Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 805 «Математическая кибернетика»

**Отчёт по курсовому проекту**

**по курсу «Базы данных»**

**Тема: «Интернет-магазин музыкальных инструментов»**

**(СУБД SQLite)**

Подготовили:

студенты группы М8О-303Б-21

Жилин М.Д.

Зверева Е.Л.

Субботина М.А.

Жалялетдинов М.Р.

Проверила:

Кузнецова С.В.

Москва, 2023

# **Техническое задание**

**Цель**: Разработать веб-сервис для выбора музыкальных инструментов и совершения покупок в интернет магазине

**Особенности**: Cервис ориентирован на обычных пользователей (покупателей)

**Описание работ:**

* Сбор требований
* Проработка сценариев использования
* Концептуальное и логическое проектирование базы данных
* Реализация пользовательского интерфейса
* Разработка серверной части
  + Создание и настройка базы данных
  + Разработка API
  + Настройка роутов
* Тестирование

# Требования

## 

## Функциональные требования

* Регистрация и авторизация
* Оформление заказа
* Просмотр товаров

## Нефункциональные требования

СУБД: SQLite

Используемые языки: Python3.10, HTML, CSS, JS

Средства разработки: PyCharm, SQLiteStudio (3.4.0), Visual Studio Code

Поддержка: desktop версия браузеров

Дополнительно: на данном этапе веб-приложение является MVP, но должно обеспечивать масштабируемость и, по возможности, безопасность

# **Use case**

Предусмотрены следующие пользовательские сценарии:

* Переход на главную страницу сайта
* Регистрация на сайте
* Аутентификация
* Получение каталога товаров
* Получение информации о товаре
* Добавление товара в корзину
* Оформление заказа
* Просмотр истории заказов или заказов, находящихся “в доставке”

Сценарий №1: Переход на главную страницу сайта

Цель: перейти на главную страницу

Актор: любой пользователь

Исходная страница: любая

Результат: клиент переходит на главную страницу

Шаги:

1. Пользователь нажимает на логотип в левом верхнем углу страницы
2. Сайт переводит пользователя на главную страницу

Сценарий №2: Регистрация

Цель: зарегистрироваться на сайте

Актор: незарегистрированный пользователь

Исходная страница: любая

Результат: клиент проходит регистрацию, его данные сохраняются в базе данных

Шаги:

1. Пользователь нажимает кнопку “Вход”
2. Сайт выводит форму входа
3. Пользователь нажимает на кнопку “Регистрация”
4. Сайт отображает форму регистрации
5. Пользователь вводит корректные данные, в частности электронную почту, пароль и имя, и нажимает кнопку “Зарегистрироваться”
6. Сайт отправляет данные на сервер
7. Сервер валидирует данные и проверяет введенный email на существование в базе данных
8. Сервер отправляет ответ на сайт
9. В зависимости от ответа сервера, сайт выводит пользователю информацию
   1. Если пользователь ввел некорректные данные, то сайт отобразит ошибку “Проверьте корректность данных”
   2. Если пользователь ввел существующий в базе данных email, то сайт отобразит ошибку “Пользователь с данным email уже существует”
   3. В оставшихся случаях, сайт отобразит сообщение “Успешная регистрация”

Сценарий №3: Аутентификация

Цель: войти в аккаунт

Актор: не аутентифицированный пользователь

Исходная страница: любая

Результат: клиент проходит аутентификацию

Шаги:

1. Пользователь нажимает кнопку “Вход”
2. Сайт выводит форму входа
3. Пользователь вводит корректные данные, в частности электронную почту и пароль, и нажимает кнопку “Войти”
4. Сайт отправляет данные на сервер
5. Сервер валидирует данные
6. Сервер отправляет ответ на сайт
7. В зависимости от ответа сервера, сайт выводит пользователю информацию
   1. Если пользователь ввел некорректные данные, то сайт отобразит ошибку “Проверьте корректность данных”
   2. Если пользователь ввел не существующий в базе данных email, то сайт отобразит ошибку “Пользователя с данным email не существует”
   3. В оставшихся случаях, сайт отобразит сообщение “Вход выполнен успешно”

Сценарий №4: Получение каталога товаров

Цель: получение каталога товаров

Актор: любой пользователь

Исходная страница: главная страница

Результат: клиент проходит аутентификацию

Шаги:

1. Пользователь наводит курсор на название категории музыкального инструмента в левой части страницы
2. Сайт отображает список подкатегорий музыкальных инструментов, принадлежащих выбранной категории
3. Пользователь нажимает на название необходимой подкатегории музыкального инструмента
4. Сайт переводит пользователя на страницу каталога музыкальных инструментов, соответствующих выбранной подкатегории
5. Далее пользователь может взаимодействовать с элементами взаимодействия

Сценарий №5: Получение информации о товаре

Цель: получение информации о товаре

Актор: любой пользователь

Исходная страница: главная страница, страница каталога

Результат: пользователь переходит на страницу товара

Шаги:

1. Пользователь нажимает на название музыкального инструмента в карточке товара
2. Сайт переводит пользователя на страницу товара

Сценарий №6: Добавление товара в корзину

Цель: Добавить музыкальный инструмент в корзину товаров

Актор: пользователь

Исходная страница: каталог, страница товара

Результат: музыкальный инструмент добавлен в корзину товаров

Шаги:

1. Пользователь нажимает иконку с корзиной
2. Сайт отображает ответ
   1. Если пользователь не авторизован, то сайт отобразит сообщение “Войдите в аккаунт”
   2. Если пользователь пытается добавить в корзину больше товаров, чем есть в магазинах выбранного города, то клиент получает ошибку “Попытка выбрать большее количество товаров, чем есть на складах”
   3. В остальных случаях счетчик количества товаров увеличивается на 1, в корзину пользователя добавляется товар

Сценарий №7: Оформление заказов

Цель: оформить заказ или заказы

Актор: авторизованный пользователь

Исходная страница: любая страница

Результат: заказы оформлены и переведены в состояние “в доставке”

Шаги:

1. Пользователь нажимает на иконку корзины в правом верхнем углу страницы
2. Сайт перенаправляет пользователя на страницу корзины товаров
3. При необходимости, пользователь корректирует заказы, например, убирает лишние товары или добавляет необходимые
4. Пользователь нажимает кнопку “Оформить”
5. Сайт передает данные пользователя на сервер
6. Сервер валидирует данные и дает ответ на сайт
7. Сайт отображает форму в зависимости от ответа сервера
   1. Если пользователь отправил некорректные данные, то сайт отобразит сообщение “Проверьте корректность заказа”
   2. Если пользователь заказал товаров больше, чем есть на складе в данный момент, то сайт отобразит сообщение “Вы выбрали больше товаров, чем есть на складе”
   3. В остальных случаях заказы оформляются, соответствующие записи сохраняются в базе данных

Сценарий №8: Просмотр истории заказов или заказов, находящихся “в доставке”

Цель: посмотреть историю заказов

Актор: авторизованный пользователь

Исходная страница: любая страница

Результат: пользователь переведен на страницу истории заказов

Шаги:

1. Пользователь нажимает на иконку - аватарку в правом верхнем углу
2. Сайт перенаправляет пользователя на страницу личного кабинета пользователя
3. Пользователь нажимает на кнопку “История заказов” или “Текущие заказы”
4. Сайт отправляет запрос на получение данных с сервера
5. Сервер отправляет ответ на сайт
6. Сайт отображает пользователю запрошенную информацию

# Структура проекта и взаимодействие компонентов

Проект состоит из трех основных частей: frontend (дальше сайт), backend (дальше сервер) и база данных. Пользователь взаимодействует с сайтом, сайт отправляет HTTP запросы на сервер, сервер получает HTTP запросы, обрабатывает полученные данные, при необходимости взаимодействует с базой данных, сервер отправляет HTTP запрос с ответом на сайт, сайт отображает полученную информацию пользователю.

# База данных

Основными сущностями, которые нас интересуют, являются клиент, товар и склад. Также существуют другие сущности, которые помогают описать клиента, товар или склад. Более подробно структуру базы данных описывают схемы, приведенные ниже. Но вначале несколько важных замечаний по СУБД SQLite:

1. INTEGER может занимать 0, 1, 2, 3, 4, 6 или 8 байт в зависимости от хранимых значений
2. REAL занимает 8 байт
3. TEXT по умолчанию кодируется с помощью UTF-8, в которой кириллица занимает по 2 байта на символ, а латиница и другие символы из ASCII по 1 байту
4. Номер телефона можно хранить как в текстовом, так и в целочисленном типе, но в первом случае тратится от 11 до 16 байт, а во втором 5, поэтому было решено хранить номер телефона в INTEGER.
5. В SQLite нет отдельного типа для хранения дат, поэтому для них используется TEXT в формате “YYYY-MM-DD HH:MM:SS”, REAL или INTEGER в секундах от 1970-01-01 00:00:00 по UTC. Самое удобное представление имеет текстовый формат. Стоит отметить, что в SQL реализованы функции для работы с датами.

## ER-диаграммы

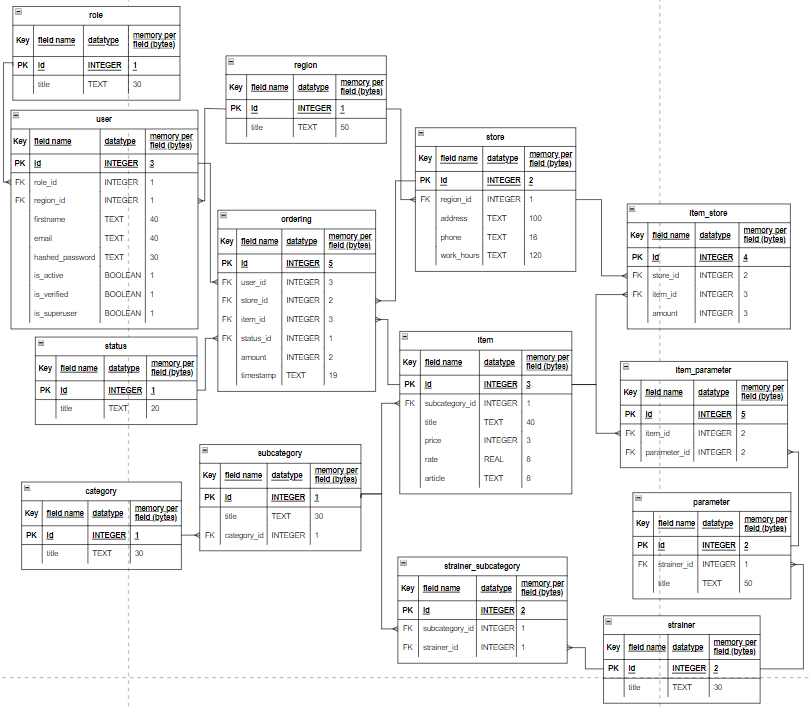
### Нотация Чена

### 

### Нотация “Воронья лапка”

### 

### Схема данных



Чтобы не перегружать последнюю схему, дополнительные параметры (такие как unique, not null, default value и другие), не были отображены на диаграмме, но они будут на скриншотах ниже.

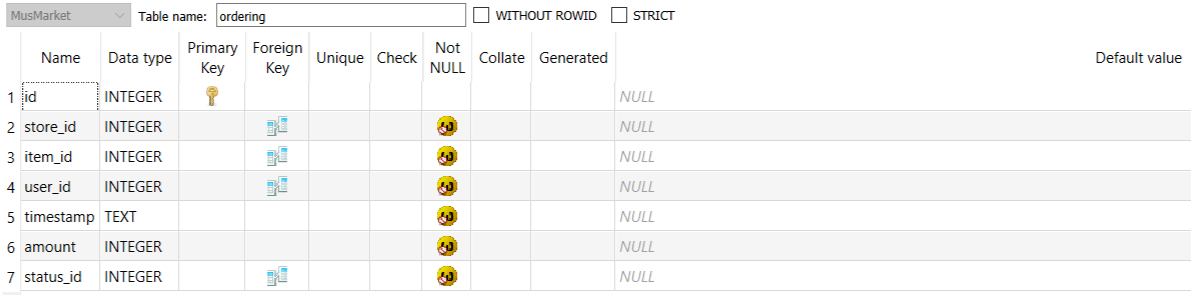
## Создание таблиц

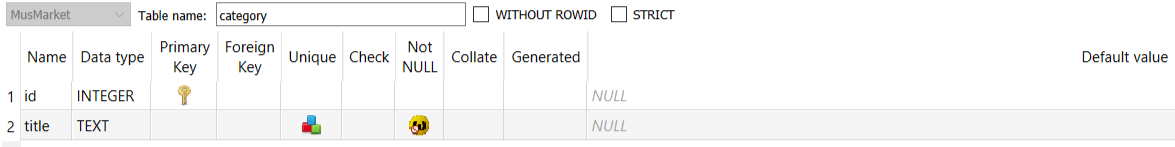
Основываясь на ER-диаграммах, с помощью SQL-запросов были созданы соответствующие таблицы и заданы соответствующие отношения между ними. В качестве примера рассмотрим запрос, который был использован для создания таблицы “ordering” (заказы)

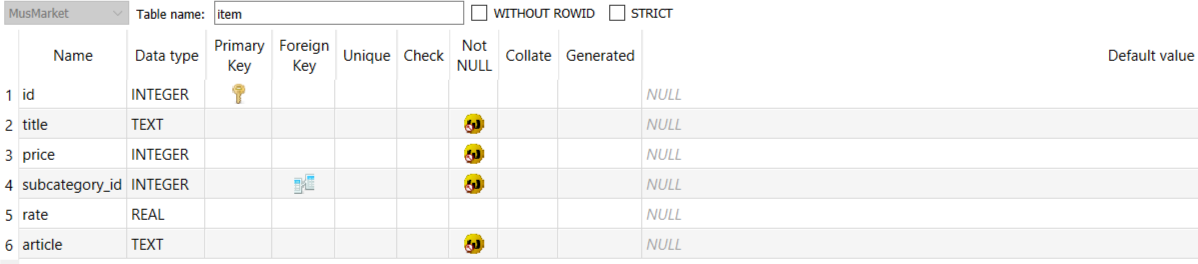
| CREATE TABLE IF NOT EXISTS ordering (  id INTEGER PRIMARY KEY AUTOINCREMENT,  store\_id INTEGER NOT NULL,  item\_id INTEGER NOT NULL,  user\_id INTEGER NOT NULL,  timestamp TEXT NOT NULL,  amount INTEGER NOT NULL,  status\_id INTEGER NOT NULL,  FOREIGN KEY (status\_id) REFERENCES status (id),  FOREIGN KEY (store\_id) REFERENCES store (id),  FOREIGN KEY (item\_id) REFERENCES item (id),  FOREIGN KEY (user\_id) REFERENCES user (id)  ); |
| --- |

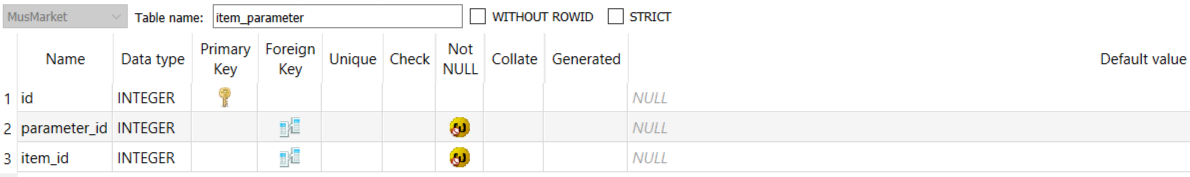
Остальные таблицы создавались с помощью аналогичных запросов.

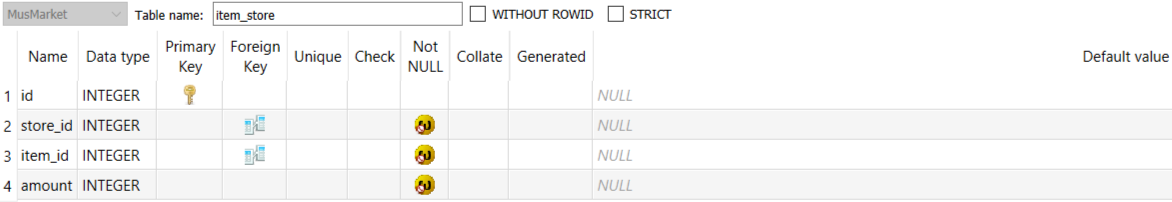
Результат:

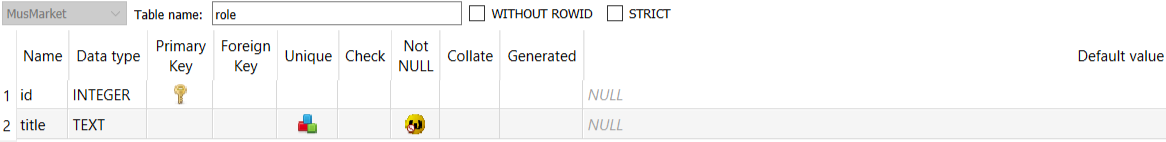


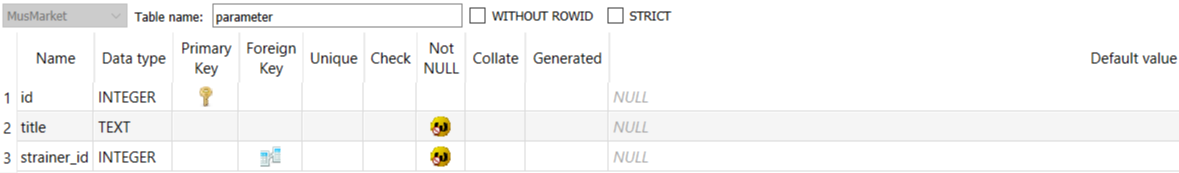


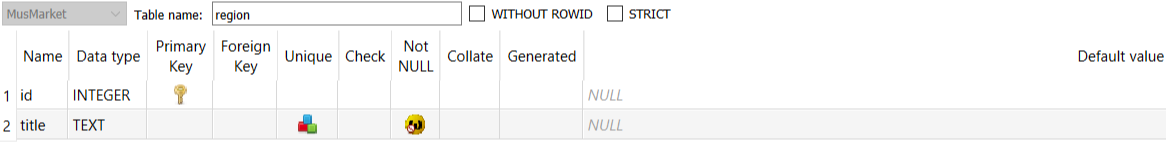


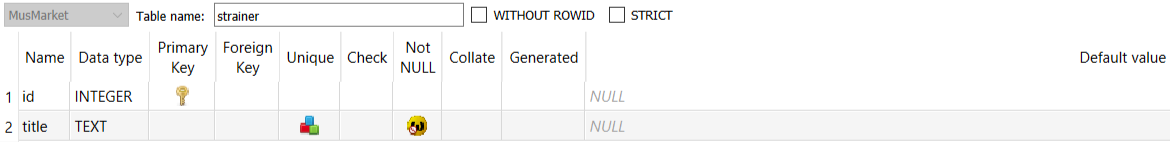


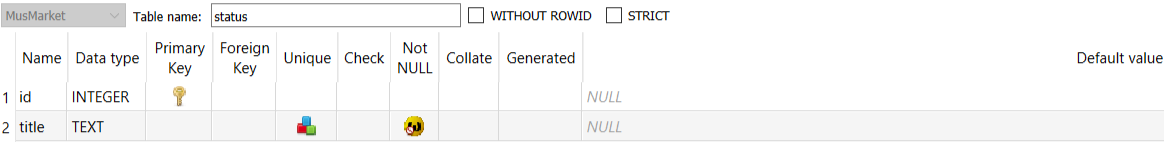


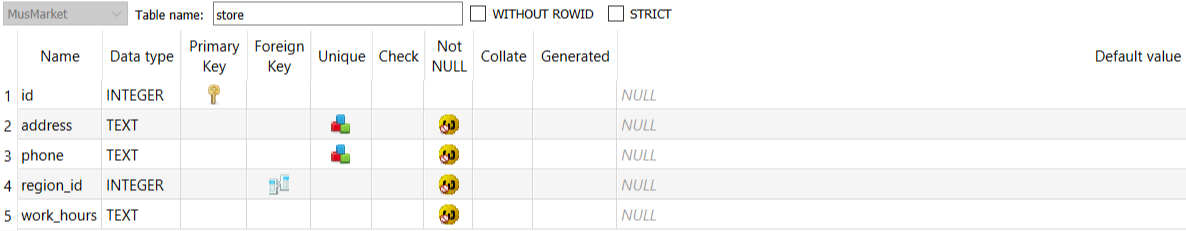


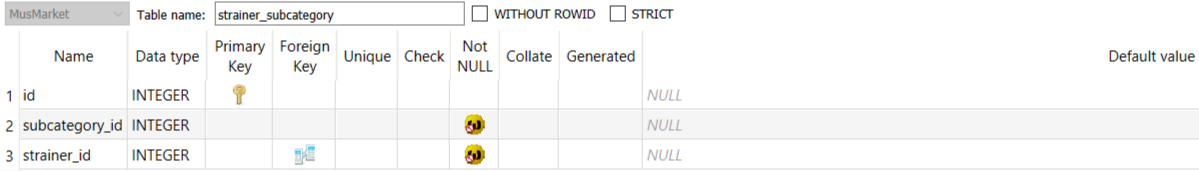


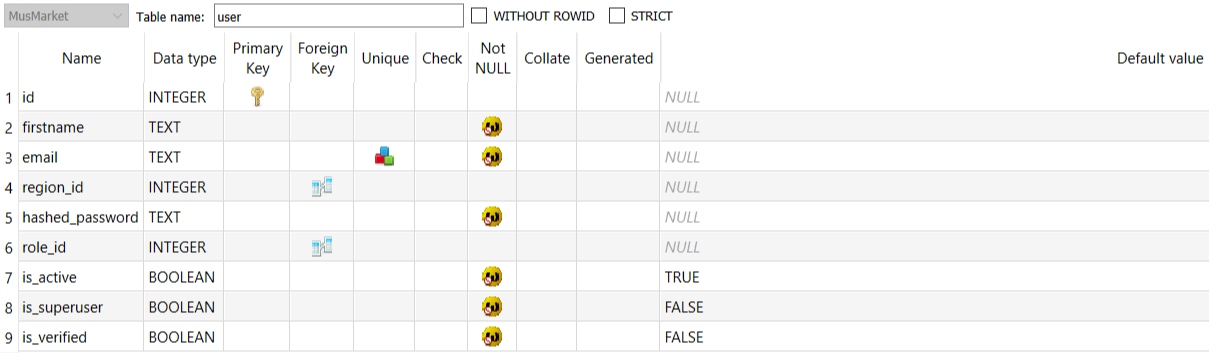


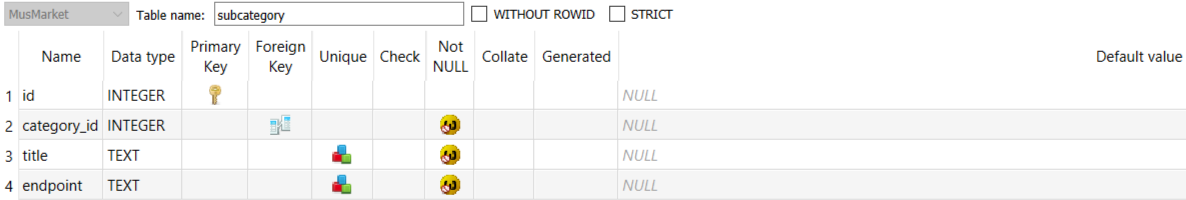










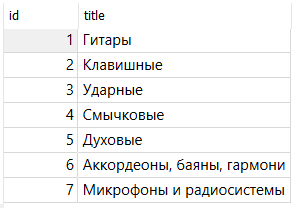


## Наполнение таблиц

Для обеспечения качества данных для тестового наполнения таблиц были использованы использованы 2 подхода: парсинг данных и написание генераторов на python. Так, например, с википедии были взяты регионы России (на октябрь 2023 года), с сайта “Музторга” основная информация о музыкальных инструментах, а сгенерированы, например, адреса магазинов, email’ы и пароли пользователей, заказы и другие таблицы и записи. Добавление записей в базу данных происходит также в python.

Результат наполнения таблиц:

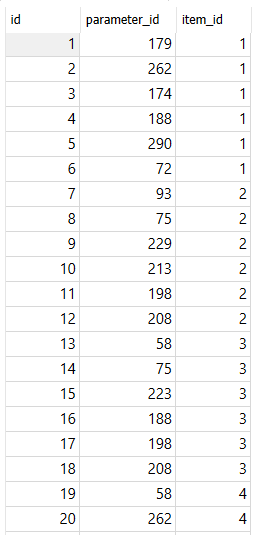
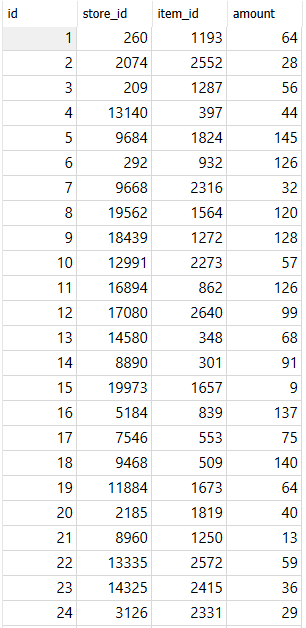
category



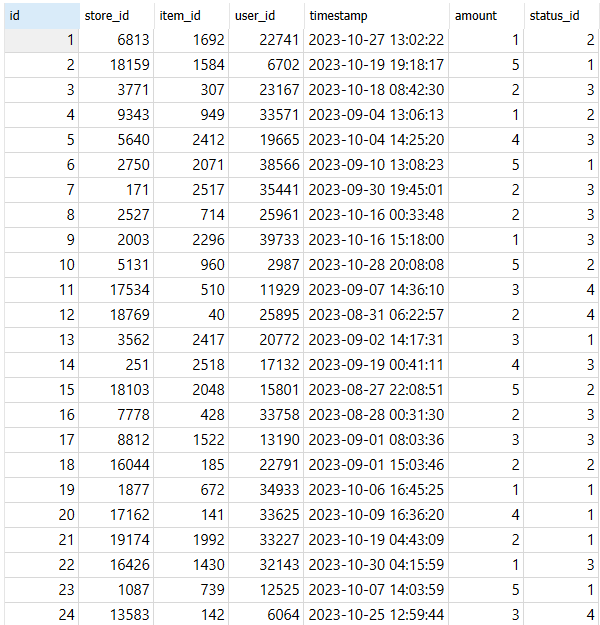
item



item\_parameter item\_store

ordering



Остальные 9 таблиц заполнены аналогично, поэтому их скриншоты прикреплять не имеет смысла.

# Frontend

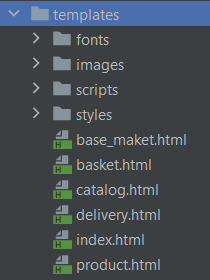
Веб-интерфейс разработан на HTML, CSS и JS.

На момент написания отчета, сайт включает в себя главную страницу, каталог товаров, страницу истории заказов, форму регистрации и аутентификации, корзину товаров.

Код размещенный непосредственно в этом документе, был бы абсолютно нечитаемым, поэтому он размещен на GitHub:

https://github.com/Krukrukruzhka/MuzMarket

Все файлы, связанные с интерфейсной частью, в проекте лежат в директории src/templates.



Файлы с расширениями .html являются шаблонами страниц, которые отправляются пользователю. Точная информация подставляется в них с помощью python и шаблонизатора Jinja2.

В директории styles лежат .css файлы, которые делают сайт красивым и понятным для пользователя.

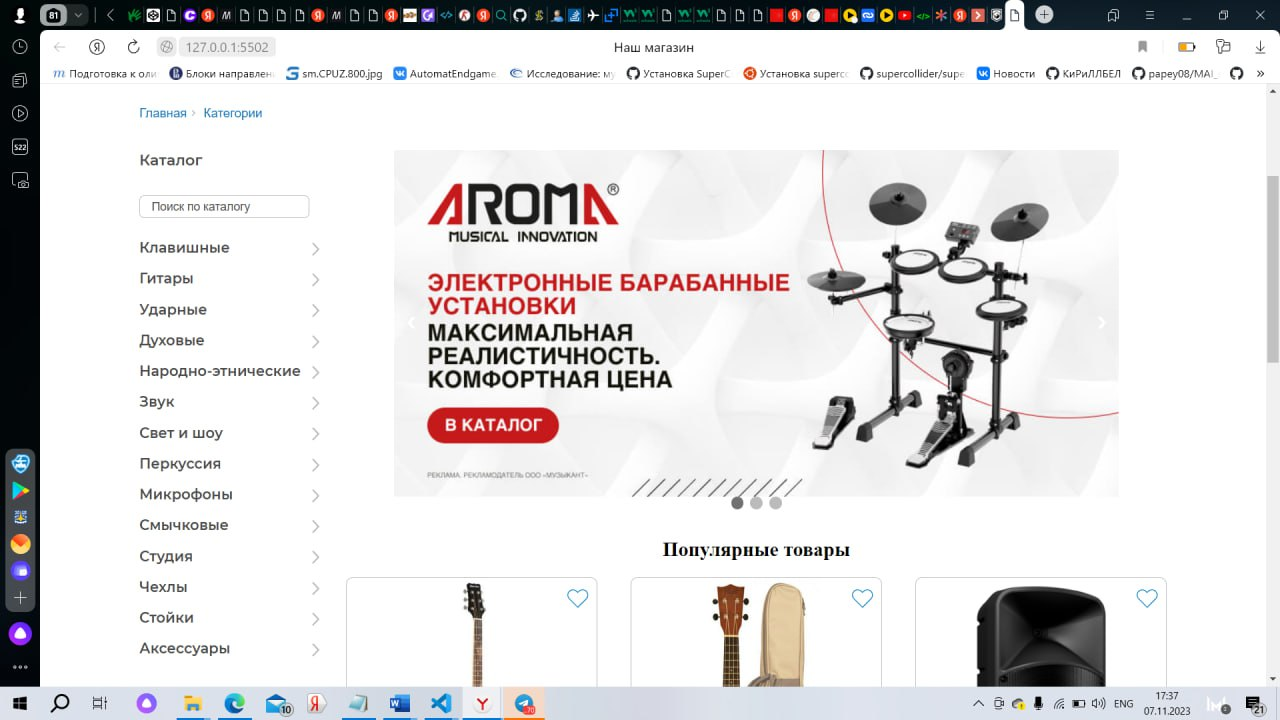
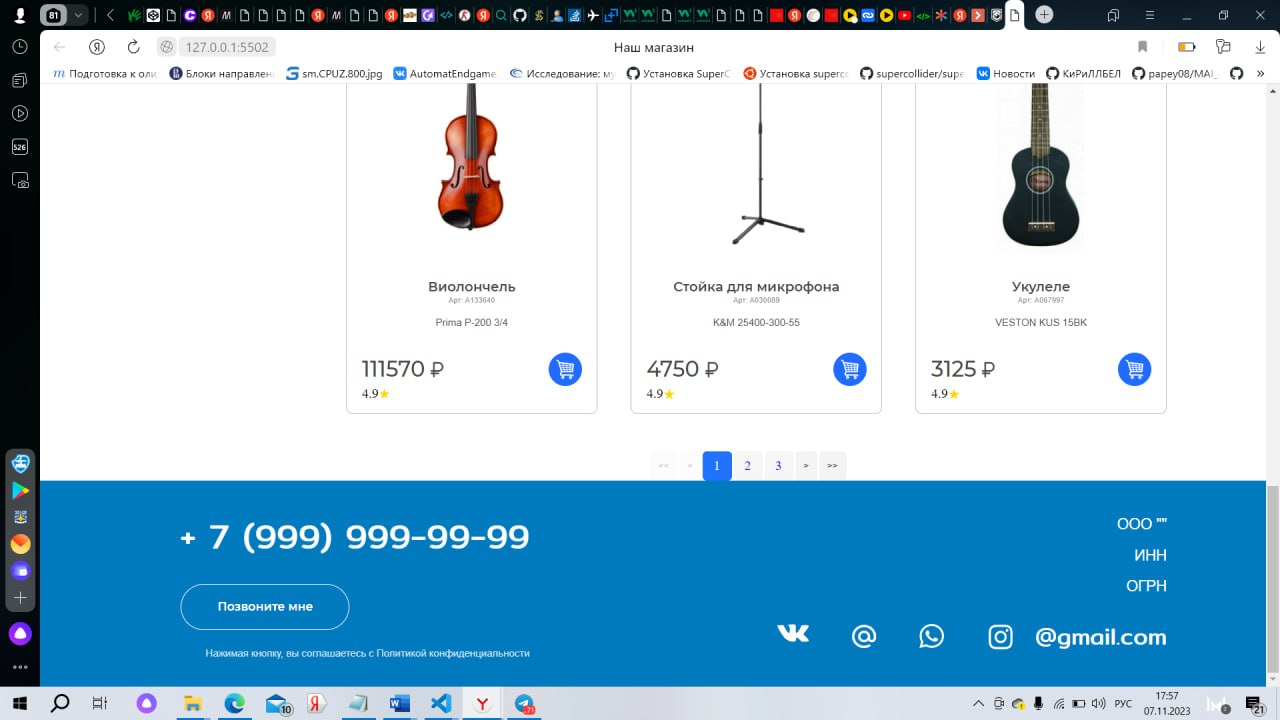
В папке scripts находятся .js файлы, которые делают сайт более интерактивным.

Директории fonts и images содержать шрифты и изображения соответственно.

**Главная страница**

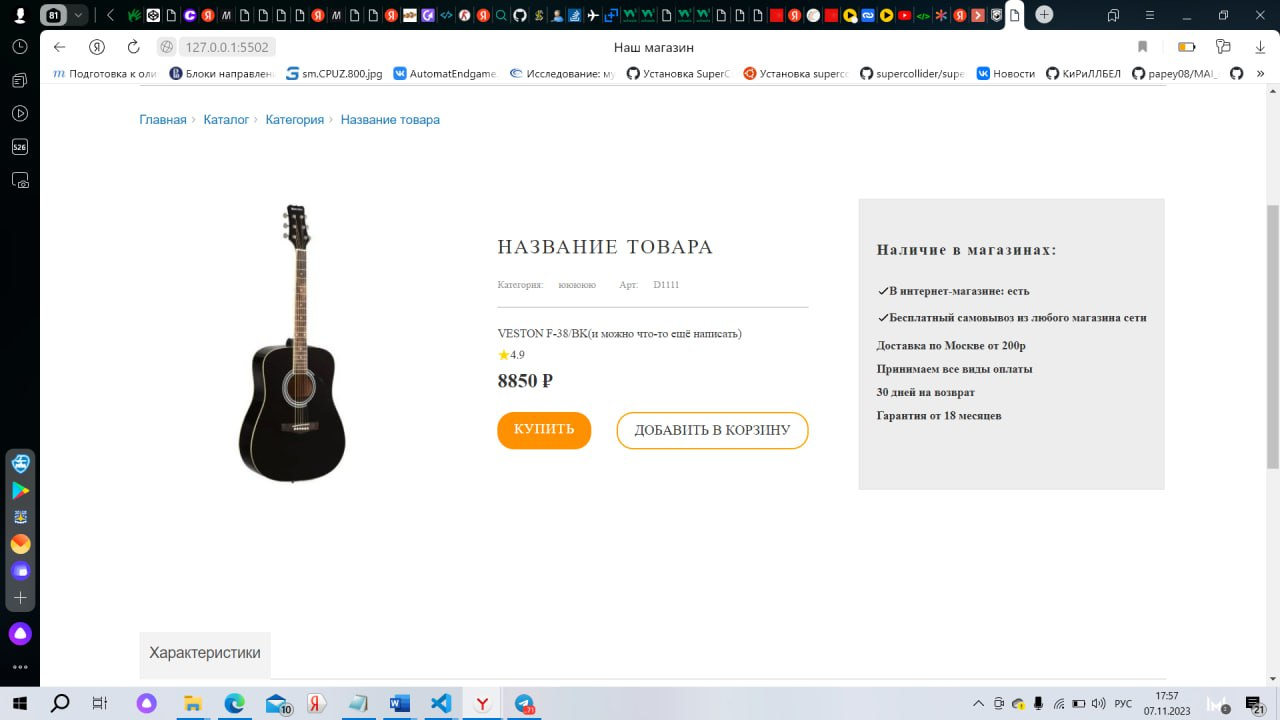
На главной странице представлена вся основная информация: адрес, телефон, вкладки с более подробной информацией, акции, обучающие курсы, наиболее популярные товары и прочее.

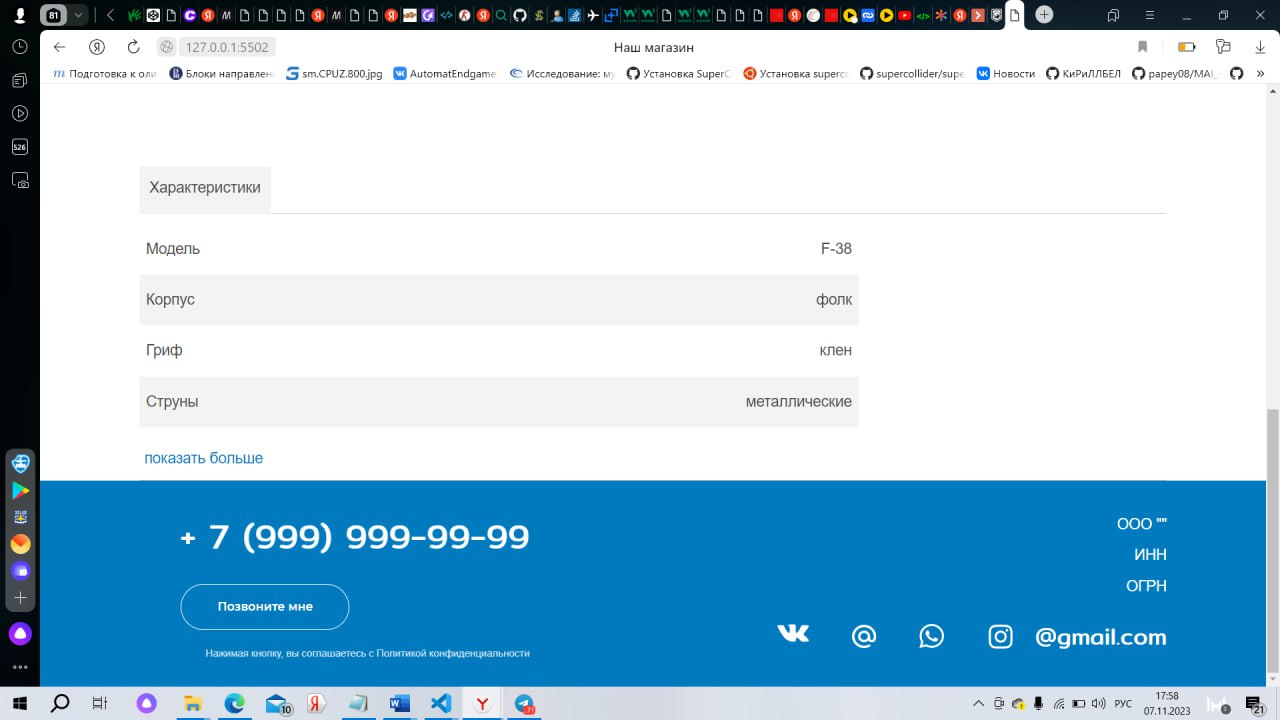
Также можно перейти в каталог товаров, нажав в левой части экрана на интересующую категорию музыкальных инструментов



**Страница товара**

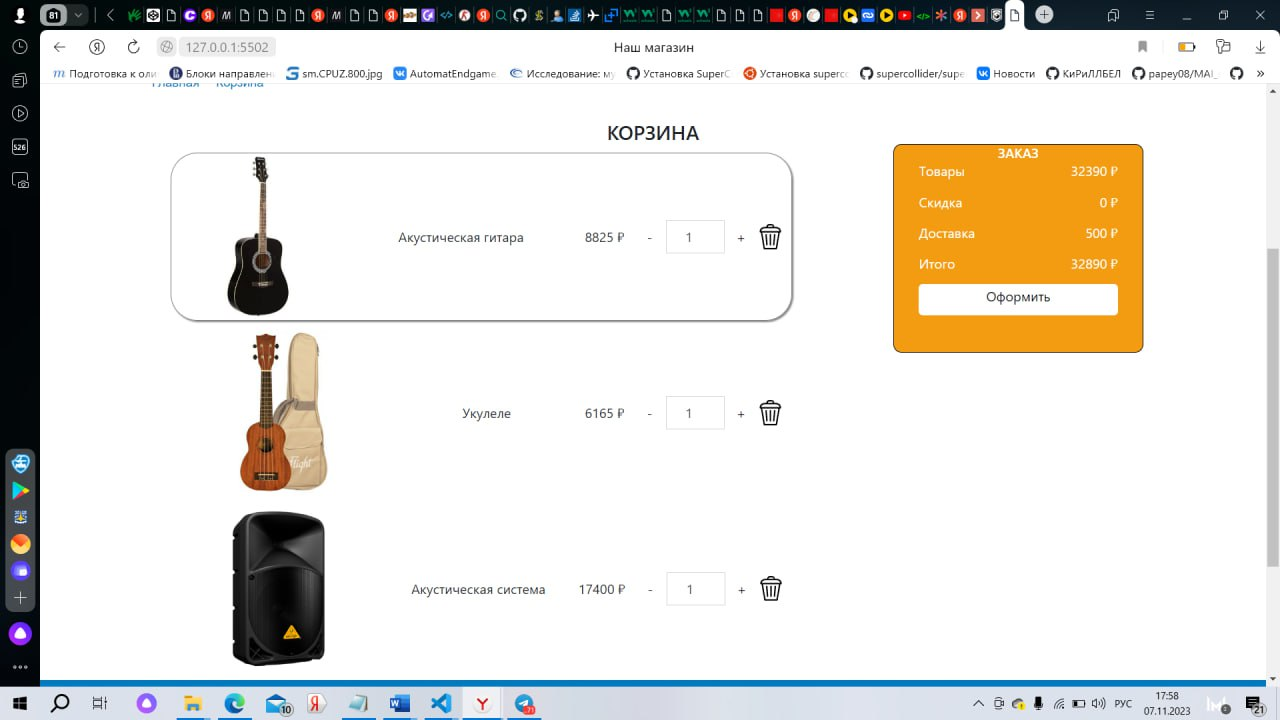
Страница продукта включает в себя описание конкретного товара и его характеристик, а также дополнительную информацию о доставке и наличии в магазинах





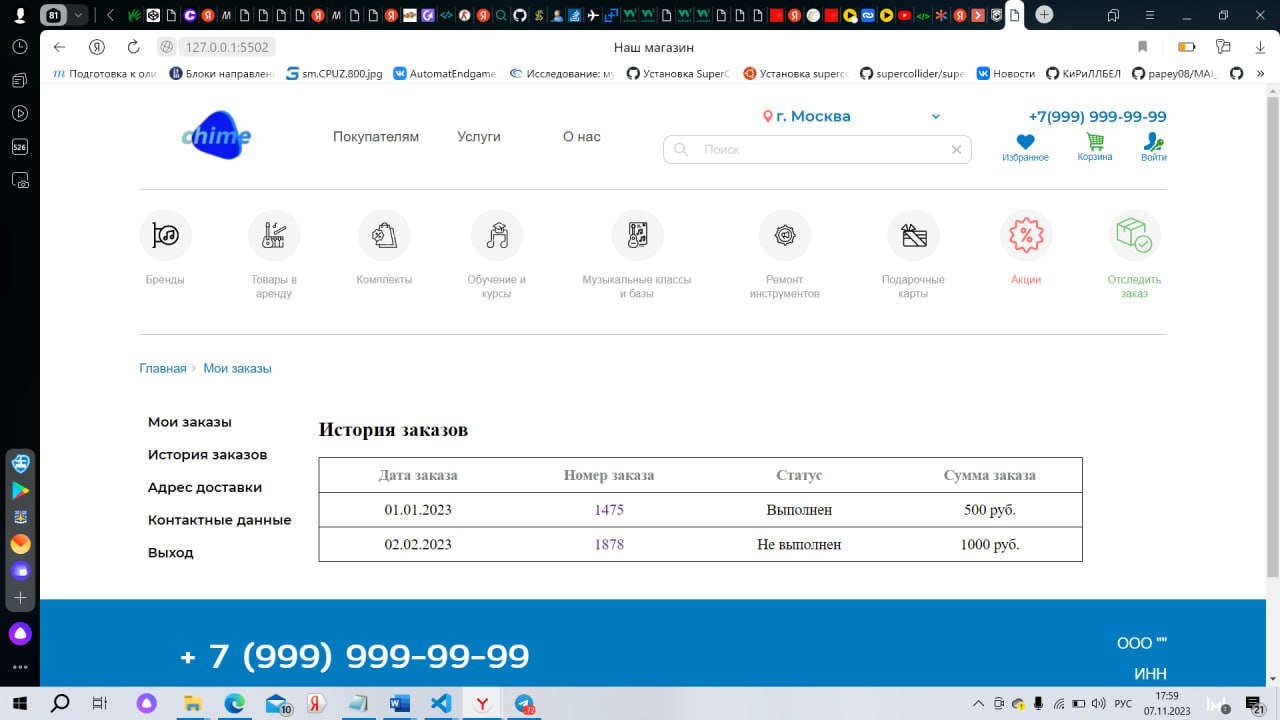
**Корзина**

Корзина содержит выбранные пользователем товары и блок оформления заказа с подсчетом итоговой суммы. На этой странице клиент может откорректировать заказ и оформить его.

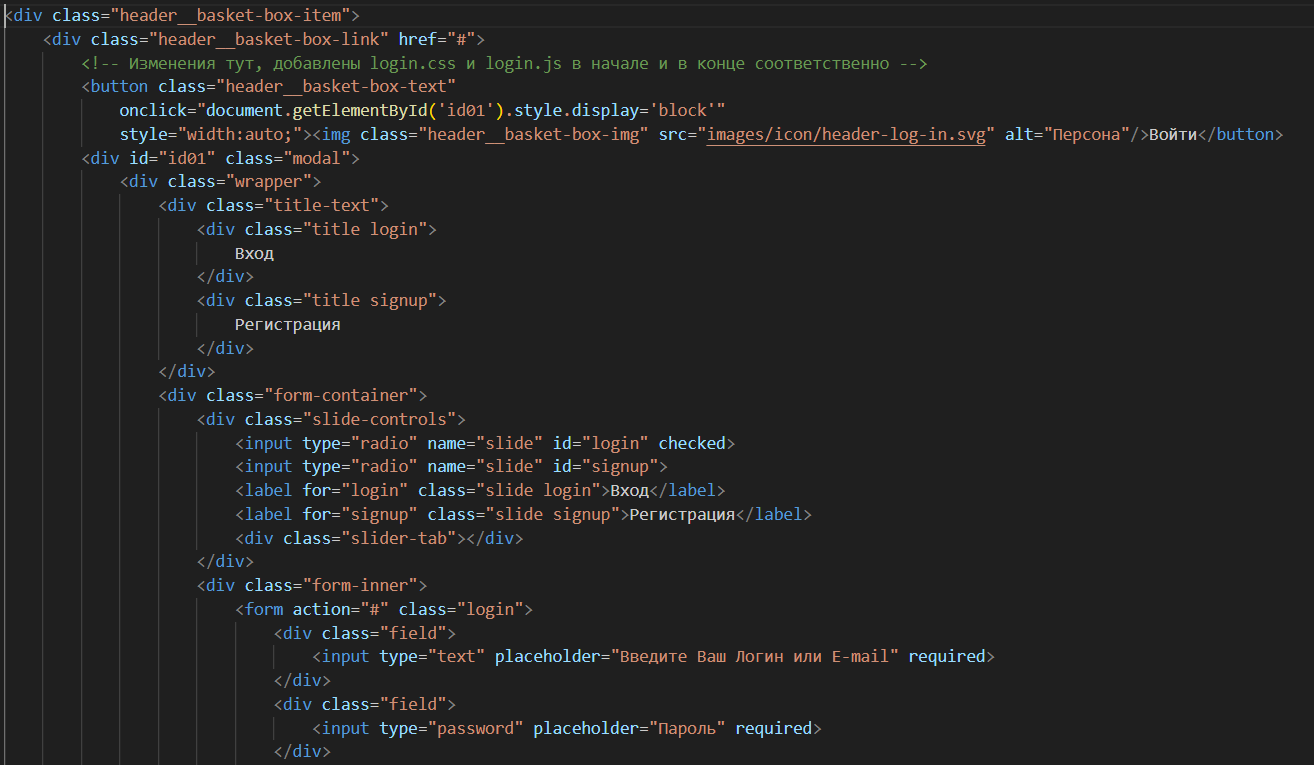


**История заказов**

Страница истории заказов включает в себя краткую информацию о заказе: дата, номер заказа, его статус и итоговую сумму.

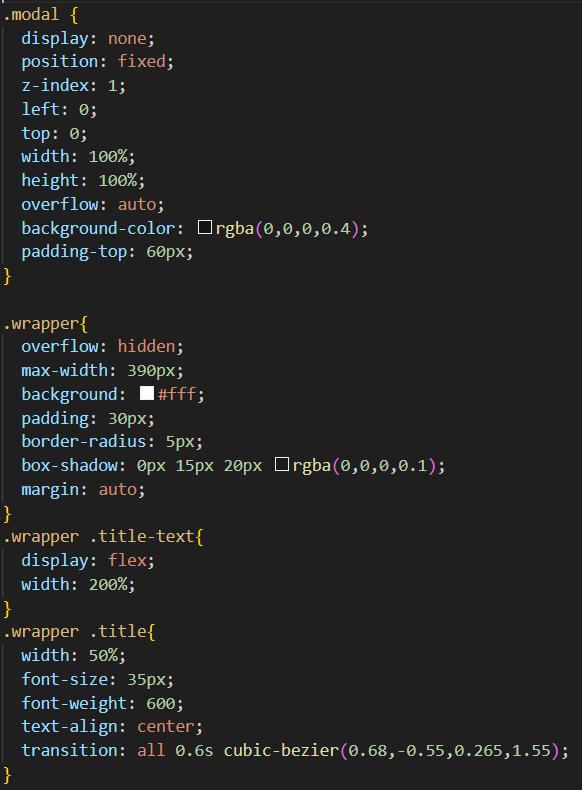


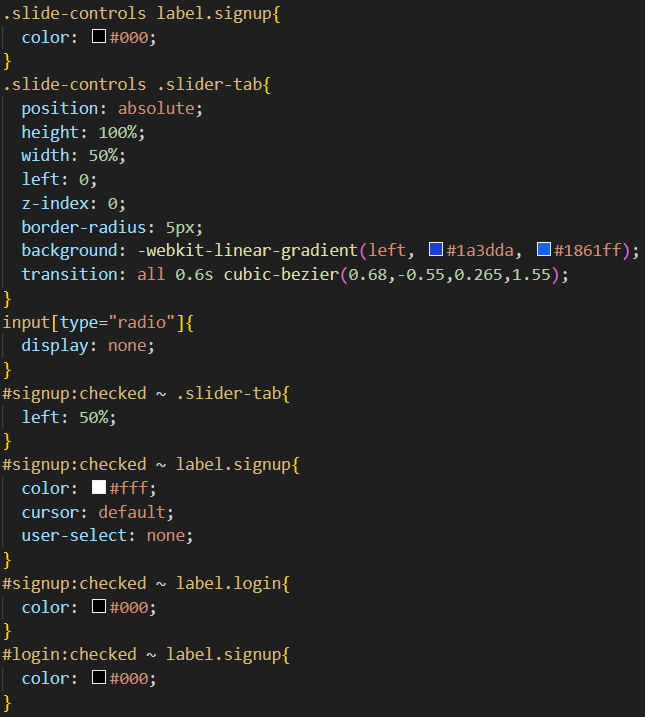
**Реализация формы аутентификации пользователя на HTML**



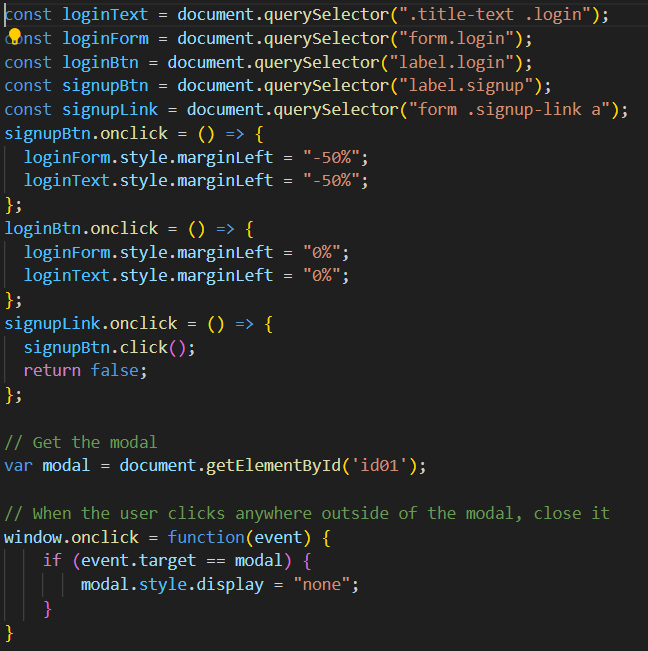


**Реализация дизайна формы аутентификации на CSS**



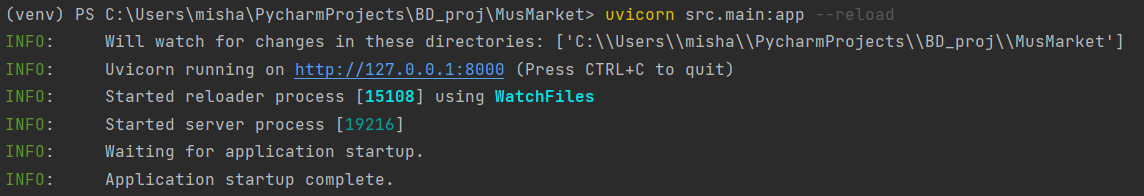


**Реализация функциональной части формы аутентификации на JS**

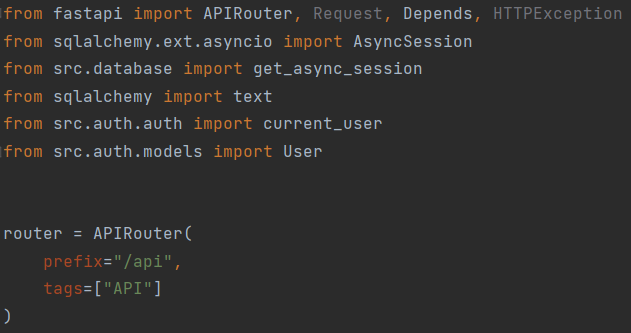


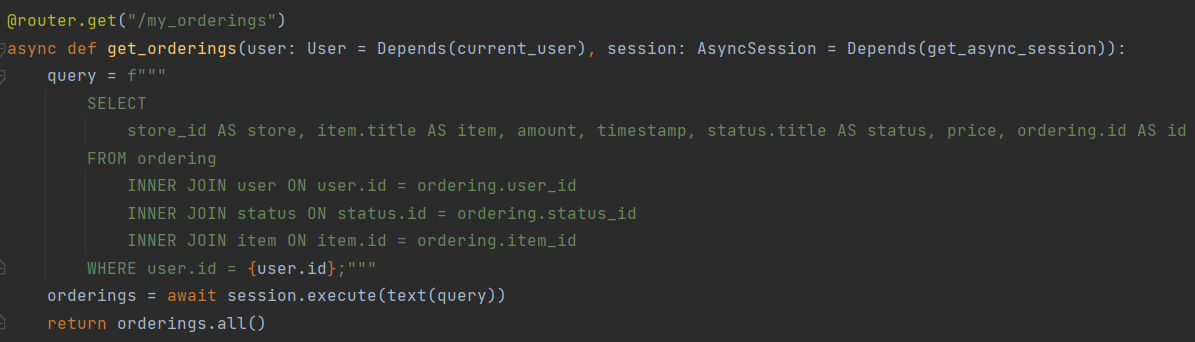
# Backend

Вся серверная часть разработана на языке python с применением фреймворка FastAPI. На момент написания отчета сервер работает и тестируется на localhost. Для запуска необходимо в консоли PyCharm прописать команду *uvicorn src.main:app –reload* после чего в браузере можно будет перейти на сайт по адресу 127.0.0.1:8000

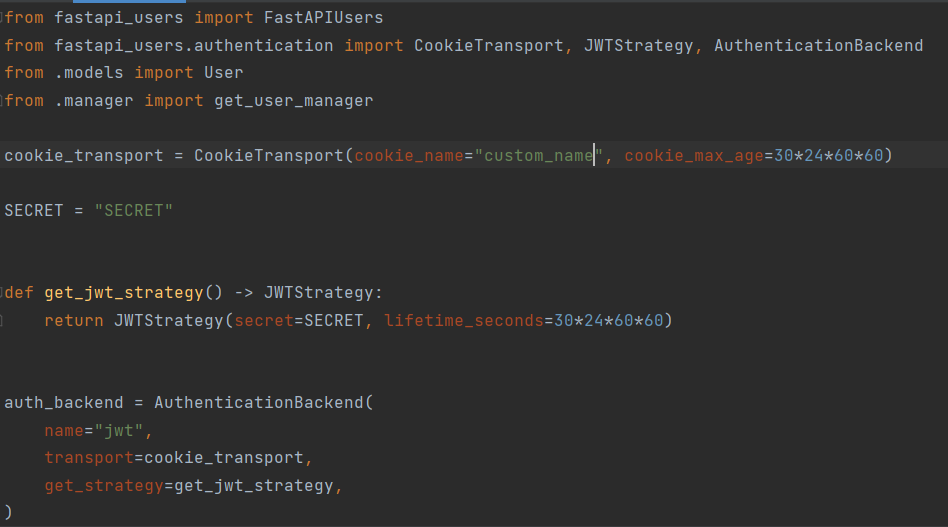


В директории src/api в файле router,py хранятся роуты, отвечающие за получение информации из базы данных и добавление данных в базу данных. Используя это API роутеры из остальных директориях формируют необходимые HTML страницы и HTTP запросы и отправляют их пользователю.

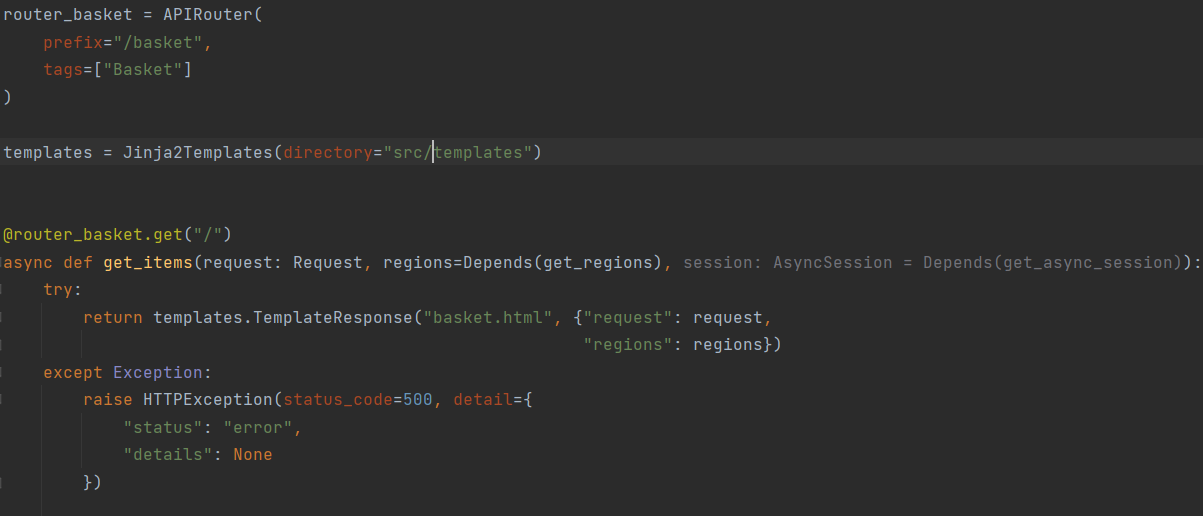




В директории src/auth хранятся файлы, отвечающие за регистрацию и аутентификацию пользователя.



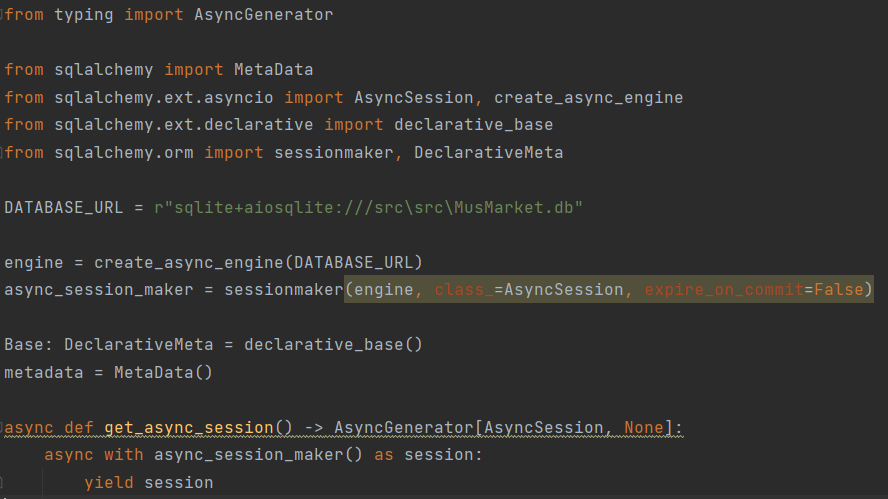
Директории catalog, basket, delivery хранят по одному файлу router.py, которые отвечают за формирование HTML страницы и ее отправку пользователю.



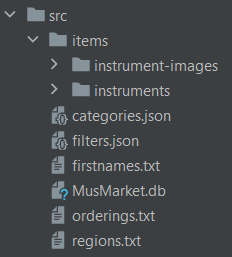
Файл main.py является точкой входа программы, он консолидирует в себе все роутеры.



Файл database.py отвечает за подключение базы данных и определяет метод создания сессий.



В директории src/src находятся такие неисполняемые файлы проекта, как, например, изображения товаров, база данных, вспомогательные файлы для создания и наполнения таблиц базы данных.



В рамках бэкенда было реализовано API, для корректного взаимодействия серверной части (с базой данных) и интерфейсной. Оно представляет из себя отдельные функции в python, которые позволяют получить данные из базы данных с помощью SQL-запросов.

* Запрос на получение списка регионов

| SELECT title FROM region |
| --- |
|  |

* Запрос на получение заказов текущего пользователя

| SELECT  store\_id AS store, item.title AS item, amount, timestamp, status.title AS status, price, ordering.id AS id FROM ordering  INNER JOIN user ON user.id = ordering.user\_id  INNER JOIN status ON status.id = ordering.status\_id  INNER JOIN item ON item.id = ordering.item\_id WHERE user.id = {user.id} |
| --- |

* Запрос на получении информации о конкретном товаре (по артикулу)

| SELECT  subcategory.title AS subcategory, endpoint,  item.title AS title, price, rate, article,  parameter.title AS parameter,  strainer.title AS strainer FROM item  INNER JOIN subcategory ON subcategory.id = item.subcategory\_id  INNER JOIN item\_parameter ON item\_parameter.item\_id = item.id  LEFT JOIN parameter ON parameter.id = item\_parameter.parameter\_id  INNER JOIN strainer ON strainer.id = parameter.strainer\_id WHERE article = "{article} |
| --- |

* Запросы на получение товаров, категорий товаров и параметров конкретной подкатегории товаров (в виде функции python)

| @router\_catalog.get("/{instrument\_subcategory}") async def get\_items(request: Request, instrument\_subcategory: str, page: int = 1, regions=Depends(get\_regions),  session: AsyncSession = Depends(get\_async\_session)):  try:  ELEMENTS\_COUNT = 6  query = f"""  SELECT item.title AS title, price, rate, endpoint, article, subcategory.title AS subcategory FROM item  INNER JOIN subcategory ON item.subcategory\_id = subcategory.id  WHERE subcategory.endpoint = "{instrument\_subcategory}"  ORDER BY rate DESC, price ASC  LIMIT {ELEMENTS\_COUNT} OFFSET {(page - 1) \* ELEMENTS\_COUNT};  """  items = await session.execute(text(query))   query = f"""  SELECT subcategory.title AS subcategory, category.title AS category, subcategory.endpoint AS endpoint FROM subcategory  INNER JOIN category ON subcategory.category\_id = category.id  WHERE subcategory.endpoint = "{instrument\_subcategory}";  """  subcategory\_info = await session.execute(text(query))   query = f"""  SELECT DISTINCT  strainer.title AS strainer\_title,  parameter.title AS parameter\_title  FROM item  INNER JOIN subcategory ON item.subcategory\_id = subcategory.id  INNER JOIN item\_parameter ON item\_parameter.item\_id = item.id  INNER JOIN parameter ON item\_parameter.parameter\_id = parameter.id  INNER JOIN strainer ON strainer.id = parameter.strainer\_id  WHERE subcategory.endpoint = "{instrument\_subcategory}";  """  parameters\_info = await session.execute(text(query))  parameters\_info = parameters\_info.all()  strainers = set(map(lambda x: x[0], parameters\_info))  strainers = {i: [] for i in sorted(strainers)}  for i in parameters\_info:  strainers[i[0]].append(i[1])   return templates.TemplateResponse("catalog.html", {"request": request,  "items": items.all(),  "strainers": strainers,  "subcategory\_info": subcategory\_info.fetchone(),  "regions": regions})  except Exception:  raise HTTPException(status\_code=500, detail={  "status": "error",  "details": None  }) |
| --- |

* Запрос на добавление пользователя в базу данных

| INSERT INTO user (firstname, email, hashed\_password) VALUES  ('{firstname}', '{email}', '{hashed\_password}') |
| --- |

# 

# Вывод

Работая над курсовым проектом, каждый участник нашей команды обрел навыки работы с реляционными базами данных: концептуальное и логическое проектирование, написание SQL-запросов, использование базы данных в программе, написанной на высокоуровневом языке программирования (в нашем случае python версии 3.10), ознакомились с разными СУБД и использовали одну из них в проекте. Также мы использовали различные языки программирования и средства разработки в зависимости от задачи. Например, для верстки страниц сайта потребовались HTML, CSS и JavaScript; Jinja2Templates позволил шаблонизировать эти странички; с помощью языка python и фреймворка FastAPI были настроены сервер, роуты и эндпоинты.

Кроме того, каждый смог углубиться в интересующей его сфере.

Мы не только развили навык командной разработки и добавили проект в свои репозитории, но и приобрели навыки и теоретические знания, которые безусловно понадобятся нам при работе над более сложными проектами.