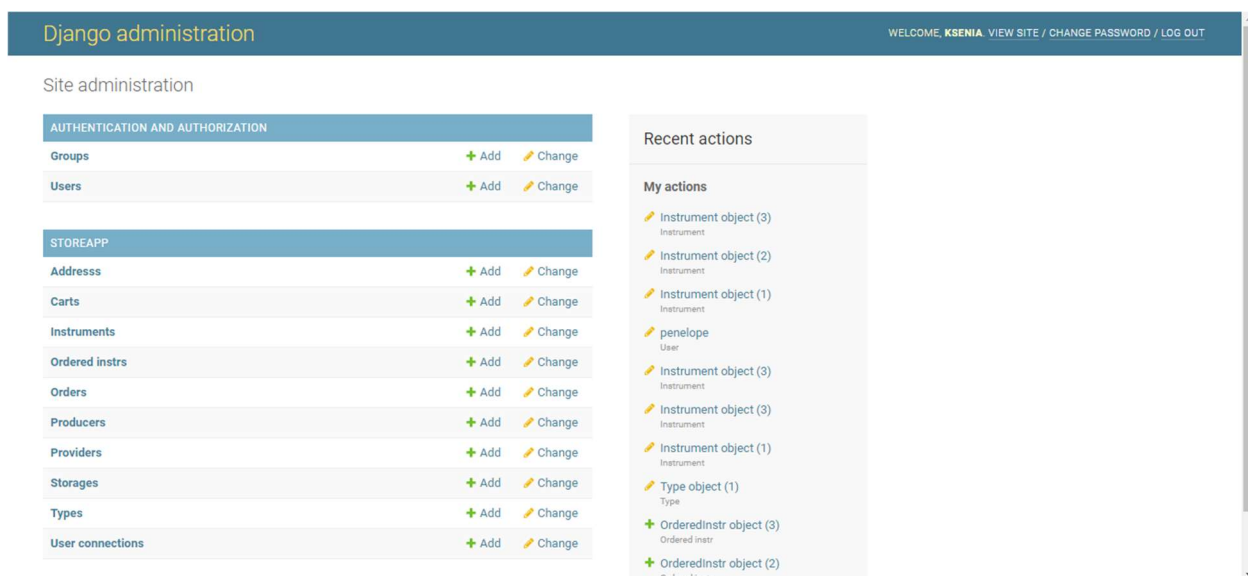


Рисунок 3.16 Панель админа

Заключение

В ходе выполнения курсовой работы были изучены основы работы в среде программирования PyCharm. В ходе работы были получены навыки написания приложения с использованием веб-фреймворка Django.

Была разработана база данных для веб-приложения и реализована работа с ней при помощи программных средств Django.

При разработке приложения были изучены и использованы основные возможности языка программирования Python.

Основная цель работы была выполнена, то есть было написано приложение «Интернет-магазин музыкальных инструментов». Стоит отметить, что приложение обладает простым и понятным пользователю интерфейсом.

Список использованных источников

1. Python 3 documentation — <https://docs.python.org/3/>
2. Django 3 documentation — <https://docs.djangoproject.com/en/3.0/>
3. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с
4. Тарасов С. В. СУБД для программиста. Базы данных изнутри. — М.: СОЛОН-Пресс, 2015. — 320 с.:

Приложение 1. Текст программы

Класс DBController:

```
import datetime

from storeApp.models import Address, Instrument, Order, Producer, Provider,\
    Storage, Type, UserConnection, Cart, OrderedInstr
from django.contrib.auth.models import User
from django.shortcuts import get_object_or_404
import pdb
from itertools import chain

class FullOrder:

    def __init__(self, order, instruments):
        self.order = order
        self.instruments = list(instruments)

class DBController(object):

    class UserKeys:
        ID = 'id'
        USERNAME = 'username'
        FIRST_NAME = 'first_name'
        LAST_NAME = 'last_name'
        PHONE = 'phone'
        PASSWORD = 'password'
        CONF_PASSWORD = 'conf_password'
        EMAIL = 'email'
        ADDRESS = 'address'
        COUNTRY = 'country'
        CITY_TYPE = 'city_type'
        CITY = 'city'
        STREET = 'street'
        FLAT = 'flat'
        GATE = 'gate'
        HOUSE = 'house'
        STREET_TYPE = 'street_type'
        INDEX = 'index'

    __instance = None

    def __new__(cls):
        if not DBController.__instance:
            cls.__instance = super(DBController, cls).__new__(cls)
        return cls.__instance

    def CreateUser(self, user_data):
        user =
User.objects.create_user(username=user_data[self.UserKeys.USERNAME],
first_name=user_data[self.UserKeys.FIRST_NAME],
last_name=user_data[self.UserKeys.LAST_NAME],
                        email=user_data[self.UserKeys.EMAIL],
password=user_data[self.UserKeys.PASSWORD])
        address = self.CreateAddress(user_data)
        user.save()
```



```

address.save()
user_profile = UserConnection(phone=user_data[self.UserKeys.PHONE],
                               id_user=user,
                               id_address=address,
                               confirmed=False)

user_profile.save()

return user

def EditUser(self, user_data, user_id):
    user = User.objects.get(pk=user_id)
    user.username = user_data[self.UserKeys.USERNAME]
    user.first_name = user_data[self.UserKeys.FIRST_NAME]
    user.last_name = user_data[self.UserKeys.LAST_NAME]
    user.email = user_data[self.UserKeys.EMAIL]

    user_inf = UserConnection.objects.get(id_user=user.id)
    user_inf.phone = user_data[self.UserKeys.PHONE]

    self.EditAddress(user_data, user_inf)

    user.save()
    user_inf.save()

def CreateAddress(self, user_data):
    address = Address(country=user_data[self.UserKeys.COUNTRY],
                     city_type=Address.CityType.CIT,
                     city=user_data[self.UserKeys.CITY],
                     street_type=Address.StreetType.STR,
                     street=user_data[self.UserKeys.STREET],
                     house=user_data[self.UserKeys.HOUSE],
                     gate=user_data[self.UserKeys.GATE],
                     flat=user_data[self.UserKeys.FLAT],
                     index=user_data[self.UserKeys.INDEX])

    address.save()
    return address

def EditAddress(self, address_data, user_inf):
    address = user_inf.id_address

    address.country = address_data[self.UserKeys.COUNTRY]
    address.city_type = Address.CityType.CIT
    address.city = address_data[self.UserKeys.CITY]
    address.street_type = Address.StreetType.STR
    address.street = address_data[self.UserKeys.STREET]
    address.house = address_data[self.UserKeys.HOUSE]
    address.gate = address_data[self.UserKeys.GATE]
    address.flat = address_data[self.UserKeys.FLAT]
    address.index = address_data[self.UserKeys.INDEX]

    address.save()

def AddToCart(self, id_instrument, user_id):

    instrument = Instrument.objects.get(pk=id_instrument)
    user = get_object_or_404(User, pk=user_id)

    cart = Cart(id_instrument=instrument, id_user=user)
    cart.save()

def DeleteFromCart(self, cart_id):
    # pdb.set_trace()
    cart = Cart.objects.get(pk=cart_id)

```

```

        cart.delete()

    def GetUserCart(self, user_id):
        user = get_object_or_404(User, pk=user_id)
        instruments_con = Cart.objects.all().filter(id_user=user)

        total_price = 0
        instruments_list = []
        for i in range(len(instruments_con)):
            instruments_list.append(instruments_con[i].id_instrument)
            total_price += float(instruments_list[i].price)

        return instruments_list, instruments_con, total_price

    def GetOrders(self, user):
        orders = Order.objects.all().filter(id_user=user.id, is_active=True)
        result_list = []

        for order in orders:

            intruments = OrderedInstr.objects.all().filter(id_order=order.id)
            result_list.append(FullOrder(order, intruments))

        return result_list

    def GetCanceledOrders(self, user):
        orders = Order.objects.all().filter(id_user=user.id, is_active=False)
        result_list = []

        for order in orders:
            intruments = OrderedInstr.objects.all().filter(id_order=order.id)
            result_list.append(FullOrder(order, intruments))
        # pdb.set_trace()
        return result_list

    def CreateOrder(self, user):

        cart_instr = Cart.objects.all().filter(id_user=user.id)
        user_info = UserConnection.objects.get(id_user=user.id)
        order = Order(id_user=user, id_user_info=user_info,
date=datetime.datetime.now().date(),
                    is_active=True, is_canceled=False)
        order.save()

        for cart in cart_instr:
            new_ord = OrderedInstr(id_instrument=cart.id_instrument,
id_order=order)
            new_ord.save()
            cart.delete()
        # ввод данных карты

    def CancelOrder(self, order_id):
        order = Order.objects.get(pk=order_id)
        order.is_canceled = True
        order.is_active = False
        order.save()
        # return money?

    def FinishOrder(self, order_id):
        order = Order.objects.get(pk=order_id)
        order.is_canceled = False
        order.is_active = False
        order.save()

```

```

    def GetInstruments(self, type, name, producer_country, producer,
low_price, high_price, sort_by):

        instruments = Instrument.objects.all().filter(id_type=type.id)

        # pdb.set_trace()
        if producer: # +++
            if producer.lower() != 'select':
                producer_obj = Producer.objects.get(name=producer)
                instruments = instruments.filter(id_producer=producer_obj)

        if producer_country:
            if producer_country.lower() != 'select':
                # pdb.set_trace()

                producers =
Producer.objects.all().filter(id_address__country=producer_country)
                instruments = instruments.filter(id_producer__in=producers)

        # if high_price:
        #     instruments = instruments.filter(price__lte=high_price,
price__gte=low_price)

        if name:
            instruments = instruments.filter(name__contains='name')

        if sort_by:
            if sort_by != 'select':
                if sort_by == 'high to low':
                    instruments = instruments.order_by('-price')
                elif sort_by == 'low to high':
                    instruments = instruments.order_by('price')

        return instruments

    def GetTypes(self):
        return Type.objects.all()

    def GetProdAndContr(self):
        producers_obj = Producer.objects.all()

        countries = []
        producers = []
        for prod in producers_obj:
            countries.append(prod.id_address.country)
            producers.append(prod.name)

        return producers, countries

    def GetProducer(self, name):
        return Producer.objects.get(name=name)

    def GetPrices(self, type):
        instruments = Instrument.objects.all().filter(id_type=type.id)
        instruments_max = instruments.order_by('price')[0]
        instruments_min = instruments.order_by('-price')[0]

        return instruments_max.price, instruments_min.price

    def GetUser(self, user_id):
        return User.objects.get(pk=user_id)

```

```
def GetUserData(self, user_id, user):

    user_info = UserConnection.objects.get(id_user=user_id)
    address = user_info.id_address
    return user_info, address
```

profile.py view:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from storeApp.Controllers.DBController import DBController

db = DBController()

@login_required
def profile(request):

    user_cart, cart_info, total_price = db.GetUserCart(request.user.id)
    show = 1
    empty = False
    if len(user_cart) == 0:
        empty = True

    if request.POST.get('show_orders', False):
        orders = db.GetOrders(request.user)
        show = 2
        return render(request, 'storeApp/profile.html', {
            'cart_info': None,
            'user_cart': None,
            'empty': True,
            'canceled_orders': None,
            'orders': orders,
            'show': show
        })
    elif request.POST.get('show_canceled_orders', False):
        canceled_orders = db.GetCanceledOrders(request.user)
        show = 3
        return render(request, 'storeApp/profile.html', {
            'cart_info': None,
            'user_cart': None,
            'empty': True,
            'canceled_orders': canceled_orders,
            'orders': None,
            'show': show,
        })

    return render(request, 'storeApp/profile.html', {
        'cart_info': cart_info,
        'total_price': total_price,
        'user_cart': user_cart,
        'empty': empty,
        'canceled_orders': None,
        'orders': None,
        'show': show,
    })

@login_required
def DeleteFromCart(request, cart_id):
    db.DeleteFromCart(cart_id)
    redirect_to = request.POST['next']
```

```
return HttpResponseRedirect(redirect_to)
```

catalog.py view:

```
from django.http import HttpResponseRedirect
from storeApp.models.instrument import Instrument
from django.template import loader
from storeApp.Controllers.DBController import DBController
from django.urls import reverse
import pdb

db = DBController()

def show_catalog(request):

    types = db.GetTypes()
    show_types = True
    if request.POST.get('show_types', True) is not True:

        type = GetType(request, types)
        producers, countries = db.GetProdAndContr()
        highest_price, lowest_price = db.GetPrices(type)
        sorts = ['high to low', 'low to high']

        if request.POST.get('filter', False):

            sort_by = request.POST.get('sort_by', 'select')
            producer_country = request.POST.get('countries_s', 'select')
            producer = request.POST.get('producers_s', 'select')
            low_price, high_price = GetPriceLimits(request, highest_price,
lowest_price)
            name = request.POST.get('name', None)

            instruments_list = db.GetInstruments(type, name,
producer_country, producer, low_price, high_price, sort_by)
            template = loader.get_template('storeApp/catalog.html')
            context = {'instruments_list': instruments_list,
                        'show_types': False,
                        'type': type,
                        'producer': producer,
                        'producer_country': producer_country,
                        'low_limit': low_price,
                        'high_limit': high_price,
                        'sort_by': sort_by,
                        'countries': countries,
                        'producers': producers,
                        'highest_price': highest_price,
                        'lowest_price': lowest_price,
                        'sorts': sorts,
                        }
            return HttpResponseRedirect(template.render(context, request))

        instruments_list = db.GetInstruments(type, None, None, None, None,
None, None)
        template = loader.get_template('storeApp/catalog.html')
        context = {'instruments_list': instruments_list,
                    'show_types': False,
                    'type': type,
                    'producer': request.POST.get('producers_s', 'select'),
                    'producer_country': request.POST.get('countries_s',
'select'),
```

```

        'countries': countries,
        'producers': producers,
        'low_limit': lowest_price,
        'high_limit': highest_price,
        'sort_by': request.POST.get('sort_by', 'select'),
        'highest_price': highest_price,
        'lowest_price': lowest_price,
        'sorts': sorts,
    }
    return HttpResponse(template.render(context, request))

template = loader.get_template('storeApp/catalog.html')
context = {'instruments_list': None,
           'types': types,
           'show_types': show_types,
           }
return HttpResponse(template.render(context, request))

def Search(request):
    redirect_to = request.POST['next']
    return HttpResponseRedirect(redirect_to)

def SearchWithFilters(request):
    redirect_to = request.POST['next']
    return HttpResponseRedirect(redirect_to)

def SearchByName(request):
    redirect_to = request.POST['next']
    return HttpResponseRedirect(redirect_to)

def addToCart(request, instrument_id, user_id):
    db.AddToCart(instrument_id, user_id)
    redirect_to = request.POST['next']
    return HttpResponseRedirect(redirect_to)

def GetType(request, types):
    for type in types:
        if request.POST.get(type.inst_type, False):
            # pdb.set_trace()
            return type
    return None

def GetPriceLimits(request, highest_price, lowest_price):
    low_limit = request.POST.get('low_limit', lowest_price)
    high_limit = request.POST.get('high_limit', highest_price)

    return low_limit, high_limit

```

