

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. СВЯЗАННЫЕ ОПРЕДЕЛЕНИЯ.....	5
1.1 Базы данных .....	5
1.2 Модель «Сущность-Связь» .....	6
1.3 Информация о Flask .....	7
1.4. Установка Flask.....	8
1.5. Минимальное приложение Flask .....	9
1.6. Информация об SQLite3 .....	9
2. РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ .....	11
3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ .....	13
3.1. Выбор средств реализации.....	13
3.2. Структура веб-приложения .....	13
3.3. Описание функционала приложения .....	14
3.4. Работа с реализованным приложением.....	18
4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ .....	20
ЗАКЛЮЧЕНИЕ .....	26
СПИСОК ЛИТЕРАТУРЫ.....	27

## **ВВЕДЕНИЕ**

В современном мире важнейшим компонентом любой информационной системы является база данных. Умение работать с реляционными базами данных является обязательным для любого прикладного разработчика. Несмотря на то, что в данный момент существует множество различных СУБД, всех их объединяет множество вещей.

В курсовой работе была поставлена цель создать базу данных «Интернет-магазин спортивных товаров», используя систему управления базами данных SQLite, и для ее визуализации написать веб-приложение на языке Python 3 с использованием микрофреймворка Flask. Приложение создавалось с помощью таких технологий как HTML, CSS, PYTHON, FLASK, SQL, SQLite.

Для выполнения данной работы было необходимо:

- Изучить микрофреймворк Flask.
- Повторить работу с программными средствами описания внешнего вида документа, такими как html и css.
- Изучить работу с СУБД SQLite3.

# 1. СВЯЗАННЫЕ ОПРЕДЕЛЕНИЯ

## 1.1 Базы данных

База данных — это организованная структура, предназначенная для хранения информации[1]. Другими словами, база данных является структурированным набором постоянно хранимых данных. Изначально предполагалось, что в базах данных будет храниться только информация. Однако в настоящее время большинство систем управления базами данных позволяют размещать в своих структурах не только данные, но и методы, с помощью которых происходит взаимодействие с пользователем.

Существуют реляционные и нереляционные базы данных. В реляционных базах данных информация хранится в виде двумерных таблиц.

Пример реляционной базы данных можно увидеть на рис.1:

The diagram illustrates a relational database structure with three tables: **products**, **orders**, and **customers**. The **products** table contains information about various fruits. The **orders** table tracks purchases, linking customers to specific products. The **customers** table lists individual clients with their contact details.

prod_id	prod_name	prod_price
1000pr	apples	1.23
1001pr	oranges	2.34
1002pr	bananas	3.45
1003pr	pears	4.45

**products**

order_id	cust_id	prod_id	price
100001	2001cu	1002pr	3
100002	2003cu	1001pr	5
100003	2000cu	1000pr	12
100004	2002cu	1003pr	9

**orders**

cust_id	cust_name	cust_email
2000cu	Milano	felix@milano.com
2001cu	Apress	info@apress.com
2002cu	ABC, Inc.	jeff@abc.com
2003cu	XYZ, Inc.	web@xyz.com

**customers**

*Рис.1 Пример реляционной базы данных*

Как можно понять из рисунка 1, в реляционной базе данных информация

хранится в виде двумерных таблиц. Таблица в реляционной БД - это структура, состоящая из строк и столбцов, заполненная данными. Строки в таблицах часто называют записями, столбцы - полями.

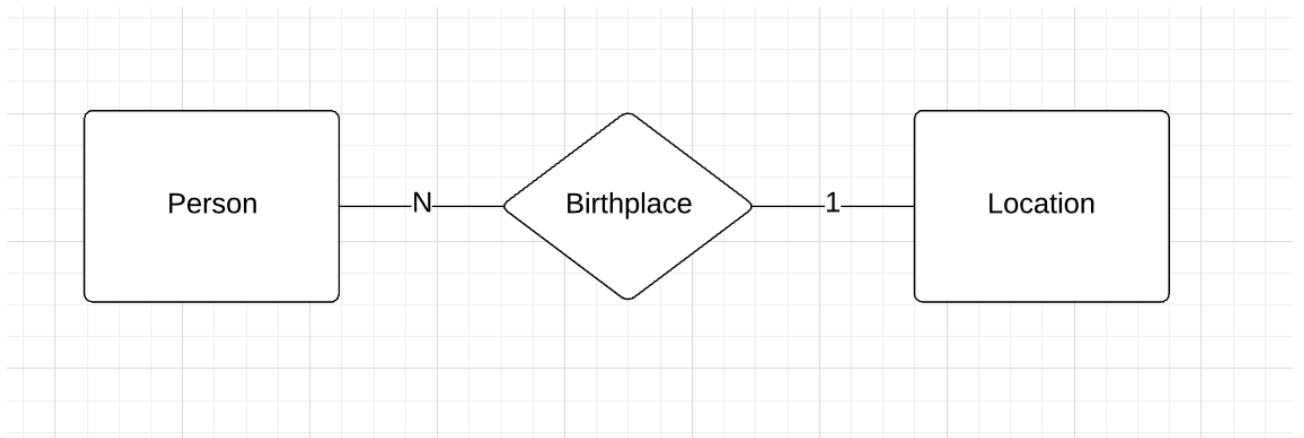
## 1.2 Модель «Сущность-Связь»

Модель «сущность – связь» (ER-модель) представляет собой набор концепций, используемых для описания логической структуры базы данных[2]. Связь - это отношения между моделями (можно считать связью упорядоченный набор сущностей). У связи есть несколько параметров:

- Размерность - определяет количество видов участвующих в связи объектов. Наиболее часто используемая связь - между двумя видами объектов (бинарная связь).
- Мощность - значение максимального количества конкретных экземпляров сущностей, которые могут использоваться для данной связи.
- Тип :
  - один-к-одному: один экземпляр первой сущности связан только с одним экземпляром второй сущности.
  - один-ко-многим: один экземпляр первой сущности связан с несколькими экземплярами второй сущности.
  - много-ко-многим: каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.
- Модальность:
  - «может»: экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может и не быть связан ни с одним экземпляром.
  - «должен»: экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

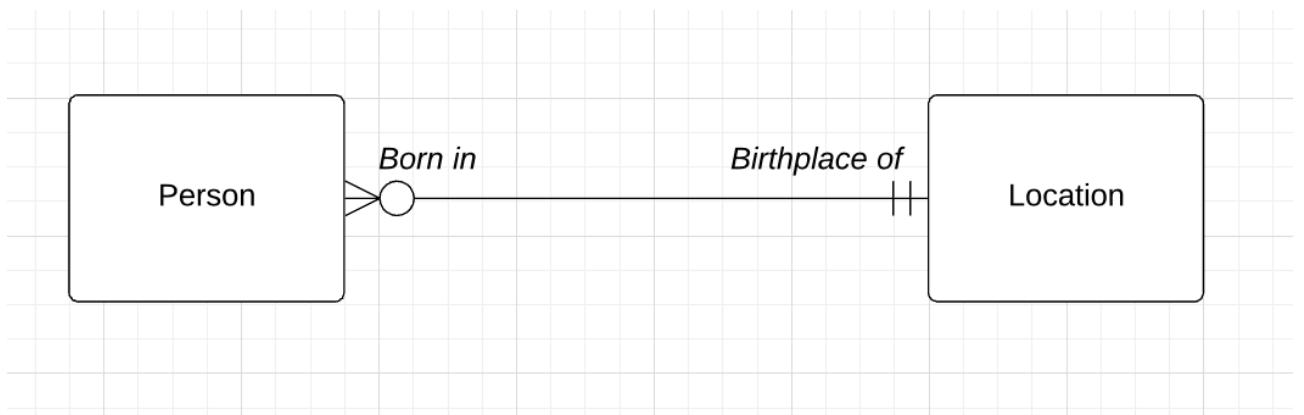
Существует несколько нотаций графического представления ER-моделей:

- Классическая нотация П. Чена (рис.2)



*Рис.2 Пример ER-модели в нотации П. Чена*

- Нотация Дж. Мартина (рис.3)



*Рис.3 Пример ER-модели в нотации Дж. Мартина*

### 1.3 Информация о Flask

Flask - это микрофреймворк для написания веб-приложений на языке Python[3]. «Микро» в слове «микрофреймворк» означает, что Flask стремится придерживаться простого, но расширяемого ядра. По умолчанию, Flask не включает валидацию форм или каких-то иных, для чего уже существуют различные занимающиеся этим библиотеки. Вместо этого, Flask поддерживает расширения для добавления подобной функциональности в приложение. Во

Flask многие вещи предварительно сконфигурированы, на основе общей базовой конфигурации. Например, шаблоны и статические файлы сохранены в подкаталогах в пределах исходного дерева.

#### 1.4. Установка Flask

Flask зависит от некоторых внешних библиотек - таких, как Werkzeug и Jinja2. Werkzeug - это инструментарий для WSGI - стандартного интерфейса Python между веб-приложениями и различными серверами, предназначен как для разработки, так и развёртывания. Jinja2 занимается отображением шаблонов.

Для разработки приложения с использованием Flask необходимо было использовать виртуальную среду `virtualenv`[4]. Виртуальная среда — это полная копия интерпретатора Python. При установке пакетов в виртуальной среде затрагивается не общесистемный интерпретатор Python, а только копия. Таким образом, лучшим решением является использование новой виртуальной среды для каждого приложения. Это решает следующую проблему: довольно часто библиотеки нарушают обратную совместимость, и несколько проектов имеют конфликтующие зависимости.

Для работы с микрофреймворком необходимо было прописать в терминале следующие команды (для операционной системы MacOS):

```
pip install virtualenv - установка виртуального окружения  
mkdir flask_project  
cd ./flask_project  
virtualenv venv - создание виртуального окружения с названием «venv»  
. venv/bin/activate - активация виртуального окружения  
pip install Flask - установка Flask в виртуальное окружение
```

Для установки недостающих пакетов была использована команда `pip install package_name`.

## 1.5. Минимальное приложение Flask

Структура веб-приложения Flask выглядит следующим образом: (рис.4)

```
from flask import Flask
app = Flask(__name__) #экземпляр класса Flask

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run() #запуск сервера
```

Рис.4: Структура простейшего Flask-приложения

Если в `app.run()` не указывать номер порта, то сервер запустится на <http://127.0.0.1:5000/>

## 1.6. Информация об SQLite3

SQLite - это библиотека, предназначенная для работы с SQL, которая написана на языке C [5]. При работе с SQLite данные будут брать непосредственно с диска, то есть нет необходимости обращаться к серверу, в отличие, например, от MySQL.

Один из возможных способов установки sqlite3 в систему MacOS - прописать в терминале команду, использующая утилиту Homebrew:

```
brew install sqlite3
```

Для работы с SQLite3 в языке Python достаточно импортировать модуль sqlite3 и создать связь с базой данных. На рис.5 показан пример взаимодействия с базой данных и внесения в нее изменений с помощью библиотеки sqlite3:

```
import sqlite3

conn = sqlite3.connect("database.db")
cursor = conn.cursor()

# Создание таблицы
cursor.execute("CREATE TABLE albums (title text, artist text, publisher text)")
# Вставка данных в таблицу
cursor.execute("INSERT INTO albums VALUES ('Glow', 'Andy Hunter', 'Xplore Records')")
# Сохранение изменений
conn.commit()
```

*Рис.5 Пример работы с SQLite3 в Python*

## 2. РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ

Темой базы данных было решено взять онлайн-магазин спортивных товаров. В базе данных реализованного приложения хранится информация об администраторах магазина (admin), товарах (product), пользователях (user) и их корзинах с товарами (cart). ER-модель базы данных веб-приложения в нотации Дж. Мартина представлена на рис.5:

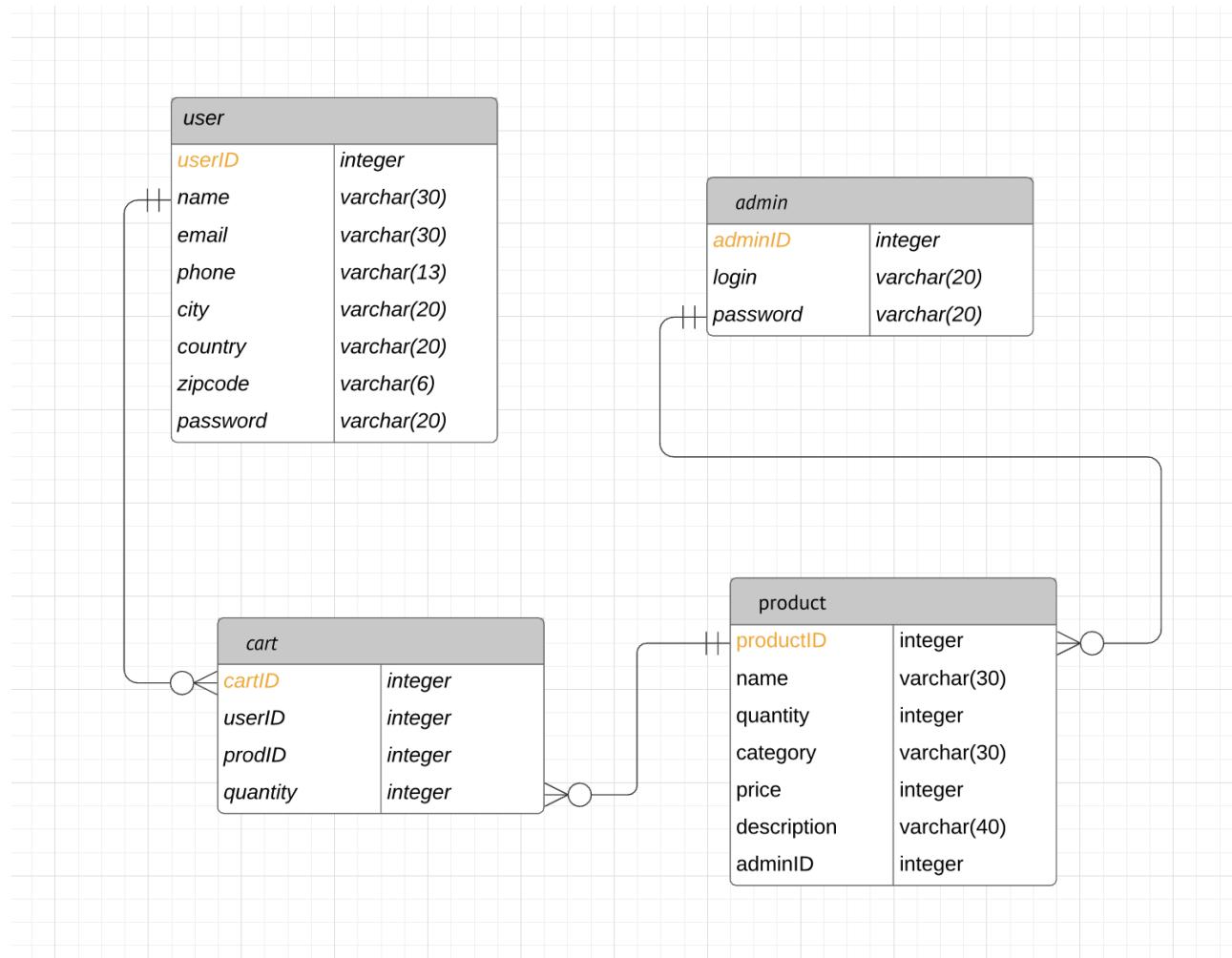


Рис.5 ER-модель базы данных веб-приложения в нотации Дж. Мартина

Структуры таблиц user и product изображены на рис. 6, 7 соответственно.

```
[CREATE TABLE "user" (
    "userID" INTEGER PRIMARY KEY AUTOINCREMENT,
    "name" varchar(30) NOT NULL,
    "email" varchar(30) UNIQUE,
    "phone" varchar(13) NOT NULL,
    "city" varchar(20) NOT NULL,
    "country" varchar(20) NOT NULL,
    "zipcode" varchar(6) NOT NULL,
    "password" varchar(20) NOT NULL,
    "public_access" TEXT
);
```

Рис.6 Структура таблицы user в базе данных shop.db

```
]CREATE TABLE "product" (
    "prodID" INTEGER PRIMARY KEY AUTOINCREMENT,
    "name" varchar(30) NOT NULL,
    "quantity" INTEGER NOT NULL,
    "category" varchar(30) NOT NULL,
    "price" INTEGER NOT NULL,
    "description" varchar(40),
    "adminID" INTEGER NOT NULL,
    FOREIGN KEY("adminID") REFERENCES "admin"("adminID")
);
```

Рис.7 Структура таблицы product в базе данных shop.db

## 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

### 3.1. Выбор средств реализации

Приложение было написано на языке Python 3 с использованием микрофреймворка Flask. Также использовались такие технологии как HTML, CSS, SQL, СУБД SQLite.

### 3.2. Структура веб-приложения

На рис.8 можно увидеть скелет реализованного приложения (выполнено с помощью команды tree).

```
ekaterinagimranova — bash — 93x31
/Users/ekaterinagimranova/PycharmProjects/flask-shop/
├── __pycache__
│   ├── app.cpython-37.pyc
│   └── database.cpython-37.pyc
├── app.py
├── database.py
├── shop.db
└── shop.db.sqbpro
├── static
│   ├── 1.jpg
│   ├── back.png
│   ├── main.js
│   └── styles.css
└── templates
    ├── addproduct.html
    ├── cart.html
    ├── change_password.html
    ├── edit_profile.html
    ├── home.html
    ├── home_admin.html
    ├── login.html
    ├── loginadmin.html
    ├── product_info.html
    ├── register.html
    ├── search_prod.html
    ├── searchprod_admin.html
    ├── success_register.html
    └── view_profile.html

3 directories, 24 files
MacBook-Pro-Ekaterina:~ ekaterinagimranova$
```

Рис.8 Структура реализованного веб-приложения

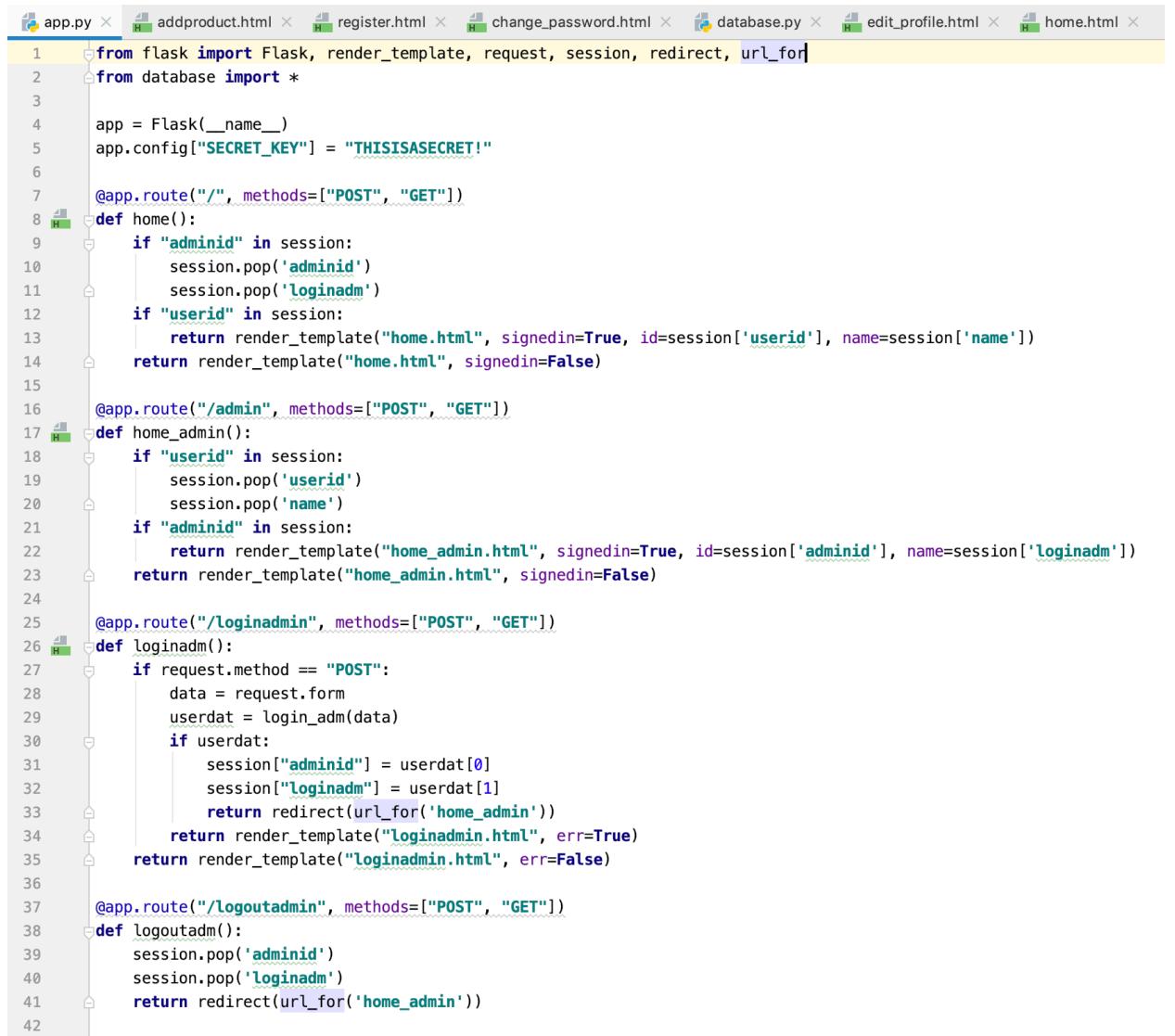
Проект «flask-shop» содержит:

- Модуль app.py, в котором происходит запуск локального сервера с приложением;

- Модуль database.py, в котором находятся функции, взаимодействующие с базой данных и редактирующие их содержимое;
- Базу данных shop.db, реализованную с помощью SQLite3;
- Директорию templates, которая содержит html-файлы;
- Директорию static, в которой расположены файлы для оформления внешнего вида веб-страниц;

### 3.3. Описание функционала приложения

Для запуска приложения необходимо запустить файл app.py. Сервер запустится на <http://127.0.0.1:8557>. На рис.9 представлены некоторые функции модуля app.py:



```

1  from flask import Flask, render_template, request, session, redirect, url_for
2  from database import *
3
4  app = Flask(__name__)
5  app.config["SECRET_KEY"] = "THISISASECRET!"
6
7  @app.route("/", methods=["POST", "GET"])
8  def home():
9      if "adminid" in session:
10          session.pop('adminid')
11          session.pop('loginadm')
12      if "userid" in session:
13          return render_template("home.html", signedin=True, id=session['userid'], name=session['name'])
14      return render_template("home.html", signedin=False)
15
16  @app.route("/admin", methods=["POST", "GET"])
17  def home_admin():
18      if "userid" in session:
19          session.pop('userid')
20          session.pop('name')
21      if "adminid" in session:
22          return render_template("home_admin.html", signedin=True, id=session['adminid'], name=session['loginadm'])
23      return render_template("home_admin.html", signedin=False)
24
25  @app.route("/loginadmin", methods=["POST", "GET"])
26  def loginadm():
27      if request.method == "POST":
28          data = request.form
29          userdat = login_adm(data)
30          if userdat:
31              session["adminid"] = userdat[0]
32              session["loginadm"] = userdat[1]
33              return redirect(url_for('home_admin'))
34          return render_template("loginadmin.html", err=True)
35      return render_template("loginadmin.html", err=False)
36
37  @app.route("/logoutadmin", methods=["POST", "GET"])
38  def logoutadm():
39      session.pop('adminid')
40      session.pop('loginadm')
41      return redirect(url_for('home_admin'))
42

```

Рис.9 Некоторые функции модуля app.py

Таблица 1 содержит названия всех функций модуля app.py, их возвращаемые значения, а так же url, к которому привязывается та или иная функция с помощью декоратора route() :

URL	Название функции	Возвращаемое значение
/	home()	render_template('home_admin.html')
/admin	home_admin()	home_admin.html
/loginadmin	loginadm()	loginadmin.html
/logautadmin	logautadm()	Redirect('/admin')
/addproducts	addproducts()	addproduct.html
/product/delete/<id>	deleteproduct(id)	Redirect ('/admin/products')
/admin/products	searchprod_admin()	searchprod_admin.html
/register	register()	register.html
/login	login()	login.html
/logout	logout()	redirect('/')
/viewprofile/<id>	view_profile(id)	view_profile.html
/myprofile	my_profile()	Redirect ('/view_profile/myid')
/changepassword	change_password()	change_password.html
/editprofile/	edit()	edit_profile.html
/buy/	buy()	search_prod.html
/product/<id>	product_info(id)	product_info.html
/product/addtocart/<pid>	addtocart(pid)	redirect(url_for('viewcart'))
/cart	viewcart()	cart.html

Таблица 1

Вместо написания триггеров и процедур непосредственно в базе данных , их логика была реализована с помощью функций в файле database.py, Рассмотрим некоторые из них.

```
def add_user(data):
    conn = sqlite3.connect("shop.db")
    cur = conn.cursor()
    email = data["email"]
    a = cur.execute("SELECT * FROM user_ WHERE email=?", (email,))
    if len(list(a))!=0:
        return False
    data_tuple = (data["name"],data["email"], data["phone"], data["city"], data["country"], data["zip"], data["password"], data["check"])
    cur.execute("INSERT INTO user_(name, email, phone, city, country, zipcode, password, public_access) VALUES (?,?,?,?,?,?,?,?)",(*data_tuple,))
    bb = cur.execute("SELECT * FROM user_")
    rows = bb.fetchall()
    for row in rows:
        print(row)
    conn.commit()
    conn.close()
    return True
```

*Рис.10 Функция add\_user(data) , файл database.py*

Функция add\_user(data) (рис.10) отвечает за добавление в базу данных нового пользователя. В первой строчке происходит соединение с базой данной shop.db, которая хранит в себе информацию о пользователях, администраторах, товарах и корзине. Далее в переменную email кладется информация об электронной почте, которая получена из формы регистрации пользователя на сайте. С помощью cur.execute() делается SELECT запрос к таблице, который использует обычный SQL-синтаксис. Это делается для того, чтобы понять, есть ли в базе данных уже пользователем с заданным email. Если да, тогда функция возвращает False. Если нет, то в таблицу user с заданным email вставляются данные из формы на сайте. con.commit() сохраняет транзакцию после внесения изменений в таблицу.

```

def add_to_cart(prodID, userID, number=1):
    conn = sqlite3.connect('shop.db')
    cur = conn.cursor()
    a = cur.execute("SELECT * FROM cart WHERE userID=? AND prodID=?", (userID, prodID))
    rows = a.fetchall()
    if len(rows) == 0:
        cur.execute("INSERT INTO cart VALUES (?, ?, ?)", (userID, prodID))
        conn.commit()
    else:
        a = cur.execute("SELECT quantity FROM cart WHERE userID=? AND prodID=?", (userID, prodID))
        rows = a.fetchall()
        q = rows[0]
        quantity = q[0]
        cur.execute("UPDATE cart SET quantity=? WHERE userID=? AND prodID=?", (quantity + number, userID, prodID))
        conn.commit()
    conn.close()

```

*Рис.11 Функция add\_to\_cart, модуль database.py*

Функция add\_to\_cart отвечает за добавление товаров в корзину пользователя. На вход ей передается id добавляемого продукта, id пользователя и количество товара (если последнее число явно не указывается, то по умолчанию добавляется 1 товар). Аналогично с предыдущей функцией , в начале происходит соединение с базой данных shop.db. Делается запрос к таблице cart, выбираются записи, которые имеют заданный userID и prodID, результат сделанного запроса кладется в переменную rows. Если len(rows)==0, то есть пользователь не добавлял в корзину себе указанный товар, то в cart добавляется товар с заданным количеством. Если товар уже хранится в корзине, и пользователь добавил его еще раз, то в начале узнается количество товара в корзине с помощью select-запроса, запоминается в памяти, а затем происходит увеличение количества товара ровно на ранее полученное число.

Для реализации удаления конкретного продукта из базы данных была написана функция del\_prod(prodID) (рис.12) принимающая на вход id продукта. В теле функции делаются 2 запроса к базе данных, которые удаляют продукт с указанным идентификатором продукта из таблиц cart и product.

```

def del_prod(prodID):
    conn = sqlite3.connect("shop.db")
    cur = conn.cursor()
    cur.execute("DELETE FROM CART WHERE prodID=?", (prodID,))
    conn.commit()
    cur.execute("DELETE FROM product WHERE prodID=?", (prodID,))
    conn.commit()
    conn.close()

```

*Рис.12 Функция del\_prod(prodID), модуль database.py*

Функция product\_info\_db(id) принимает на вход id товара (рис.13). Делается select-запрос, в котором выбираются только те записи таблицы, в котором prodID равен указанному id. Возвращается либо запись со всеми данными товара с указанным идентификатором, либо False.

```

def product_info_db(id):
    conn = sqlite3.connect('shop.db')
    cur = conn.cursor()
    a = cur.execute("SELECT * FROM product WHERE prodID=?", (id,))
    rows = a.fetchall()
    for row in rows:
        print(row)
    return rows[0]

```

*Рис.13 Функция product\_info\_db(id), модуль database.py*

### 3.4. Работа с реализованным приложением

Для работы с приложением необходимо установить в систему(если этого не было сделано заранее):

- язык Python версии 3.0 или выше
- микрофреймворк Flask

Так как Python имеет встроенную поддержку SQLite базы данных, дополнительного устанавливать для взаимодействия с БД ничего не нужно.

Требуется загрузить все исходные файлы и собрать проект. Чтобы собрать и запустить проект из командной строки, нужно выполнить следующую последовательность команд (для операционной системы macOS):

```
cd ./flask-shop  
. venv/bin/activate  
python3 app.py
```

## 4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Для тестирования было решено взять функцию добавления нового пользователя в систему (базу данных). Изначально данные таблицы user выглядели следующим образом(рис.14):

Таблица:		user	Добавить запись	Удалить запись	
userID	name	email	phone	city	country
Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	Екатерина Гим...	kate.gimranova...	89621085576	Южно-Сахалинск	Россия
2	Ольга Васильева	suryanshsharma...	8448341570	Львов	Украина
3	Random Custo...	randomcustomere...	1234567890	New York	USA
4	Алиса Королева	vishwas.latiyan0...	8963063830	Санкт-Петербург	Россия
5	Анна Сорова	anyanya@mail.ru	8987654321	Москва	Россия
6	Максим Соловьев	abc@gmail.com	9900990099	Самара	Россия

Рис.14 Данные таблицы user до добавления пользователя

После перехода по следующему url: <http://127.0.0.1:8557/register/>. необходимо заполнить форму . Поля формы и вид страницы можно увидеть на рис. 15:

The screenshot shows a registration form titled 'Создание аккаунта пользователя' (Create account user). The form fields are as follows:

- Имя: Анна Кириллова
- Email: annkir@gmail.com
- Телефон: 89145567733
- Город: Красноярск
- Страна: Россия
- Индекс: 600030
- Пароль: ..... (redacted)
- Пароль (повтор): veryhardpar1
- Разрешить публичный доступ:
- Создать:

Рис.15 Страница <http://127.0.0.1:8557/register/>

После нажатия кнопки «Создать» в базу данных должен быть добавлен пользователь с внесенными в форму данными.

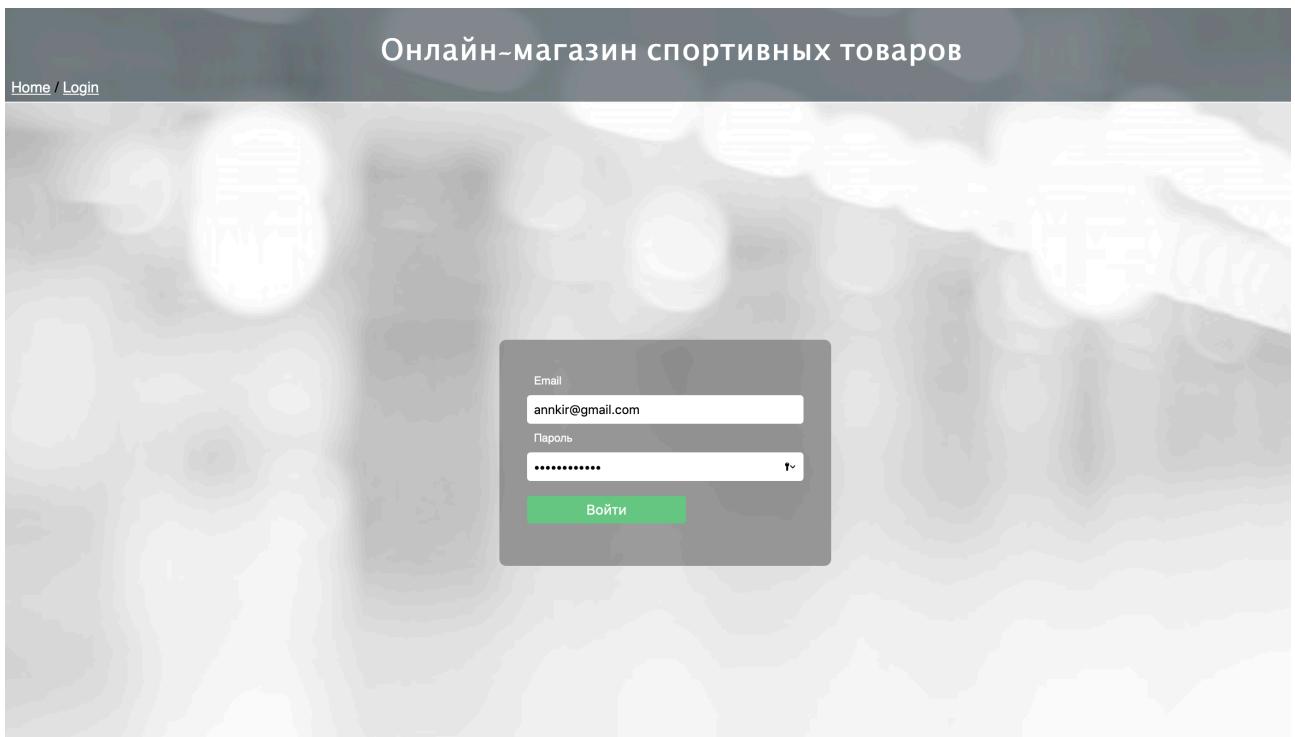


Рис.16 Ввод данных пользователя на странице <http://127.0.0.1:8557/login/>

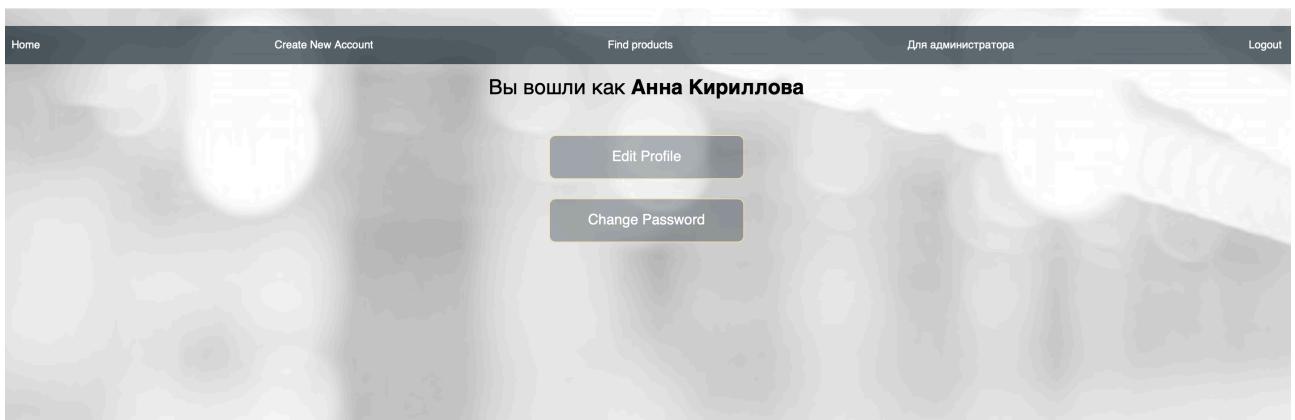


Рис.17 Результат успешного входа на <http://127.0.0.1:8557>

По рис.16-17 можно увидеть, что вход в систему под заданными email и паролем прошли успешно, что говорит о корректной работе функции добавления нового пользователя в базу данных. Также это подтверждает SQL-

запрос «*SELECT \* FROM user*» (последняя запись содержит данные о крайнем добавленном пользователе)(рис.18)

```
127.0.0.1 -- [29/Sep/2019 03:38:24] "GET /login/ HTTP/1.1" 200 -
127.0.0.1 -- [29/Sep/2019 03:38:25] "POST /login/ HTTP/1.1" 302 -
(1, 'Екатерина Гимранова', 'kate.gimranova@mail.ru', '89621085576', 'Южно-Сахалинск', 'Россия', '110059', 'abcd1234', 'off')
(2, 'Ольга Васильева', 'suryanshsharma@gmail.com', '8448341570', 'Львов', 'Украина', '110059', '16122004', 'off')
(3, 'Random Customer', 'randomcustomer@rediffmail.com', '1234567890', 'New York', 'USA', '987654', 'abcd1234', 'on')
(4, 'Алиса Королева', 'vishwas.latiyan007@gmail.com', '8963063830', 'Санкт-Петербург', 'Россия', '420420', 'qwerty', 'on')
(5, 'Анна Сорова', 'anyanya@mail.ru', '8987654321', 'Москва', 'Россия', '776611', 'qwerty', 'on')
(6, 'Максим Соловьев\n', 'abc@gmail.com', '9900990099', 'Самара', 'Россия', '110190', 'qwerty', 'off')
(15, 'Анна Кириллова', 'annkir@gmail.com', '89145567733', 'Красноярск', 'Россия', '600030', 'veryhardpar1', 'on')
```

*Рис.18 Выполнение запроса SELECT \* FROM user*

Произведем также тестирование функции добавления нового товара в базу данных администратором. Выполним запрос «*SELECT \* FROM product*», чтобы увидеть данные о товарах, которые находятся в системе до добавления (рис.19)

```
127.0.0.1 -- [29/Sep/2019 14:54:06] "GET /addproducts HTTP/1.1" 200 -
(11, 'Куртка утепленная женская Outventure ', 15, 'Одежда для спорта', 1500, 'Куртка утепленная женская Outventure, цвет "Красный", размер 30', 1)
(12, 'Куртка утепленная женская Outventure (32)', 10, 'Одежда для спорта', 1500, 'Куртка утепленная красного цвета, артикул P0032112cl', 2)
(13, 'Куртка утепленная женская Outventure (36)', 15, 'Одежда для спорта', 1500, 'Куртка утепленная красного цвета, артикул P0032113cl', 2)
(14, 'Куртка утепленная женская Outventure (40)', 3, 'Одежда для спорта', 1500, 'Джемпер флисовый женский Termit синего цвета, артикул P0032117cl', 1)
(15, 'Джемпер флисовый женский Termit', 3, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termit синего цвета, артикул P0032115cl', 1)
(16, 'Джемпер флисовый женский Termit (44)', 2, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termit синего цвета, артикул P0032116cl', 1)
(17, 'Джемпер флисовый женский Termit (34)', 5, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termit синего цвета, артикул P0032117cl', 2)
(18, 'Велосипед подростковый Stern Attack 20"', 5, 'Велоспорт', 6999, 'Артикул 12321', 2)
(19, 'Велосипед городской женский Stern Urban 1.0 Lady 28"', 3, 'Велоспорт', 11990, 'Артикул 42199', 3)
(20, 'Велосипед складной Stern Compact 24"', 7, 'Велоспорт', '17 999', 'Артикул 112233', 3)
(21, 'Велосипед горный CUBE REACTION C:62 29"', 2, 'Велоспорт', '128 99', 'Артикул 11223344', 1)
(22, 'Беговел Stern Kidster', 3, 'Велоспорт', 11200, 'Артикул 118896', 2)
(23, 'Велосипед для девочек Stern Bunny 12"', 2, 'Велоспорт', 11220, 'Артикул 12312', 2)
127.0.0.1 -- [29/Sep/2019 14:57:54] "GET /addproducts HTTP/1.1" 200 -
```

*Рис.19 Выполнение запроса SELECT \* FROM product*

Запускаем веб-приложение (сервер запускается на <http://127.0.0.1:8557>).

Перейдем на <http://127.0.0.1:8557/admin>. (рис.20)

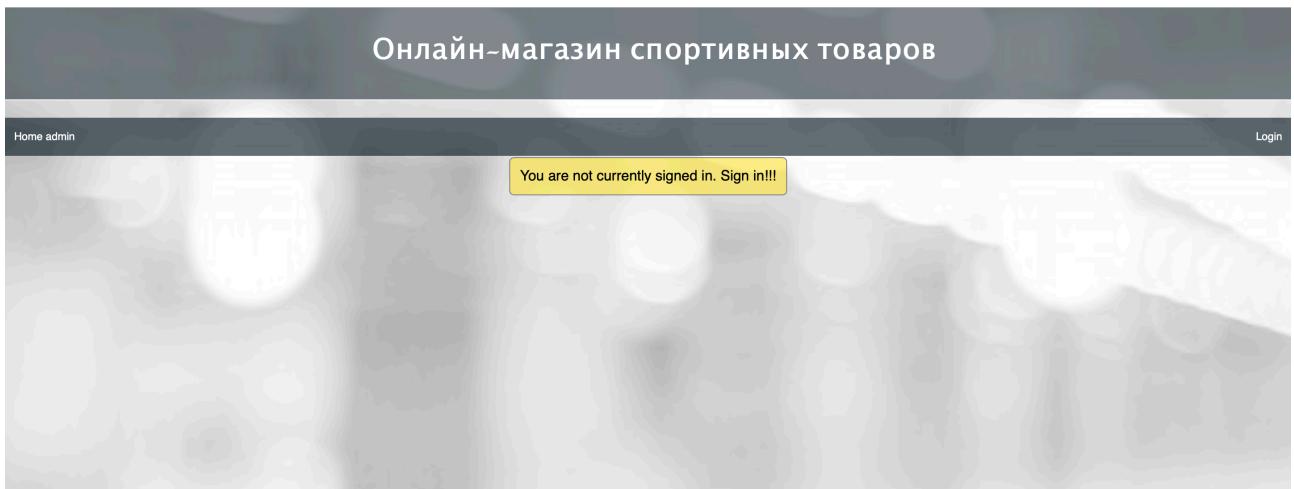


Рис.20 Страница <http://127.0.0.1:8557/admin>

Видим, что для не вошедших администраторов доступна только функция входа в систему.

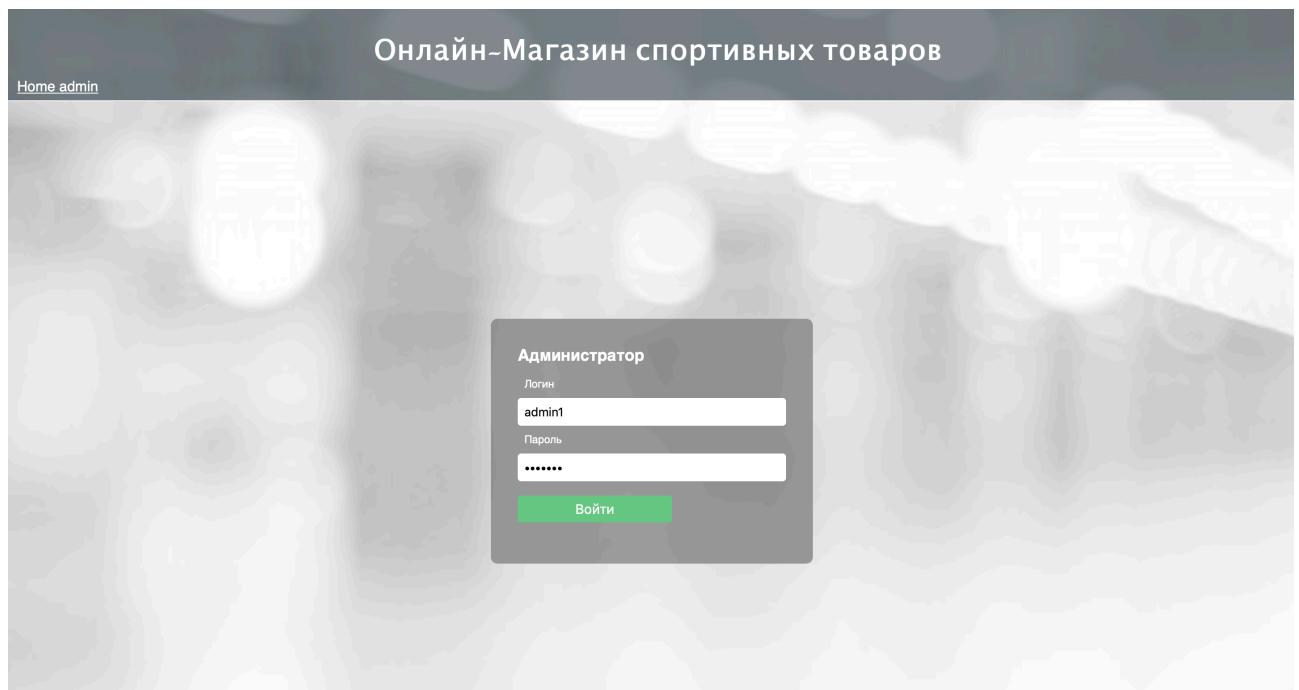
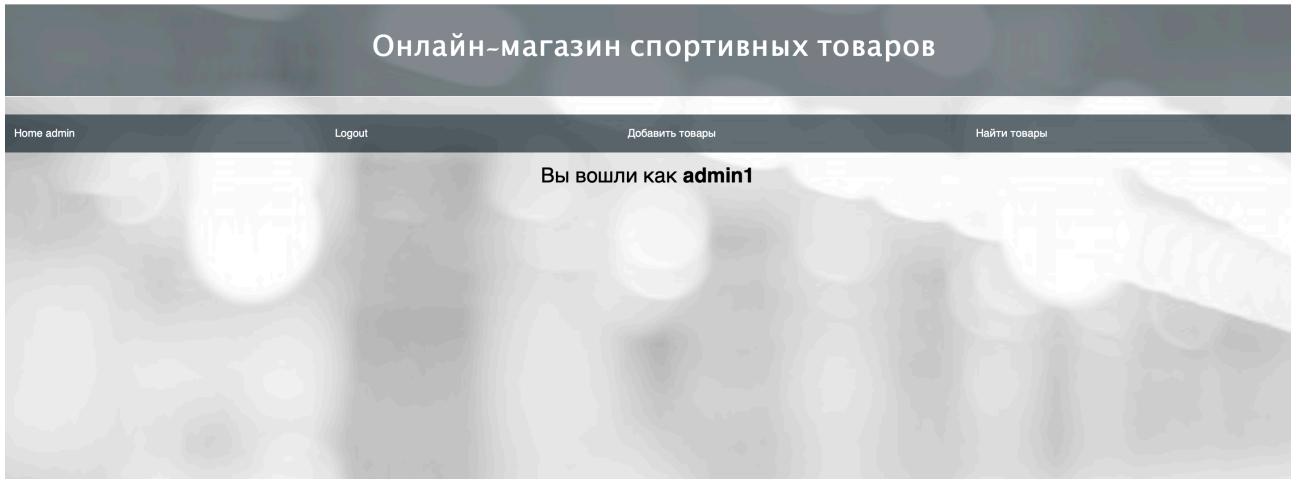


Рис.21 Форма входа администратора <http://127.0.0.1:8557/loginadmin>



*Рис.22 Успешный вход в систему в качестве администратора*

После входа в качестве администратора видим, что возможностей для администратора стало больше: появились доступные ссылки поиска товаров и их добавления в систему(рис.22). Нас интересует последняя.

Далее после перехода на страницу в форму были введены следующие данные добавляемого товара(рис.23):

The screenshot shows the 'Добавление нового товара' (Adding a new product) form. It includes fields for 'Категория товара' (Sportswear), 'Название' (Nike Victory Compression Bra), 'Цена' (1799), and 'Описание товара' (Articule: 3758332-X1). A green 'Добавить товар' (Add product) button is at the bottom.

*Рис. 23 <http://127.0.0.1:8557/addproducts>*

Убедимся, что после нажатия клавиши «Добавить товар» указанный товар был в действительности добавлен в базу данных. Для этого вновь сделаем запрос «*SELECT \* FROM product*»(рис.24)

```
127.0.0.1 - - [29/Sep/2019 15:29:37] "POST /loginadmin HTTP/1.1" 302 -
127.0.0.1 - - [29/Sep/2019 15:29:37] "GET /admin HTTP/1.1" 200 -
(11, 'Куртка утепленная женская Outventure ', 15, 'Одежда для спорта', 1500, 'Куртка утепленная женская Outventure, цвет "Красный", размер 30', 1)
(12, 'Куртка утепленная женская Outventure (32)', 10, 'Одежда для спорта', 1500, 'Куртка утепленная красного цвета, артикул P0032112cl', 2)
(13, 'Куртка утепленная женская Outventure (36)', 15, 'Одежда для спорта', 1500, 'Куртка утепленная красного цвета, артикул P0032113cl', 2)
(14, 'Куртка утепленная женская Outventure (40)', 3, 'Одежда для спорта', 1500, 'Джемпер флисовый женский Termite синего цвета, артикул P0032117cl', 1)
(15, 'Джемпер флисовый женский Termite', 3, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termite синего цвета, артикул P0032115cl', 1)
(16, 'Джемпер флисовый женский Termite (44)', 2, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termite синего цвета, артикул P0032116cl', 1)
(17, 'Джемпер флисовый женский Termite (34)', 5, 'Одежда для спорта', 1300, 'Джемпер флисовый женский Termite синего цвета, артикул P0032117cl', 2)
(18, 'Велосипед подростковый Stern Attack 20"', 5, 'Велоспорт', 6999, 'Артикул 12321', 2)
(19, 'Велосипед городской женский Stern Urban 1.0 Lady 28"', 3, 'Велоспорт', 11990, 'Артикул 42199', 3)
(20, 'Велосипед складной Stern Compact 24"', 7, 'Велоспорт', 17999, 'Артикул 112233', 3)
(21, 'Велосипед горный CUBE REACTION C:62 29"', 2, 'Велоспорт', 12899, 'Артикул 11223344', 1)
(22, 'Беговел Stern Kidster', 3, 'Велоспорт', 11200, 'Артикул 118896', 2)
(23, 'Велосипед для девочек Stern Bunny 12"', 2, 'Велоспорт', 11220, 'Артикул 12312', 2)
(24, 'Бра Nike Victory Compression', 100, 'Одежда для спорта', 1799, 'Артикул: 3758332-X1', 1)
127.0.0.1 - - [29/Sep/2019 15:29:50] "GET /addproducts HTTP/1.1" 200 -
```

*Ruc.24 SELECT \* FROM product*

Последняя запись хранит информацию о добавленном товаре, id товара равен 24.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта были написаны база данных интернет-магазина и веб-приложение, которое воспроизводит многие функции работы реального онлайн-магазина.

Работая над данным проектом, я открыла для себя новые возможности языка Python 3, изучила основы микрофреймворка Flask, Jinja2, а так же необходимых для разработки библиотек.

Дальнейшим развитием данного проекта может быть увеличение функциональности приложения, например, добавление проверки корректности почты, уровня сложности пароля и многих других функций.

## **СПИСОК ЛИТЕРАТУРЫ**

- 1) Кузнецов С. Д. Основы баз данных. — 2-е изд. — М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. — 484 с.
- 2) Date, C. J. Date on Database: Writings 2000–2006. — Apress, 2006. — 566 с. — ISBN 978-1-59059-746-0, 1-59059-746-X.
- 3) Документация Flask (русский перевод) 0.10.1: <https://flask-russian-docs.readthedocs.io/ru/latest/index.html>
- 4) Grinberg M. Flask Web Development. Developing web applications with Python — O'Reilly Media, 2014—258 p
- 5) Самоучитель Sqlite <https://sites.google.com/site/javatokens/sqlite>