

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное образовательное учреждение высшего образования  **«Дальневосточный федеральный университет»**  **(ДВФУ)** |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**  **Департамент математического и компьютерного моделирования** |
| **О Т Ч Е Т**  о прохождении учебной практики.  Технологическая (проектно-технологическая) практика  направление подготовки 02.03.01 «Математика и компьютерные науки»  профиль «Сквозные цифровые технологии» |

|  |  |
| --- | --- |
| Отчет защищена  с оценкой \_\_\_\_\_\_\_\_ | Выполнил студент  группы № Б9121-02.03.01сцт  Москера Креспо Адриан Хосуэ Ayudas-removebg-preview  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |
| Рег. № \_\_\_\_\_\_\_\_\_\_\_ | Руководитель практики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(должность, уч.звание)*  **\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(Ф.И.О.) (подпись)*  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |
| «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г. | Практика пройдена в срок  с «18» июля 2022 г.  по «30» июля 2022 г.  (2 недели) |

г. Владивосток

2022

Содержание

[Введение 1](#_Toc109167896)

[Проект 2](#_Toc109167897)

[Настройка 2](#_Toc109167898)

[Python и объектно-ориентированное программирование (ООП) 3](#_Toc109167899)

[Flask 4](#_Toc109167900)

[Базовое приложение 4](#_Toc109167901)

[Шаблоны и Джинджа 5](#_Toc109167902)

[Другие инструменты от Flask 6](#_Toc109167903)

[Функция redirect 6](#_Toc109167904)

[Функция url\_for 7](#_Toc109167905)

[Переменная request 7](#_Toc109167906)

[Переменная session 7](#_Toc109167907)

[Реализация проекта 8](#_Toc109167908)

[База данных 8](#_Toc109167909)

[Класс – Database 8](#_Toc109167910)

[Класс - Cart 9](#_Toc109167911)

[Класс - User 9](#_Toc109167912)

[Класс - Product 10](#_Toc109167913)

[App 10](#_Toc109167914)

[Вспомогательные функции 11](#_Toc109167915)

[Глобальные переменные 12](#_Toc109167916)

[Index 12](#_Toc109167917)

[Sign up 12](#_Toc109167918)

[Sign in 13](#_Toc109167919)

[Success 13](#_Toc109167920)

[Logout 13](#_Toc109167921)

[Add product 13](#_Toc109167922)

[Cart 14](#_Toc109167923)

[Thank you 14](#_Toc109167924)

[Заключение 15](#_Toc109167925)

[Список литературы 16](#_Toc109167926)

[Сноски 17](#_Toc109167927)

[Приложения 18](#_Toc109167928)

Введение

Веб-разработка - это термин, который относится ко всей работе, связанной с созданием, созданием и поддержкой приложения, работающего через Интернет, то есть веб-сайта. Она включает в себя множество аспектов, таких как дизайн, публикация, программирование и управление базами данных. И каждый из этих аспектов исследуется в этом проекте, в котором применяется на практике все, что было изучено до сих пор.

Целью этого проекта было создание веб-приложения для интернет-магазина под названием «Shopee». Это означало рассмотреть все аспекты, связанные с созданием веб-сайта, о которых упоминалось ранее. Для клиентской части1 приложения, использовались HTML2 и CSS3, а также иконки и шрифт, предоставленные с использованием бесплатной версии проекта с открытым исходным кодом Font Awesome4. Серверная часть5 была разработана с использованием языка программирования Python с использованием Flask в качестве фреймворка и реализации объектно-ориентированного программирования для создания базы данных для хранения пользовательских данных в виде файлов JSON6.

При разработке веб-сайта присутствовали некоторые ограничения. Язык был одной из них, и она была решена путем принятия решения сделать интерфейс на английском языке вместо русского. Знание инструментов, используемых для этого приложения, также было проблемой, но ее удалось преодолеть, разделив задачу бэкэнда с участником, который знал, как использовать Python, а другой участник выполнял бы интерфейс.

Проект

Чтобы начать разработку, было решено назвать интернет-магазин «Shopee». А продукты, используемые на веб-сайте, будут представлять собой стеклянные бутылки, которые обладают многими свойствами, упомянутыми ниже. Для его разработки мы будем использовать HTML, CSS и Python. Чтобы лучше понять ход реализации проекта, важно описать инструменты и требования, необходимые для достижения поставленной цели.

В этом разделе будет объяснено использование Python и Flask для разработки серверной части.

# Настройка

Для начала необходима установка Python. Для этого проекта 64-разрядный Python 3.10.5 использовался с Visual Studio Code в качестве текстового редактора в Windows 11. Чтобы иметь больший контроль над библиотеками, используемыми в проекте, и упростить создание файла требований, для него была создана виртуальная среда.

Чтобы создать его, сначала был создан новый каталог с именем «Shop». Здесь будут храниться все файлы, соответствующие веб-сайту. В каталоге созданной новой папки виртуальная среда может быть создана с помощью следующих команд в зависимости от используемой операционной системы. После настройки виртуальной среды в каталоге появится новая папка с именем «.venv», как показано в Приложении 1, которая содержит библиотеки, используемые в приложении.

|  |  |
| --- | --- |
| **Команды в консоли для создания виртуальной среды (для Python 3)** | |
| Linux | sudo apt-get install python3-venv # При необходимости  python3 -m venv .venv  source .venv/bin/activate |
| macOS | python3 -m venv .venv  source .venv/bin/activate |
| Windows | py -3 -m venv .venv  .venv\scripts\activate |

Сменив интерпретатор с установки Python на виртуальную среду, переходим к следующему шагу - обновлению pip и загрузке Flask.

|  |
| --- |
| **Обновление pip и установка Flask** |
| python -m pip install --upgrade pip  python -m pip install flask |

Установленный пакет Flask имеет версию 2.1.2, которая поддерживает Python 3.7 или новее. Он также устанавливает зависимости, которые являются другими пакетами, необходимыми для его правильной работы, такими как: Werkzeug, Jinja, MarkupSafe, ItsDangerous и Click. Функции, выполняемые этими зависимостями, за исключением Jinja, не входят в объем этой работы. (Pallets, 2010)

Эти шаги, приведенные в руководстве Visual Studio Code для Flask (Microsoft, 2021), являются основой, на которой будет построено веб-приложение.

# Python и объектно-ориентированное программирование (ООП)

Перед использованием Flask будет использоваться ООП, поэтому уместно дать общий обзор того, как Python управляет объектами и свойствами, которыми они обладают.

В Python классы можно использовать для создания объектов класса, которые поддерживают ссылки на атрибуты и создание экземпляров. Последнее относится к вызову объекта класса для создания объекта экземпляра. Ссылки на атрибуты - это единственная операция, понятная объектам экземпляра, где единственными допустимыми именами атрибутов являются атрибуты данных и методы. (Python Software Foundation, 2022)

Короче говоря, экземпляр класса может иметь некоторые атрибуты данных (например, число, строку, массив и т.д.) или метод (который представляет собой функцию, определенную внутри класса). Следующий пример определяет класс MyClass и определяет начальный атрибут «data» для каждого экземпляра с помощью метода \_\_init\_\_, а также создает метод f, который возвращает атрибут data, который в данном случае представляет собой список, в обратном порядке порядок. Как видно, создав экземпляр класса в переменной x, можно получить доступ к значениям, хранящимся для этого экземпляра, и использовать определенные для него методы. Это будет полезно в ходе реализации проекта.

|  |
| --- |
| **Пример класса в Python** |
| >>> class MyClass:  ... def \_\_init\_\_(self):  ... self.data = [1,2,3]  ... def f(self):  ... return self.data[::-1]  ...  >>> x = MyClass() # создание экземпляра  >>> x.data # атрибут  [1, 2, 3]  >>> x.f() # метод  [3, 2, 1] |

# Flask

## Базовое приложение

Flask станет основой, на которой будет построен этот проект. Его основное использование можно описать следующим образом.

Чтобы запустить новое приложение, создайте файл python с именем «app.py », в котором Flask импортируется из пакета flask и создается экземпляр объекта Flask с именем «app». Для возврата содержимого по определенному URL-адресу создаются функции с помощью декоратора app.route от Flask для сопоставления маршрутов URL, указанных для этой конкретной функции. В приведенном ниже примере это код базового приложения, которое при запуске возвращает строку текста. (Microsoft, 2021)

|  |
| --- |
| **Код «app.py»** |
| from flask import Flask  app = Flask(\_\_name\_\_)  @app.route("/")  def index():  return "Hello world!" |

Этот файл Python при запуске открывает порт и, пока он не будет остановлен с помощью Ctrl+C в командной консоли, показывает в браузере веб-сайт и возвращает содержимое каждой функции для каждого указанного маршрута URL. В этом примере после запуска файла с помощью приведенной ниже команды появляется сообщение с указанием соответствующего URL-адреса и того, как остановить запуск программы. В Приложении 2 видно, что отображается в браузере. Это основная идея того, что сделано для того, чтобы Flask возвращал содержимое на определенный URL-адрес, и это основа работы, стоящей за этой работой.

|  |
| --- |
| **Журнал консоли при запуске приложения** |
| (.venv) D:\Shop> python -m flask run  \* Environment: production  WARNING: This is a development server. Do not use it in a production deployment.  Use a production WSGI server instead.  \* Debug mode: off  \* Running on http://127.0.0.1:5000 (Press CTRL+C to quit) |

При желании можно добавить этот фрагмент кода в конец кода, чтобы при обычном запуске файла использование консоли не требовалось. Здесь используется метод app.run, если запущен «app.py». Кроме того, есть режим отладки, в котором ошибкам присваиваются имена и показано их местонахождение в коде для облегчения решения проблем, что объясняет аргумент debug=true в app.run.

|  |
| --- |
| **Фрагмент кода для запуска приложения при запуске файла** |
| if \_\_name\_\_ == "\_\_main\_\_":  app.run(debug=True) |

## Шаблоны и Джинджа

Базовое созданное приложение может генерировать веб-страницы только с простым текстом. Чтобы включить HTML в приложение, можно написать HTML-код в одной из используемых функций. Это оставляет приложение открытым для атак с использованием межсайтовых сценариев, когда злоумышленник может разместить вредоносный HTML или даже Javascript в URL-адресе, который в некоторых приложениях используется внутри функций, чтобы вредоносный код запускался приложением.

Хорошей практикой является удаление HTML из кода с помощью шаблонов. Шаблоны — это HTML-файлы, содержащие заполнители для значений, предоставляемых кодом во время выполнения. Затем работа разделяется, код заботится о значениях данных, а шаблоны касаются только разметки. Механизм шаблонов по умолчанию для Flask — Jinja. Это дает возможность вводить данные из кода в файл HTML, кодировать с синтаксисом Jinja в файлах HTML и использовать наследование. С помощью наследования можно создать базовую страницу для определения общей разметки, а затем на ее основе можно добавлять дополнения для конкретных страниц. Обычно он служит для объявления заголовка и включения статических файлов для всех страниц. Чтобы использовать это, необходимо импортировать функцию render\_template. (Microsoft, 2021)

## Другие инструменты от Flask

Чтобы реализовать некоторые функции, необходимые в приложении, необходимо импортировать другие функции и переменные в пакет flask. Оператор импорта будет выглядеть так, как показано ниже.

|  |
| --- |
| **Оператор импорта пакета flask** |
| from flask import Flask, render\_template, redirect, url\_for, request, session |

### Функция redirect

Перенаправляет с текущего URL на указанный. Это может быть любой допустимый URL-адрес.

### Функция url\_for

Создает URL-адрес заданной конечной точки с помощью предоставленного метода. Конечная точка — это строка, обычно указывающая на страницу без необходимости обновлять корневой URL-адрес приложения в коде и шаблонах.

### Переменная request

Запрос имеет несколько доступных методов. Для этого приложения будут использоваться GET и POST. POST используется для запроса данных, так как сервер получает данные через метод POST в используемом шаблоне. GET используется для получения данных, полученных сервером. Чтобы указать методы, используемые каждой страницей, ее необходимо включить в декоратор app.route. (https://pythonbasics.org, 2021)

### Переменная session

Во Flask сеанс — это словарь, используемый для хранения информации, связанной с пользователем, по различным запросам. Сохраненные данные являются временными, так как сеанс завершится через определенный период времени. Для постоянного хранения информации будет использоваться база данных.

Реализация проекта

# База данных

Для начала на веб-сайте нам нужно сохранить учетные данные пользователей магазина, товары и товары, сохраненные в их корзинах до того, как они их купили. Для этого мы будем использовать файлы JSON и объектно-ориентированное программирование. Создание и обновление файлов, принадлежащих базе данных, будет управляться как модуль в папке. Итак, файл db.py был создан для разработки всех необходимых классов для этой задачи.

Для этого файла необходимы некоторые модули, которые являются pathlib и json, оба включены в стандартную библиотеку Python. Path из модуля pathlib превращает строку в допустимый объект Path, который имеет некоторые свойства, используемые позже. Модуль json позволяет нам легко работать с файлами json, используя типы данных в Python со структурами, аналогичными элементам, составляющим файлы json.

|  |
| --- |
| **Оператор импорта для необходимых модулей** |
| from pathlib import Path  import json |

Каждая база данных - это просто список элементов. Индекс объекта в списке - это его идентификатор. Это помогает в случае сопоставления каждой корзины из их базы данных с соответствующим пользователем.

## Класс – Database

(См. Приложение 3)

Чтобы создать экземпляр объекта Database, необходимо указать путь к нужной базе данных в виде строки. Как видно из метода \_\_init\_\_, эта строка превращается в экземпляр объекта Path и сохраняется как атрибут path объекта Database. После этого объекты Path имеют метод touch, который пытается получить доступ к указанному пути. Если он не найден, он создаст файл по указанному пути.

Метод write\_db принимает список в качестве аргумента и не возвращает никакого значения. Он открывает указанный файл, используя его путь, и записывает список входных данных в файл, используя отступы для лучшей читаемости. Метод read\_db используется для извлечения данных, хранящихся в файле. Если формат в файле неверен, метод reader перезапишет содержимое файла в пустую таблицу и вернет ее. С другой стороны, этот метод возвращает список объектов, хранящихся в файле.

## Класс - Cart

(См. Приложения 4, 5)

Этот класс используется для хранения представления корзины магазина в виде списка, в котором хранятся идентификаторы содержащихся в ней товаров. У него есть два атрибута: сам список корзины и его собственная база данных, экземпляр класса Database для файла, в котором будут храниться данные. Если метод read\_db базы данных возвращает пустой список, для базы данных будет установлено значение по умолчанию. Это значение базы данных по умолчанию содержит пустые корзины USER и ADMIN. Использование этих пользователей будет объяснено позже.

Методы db\_view, db\_commit и db\_add предназначены для извлечения данных из базы данных, записи списка в качестве обновления базы данных и добавления /обновления текущего объекта в базу данных с возможностью указания его идентификатора в случае, если корзина от уже зарегистрированного пользователя.

Также мы можем обновить значение атрибута корзины, указав допустимый идентификатор с помощью parse\_by\_id, добавить товар в корзину и обновить базу данных с помощью add\_product, а также очистить товары в корзине и обновить их в базе данных.

## Класс - User

(См. Приложения 6, 7)

У каждого пользователя есть имя пользователя, пароль, корзина и база данных. Каждое имя пользователя хранится в виде строки в верхнем регистре, чтобы предотвратить существование двух одинаковых имен с разным регистром. Пароль хранится в виде кода, составленного путем объединения кодов ASCII7 каждого символа пароля в целое число. Корзина является экземпляром класса Cart, и, как использовалось ранее, база данных является экземпляром класса Database с путем к базе данных пользователей. Эта база данных также имеет значения по умолчанию, пользователи USER и ADMIN. И эта пользовательская база данных по умолчанию совпадает с базой данных корзины по умолчанию.

Методы, которые работают с базой данных, выполняют те же функции, что и в классе Cart. Метод to\_dict возвращает словарь с атрибутами пользователя; parse получает словарь и обновляет значения атрибутов объекта; in\_db проверяет, находится ли пользователь в базе данных по имени пользователя; get\_id возвращает идентификатор пользователя, если он находится в базе данных; pwd\_check возвращает, если пользователь находится в базе данных. в базе данных и если пароль экземпляра совпадает с паролем, хранящимся в базе данных.

## Класс - Product

(См. Приложения 8, 9)

Каждый продукт, помимо своей собственной базы данных, имеет строковые атрибуты category, name, mold и color, а также целочисленные атрибуты capacity, height и price. Методы, показанные в коде, выполняют те же функции, что и в предыдущих классах.

# App

В главном файле app.py , flask уже была импортирована. Чтобы использовать базы данных, нам также необходимо импортировать классы Card, User и Product из bd.py файл в том же каталоге.

|  |
| --- |
| **Оператор импорта для классов** |
| from db import User, Cart, Product |

Экземпляр объекта Flask создается с именем app. Чтобы использовать сеансы, нам нужно установить строку, которая будет действовать как секретный ключ с помощью app.secret\_key. Также мы создаем экземпляры пустых объектов классов User, Card и Product.

Основная структура веб-сайта приведена в Приложении 10. У него есть индекс, который действует как главная страница, с которой можно войти в систему, войти в систему, добавить и перейти в свою корзину, а также другие функции, такие как кнопка справки, которая перенаправляет на страницу Github серверного разработчика.

## Вспомогательные функции

* current\_user: Принимает идентификатор пользователя и возвращает пользовательский объект пользователя с этим идентификатором;
* current\_user\_info: Принимает имя пользователя if и использует current\_user с помощью метода to\_dict для возврата данных пользователя в виде словаря;
* signed\_in: Возвращает, не является ли текущий пользователь пользователем по умолчанию USER, что означает, что кто-то вошел в систему;
* is\_admin: Если идентификатор текущего пользователя совпадает с идентификатором пользователя ADMIN, он возвращает true;
* add\_to\_cart: Принимает идентификатор пользователя и продукта и добавляет товар в корзину пользователя.

## Глобальные переменные

В каждом шаблоне некоторые переменные используются в каждом шаблоне, использующем Jinja. Чтобы позволить flask распознавать каждую переменную в шаблонах, функция set\_global\_html\_variable\_values с помощью декоратора \*app.context\_processor устанавливает эти переменные. Он присваивает значения user\_id текущего пользователя, его информации и того, возвращают ли signed\_in и is\_admin значение true или false. Это влияет на интерфейс и используется здесь для использования определенных функций в зависимости от этих переменных.

## Index

(См. Приложения 11, 12)

Функция index просто отображает index.html файл, который использует переменную products (список продуктов в базе данных) и функцию add\_to\_cart, описанную ранее. На этой странице пользователь может зарегистрироваться / войти в систему, если он еще не участвует в сеансе. Если они вошли в систему, то на странице отображается возможность выхода из системы или (если пользователь ADMIN вошел в систему) можно добавлять продукты в базу данных.

## Sign up

(См. Приложения 13, 14)

Функция signup получает информацию с помощью метода POST для получения учетных данных пользователя. Он также проверяет, соответствуют ли записанные учетные данные существующему пользователю в базе данных. Если пользователь уже существует в базе данных, то переменная ошибки изменяется на true и используется в шаблоне signup.html . Это приведет к перезагрузке страницы с сообщением об ошибке, в котором пользователю будет предложено вместо этого войти в систему. Если нет, то пользователь добавляется в базу данных, и его идентификатор сохраняется в сеансе как user\_id, чтобы, наконец, перенаправить пользователя на страницу success.

## Sign in

(см. Приложения 15, 16)

Эта функция загружает страницу signin и, как и signup, использует метод POST для запроса учетных данных пользователя. Эта функция может обнаружить две ошибки. Если пользователь не зарегистрирован, он перезагрузит страницу и попросит его перейти на страницу signin. Если пользователь ввел неправильный пароль, страница перезагрузится и снова запросит его учетные данные с правильным паролем. В случае успеха он перенаправит пользователя на страницу success.

## Success

(См. Приложения 17, 18)

После успешного входа или регистрации страница \*success отображается только при использовании метода GET, поскольку запрос на регистрацию или вход отправляется на эту страницу, и чтобы эта страница оставалась видимой только после входа или подписываясь.

## Logout

(См. Приложения 19)

logout перенаправляет пользователя в index после извлечения ключа user\_id из словаря сеанса, что будет означать, что в данный момент нет пользователя, вошедшего в систему.

## Add product

(См. Приложения 20, 21)

Если пользователь ADMIN вошел в систему, он может добавлять продукты в базу данных, заполнив форму. Используя метод GET и FORM, страница отправляет это на сервер и сохраняется в базе данных. Кроме того, он определяет, находится ли добавленный продукт уже в базе данных, и отправляет эту информацию в виде логической переменной ошибки, как показано ранее.

## Cart

(См. Приложения 22, 23)

В корзине отображается каждый товар, который был добавлен в нее. Чтобы добавить продукт в корзину, каждый продукт, показанный в index, имеет кнопку для добавления его в корзину, которая использует функцию add\_to\_cart. Когда пользователь решает купить то, что находится в его корзине, он нажимает кнопку Buy и перенаправляется на страницу thankyou.

## Thank you

(См. Приложения 24, 25)

После нажатия кнопки Buy он отправляет эту информацию на страницу thankyou и, используя метод GET, отображает thankyou.html шаблон. Он также очищает корзину текущего пользователя в базе данных.

Заключение

Этот проект применил на практике множество методов для достижения поставленной цели. Объектно-ориентированное программирование, управление базами данных и использование шаблонов вместе с Jinja во многих отношениях упростили эту задачу. Наряду с отзывом всего, что здесь использовалось, было использовано и изучено множество процессов и функций, что также показывает, как много еще предстоит узнать и улучшить в этой области. Лучшее использование словаря сеанса было бы лучше для хранения корзины текущего пользователя. Можно было бы добиться лучшей интеграции интерфейса. И более простые продукты можно было бы использовать для добавления большего количества визуальных элементов, таких как изображения.

Создание бэкэнда было большой задачей, так как он управляет большим количеством информации и должен быть подключен к интерфейсу, что сложнее при работе отдельно с другим человеком. Это наряду с языковым барьером между обоими создателями сайта оставляет сайт для улучшения в слиянии интерфейса и бэкэнда. Но это опыт обучения командной работе в совместном проекте, который будет полезен для будущих проектов.

Список литературы

Fonticons. (б.д.). *Font Awesome*. Получено из Font Awesome: https://fontawesome.com

GeeksforGeeks. (06 12 2021 г.). *Web Development*. Получено из GeeksforGeeks: https://www.geeksforgeeks.org/web-development/

https://pythonbasics.org. (2021). *Flask HTTP methods, handle GET & POST requests*. Получено из Python Tutorial: https://pythonbasics.org/flask-http-methods/

JSON. (б.д.). *Введение в JSON*. Получено из JSON.org: https://www.json.org/json-ru.html

Microsoft. (27 10 2021 г.). *Flask Tutorial in Visual Studio Code*. Получено из Visual Studio Code: https://code.visualstudio.com/docs/python/tutorial-flask

Pallets. (2010). *Installation*. Получено из Flask Documentation (2.1.x): https://flask.palletsprojects.com/en/2.1.x/installation/

Python Software Foundation. (16 06 2022 г.). *9. Classes*. Получено из Python 3.10.5 documentation: https://docs.python.org/3/tutorial/classes.html

Сноски

1Часть веб-сайта, с которой пользователь взаимодействует напрямую. (GeeksforGeeks, 2021)

2Расшифровывается как HyperText Markup Language (язык разметки гипертекста). Он используется для разработки интерфейсной части веб-страниц с использованием языка разметки. Он действует как каркас для веб-сайта, поскольку используется для создания структуры веб-сайта. (GeeksforGeeks, 2021)

3Cascading Style Sheets (каскадные таблицы стилей) - это просто разработанный язык, предназначенный для упрощения процесса придания веб-страницам презентабельного вида. Он используется для оформления веб-сайта. (GeeksforGeeks, 2021)

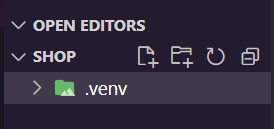
4Проект с открытым исходным кодом, который помогает включить в проект основной набор профессионально разработанных иконок, анимаций и различных элементов стиля. (Fonticons, s.f.)

5Это часть веб-сайта, которую пользователи не могут видеть и взаимодействовать с ней; она используется для хранения и упорядочивания данных. (GeeksforGeeks, 2021)

6JavaScript Object Notation (объектная нотация JavaScript) - это облегченный формат обмена данными, основанный на парах имя/значение и массивах. (JSON, s.f.)

7American standard code for information interchange — название таблицы, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.

Приложения



Приложение 1. Виртуальная среда задается в текстовом редакторе.



Приложение 2. Вывод базового приложения.

|  |
| --- |
| **Код для класса Database** |
| class Database:  def \_\_init\_\_(self, path: str) -> None:  self.path: Path = Path(path)  self.path.touch()  def write\_db(self, db: list) -> None:  with open(self.path, "w", encoding="utf-8") as f:  json.dump(db, f, indent="\t", ensure\_ascii=False)  def read\_db(self) -> list:  with open(self.path, encoding="utf-8") as f:  try:  db = json.load(f)  except json.decoder.JSONDecodeError:  db = []  self.write\_db(db)  return db |

Приложение 3. Код для класса Database.

|  |
| --- |
| **Код для класса Cart** |
| class Cart:  def \_\_init\_\_(self, cart: list = list()) -> None:  self.cart = cart  self.db = Database("database/carts.json")  if not self.db.read\_db():  USER\_cart = []  ADMIN\_cart = []  default\_db = [USER\_cart, ADMIN\_cart]  self.db.write\_db(default\_db)  def db\_view(self) -> list:  return self.db.read\_db()  def db\_commit(self, new\_db: list) -> None:  self.db.write\_db(new\_db)  def db\_add(self, id: int = -1) -> None:  db = self.db\_view()  if id == -1:  db.append(self.cart)  else:  db[id] = self.cart  self.db\_commit(db)  def parse\_by\_id(self, id: int) -> None:  db = self.db\_view()  if id in range(len(db)):  self.cart = db[id]  def add\_product(self, user\_id: int, product\_id: int):  self.parse\_by\_id(user\_id)  if product\_id not in self.cart:  self.cart.append(product\_id)  self.db\_add(user\_id)  def clear(self, user\_id: int):  self.cart.clear  self.parse\_by\_id(user\_id) |

Приложение 4. Код для класса Cart.

|  |
| --- |
| **carts.json** |
| [[], []] |

Приложение 5. Значение по умолчанию для базы данных Cart.

|  |
| --- |
| **Код для класса User** |
| class User:  def \_\_init\_\_(self, username: str = "user", password: str = "user") -> None:  self.username: str = username.upper()  self.password: int = int("".join([str(ord(c)) for c in password])) # hashing  self.cart = Cart()  self.db = Database("database/users.json")  if not self.db.read\_db():  USER = {"username": "USER", "password": [удалено]}  ADMIN = {"username": "ADMIN", "password": [удалено]}  default\_db = [USER, ADMIN]  self.db.write\_db(default\_db)  def db\_view(self) -> list:  return self.db.read\_db()  def db\_commit(self, new\_db: list) -> None:  self.db.write\_db(new\_db)  def db\_add(self) -> None:  db = self.db\_view()  if not self.in\_db():  db.append(self.to\_dict())  self.db\_commit(db)  self.cart.db\_add()  def to\_dict(self) -> dict:  data = {"username": self.username, "password": self.password}  return data  def parse(self, data: dict) -> None:  self.username = data["username"]  self.password = data["password"]  def parse\_by\_id(self, id: int) -> None:  db = self.db\_view()  if id in range(len(db)):  self.parse(db[id])  def in\_db(self) -> bool:  db = self.db\_view()  usernames = [user["username"] for user in db]  return self.username in usernames  def get\_id(self) -> int:  if self.in\_db():  db = self.db\_view()  user\_info = self.to\_dict()  try:  return db.index(user\_info)  except ValueError:  return -1  else:  return -1  def pwd\_check(self) -> bool:  db = self.db\_view()  id = self.get\_id()  if id != -1:  return self.password == db[id]["password"]  return False |

Приложение 6. Код для класса User.

|  |
| --- |
| **users.json** |
| [  {  "username": "USER",  "password": [удалено]  },  {  "username": "ADMIN",  "password": [удалено]  }  ] |

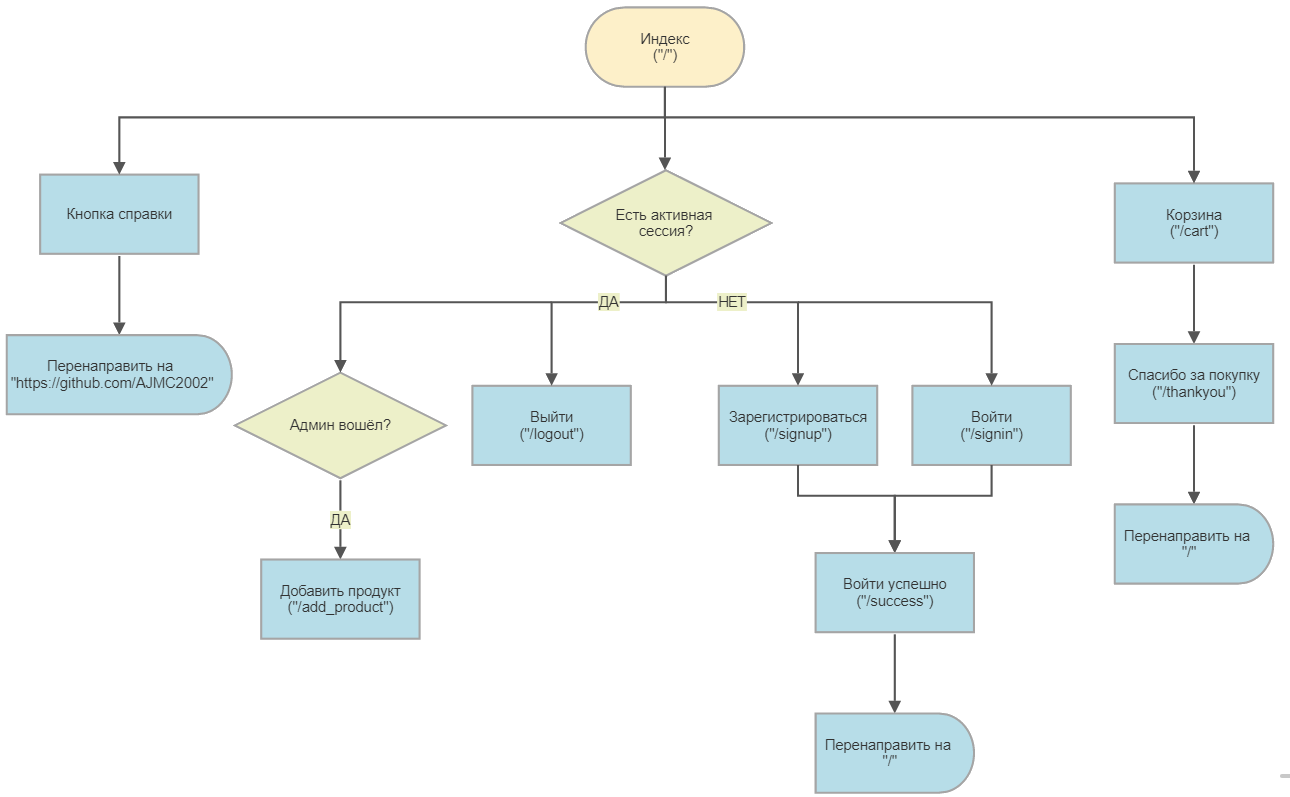
Приложение 7. Значение по умолчанию для базы данных User (пароли удалены для конфиденциальности).

|  |
| --- |
| **Код для класса Product** |
| class Product:  def \_\_init\_\_(  self,  category: str = "",  name: str = "",  mold: str = "",  color: str = "",  capacity: int = 0,  height: int = 0,  price: int = 0,  ) -> None:  self.category: str = category.capitalize()  self.name: str = name.upper()  self.mold: str = mold.upper()  self.color: str = color.capitalize()  self.capacity: int = int(capacity)  self.height: int = int(height)  self.price: int = int(price)  self.db = Database("database/products.json")  def db\_view(self) -> list:  return self.db.read\_db()  def db\_commit(self, new\_db: list) -> None:  self.db.write\_db(new\_db)  def db\_add(self) -> None:  db = self.db\_view()  if not self.in\_db():  db.append(self.to\_dict())  self.db\_commit(db)  def to\_dict(self) -> dict:  data = dict()  data["category"] = self.category  data["name"] = self.name  data["mold"] = self.mold  data["color"] = self.color  data["capacity"] = self.capacity  data["height"] = self.height  data["price"] = self.price  return data  def parse(self, data: dict) -> None:  self.category = data["category"]  self.name = data["name"]  self.mold = data["mold"]  self.color = data["color"]  self.capacity = data["capacity"]  self.height = data["height"]  self.price = data["price"]  def parse\_by\_id(self, id: int) -> None:  db = self.db\_view()  if id in range(len(db)):  self.parse(db[i])  def in\_db(self) -> bool:  db = self.db\_view()  names = [product["name"] for product in db]  return self.name in names  def get\_id(self) -> int:  if self.in\_db():  db = self.db\_view()  prod\_info = self.to\_dict()  try:  return db.index(prod\_info)  except ValueError:  return -1  else:  return -1 |

Приложение 8. Код для класса Product.

|  |
| --- |
| **products.json** |
| [] |

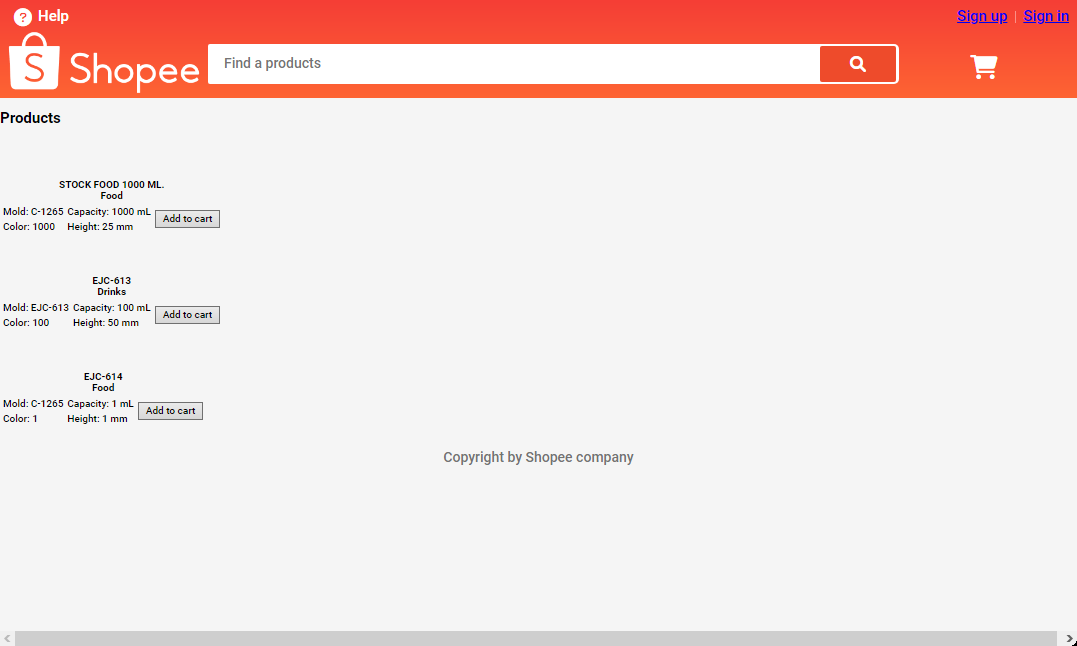
Приложение 9. Значение по умолчанию для базы данных Product.



Приложение 10. Схема веб-приложения.

|  |
| --- |
| **Код для функции index** |
| @app.route("/")  def index():  global base\_product  return render\_template(  "index.html",  products=base\_product.db\_view(),  add\_to\_cart=add\_to\_cart  ) |

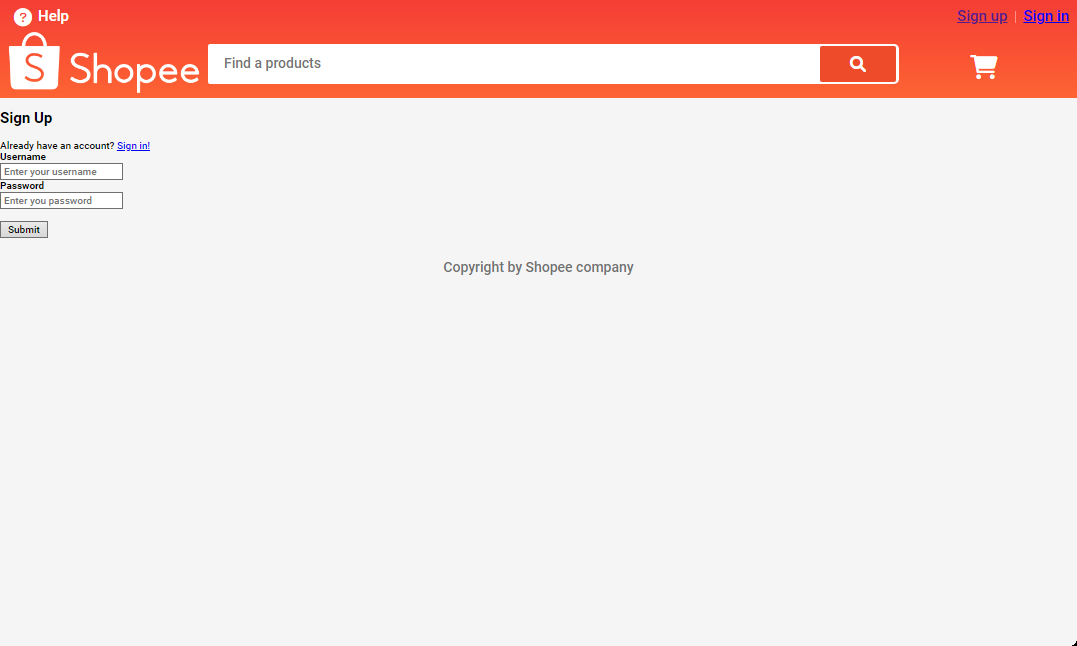
Приложение 11. Код для функции index.



Приложение 12. Вывод для "/".

|  |
| --- |
| **Код для функции signup** |
| @app.route("/signup", methods=["GET", "POST"])  def signup():  error = None # user input already in db  if request.method == "POST":  username = request.form.get("usr")  password = request.form.get("pwd")  user = User(username, password)  error = user.in\_db()  if not error:  user.db\_add()  session["user\_id"] = user.get\_id()  return redirect(url\_for("success"))  return render\_template("signup.html", error=error) |

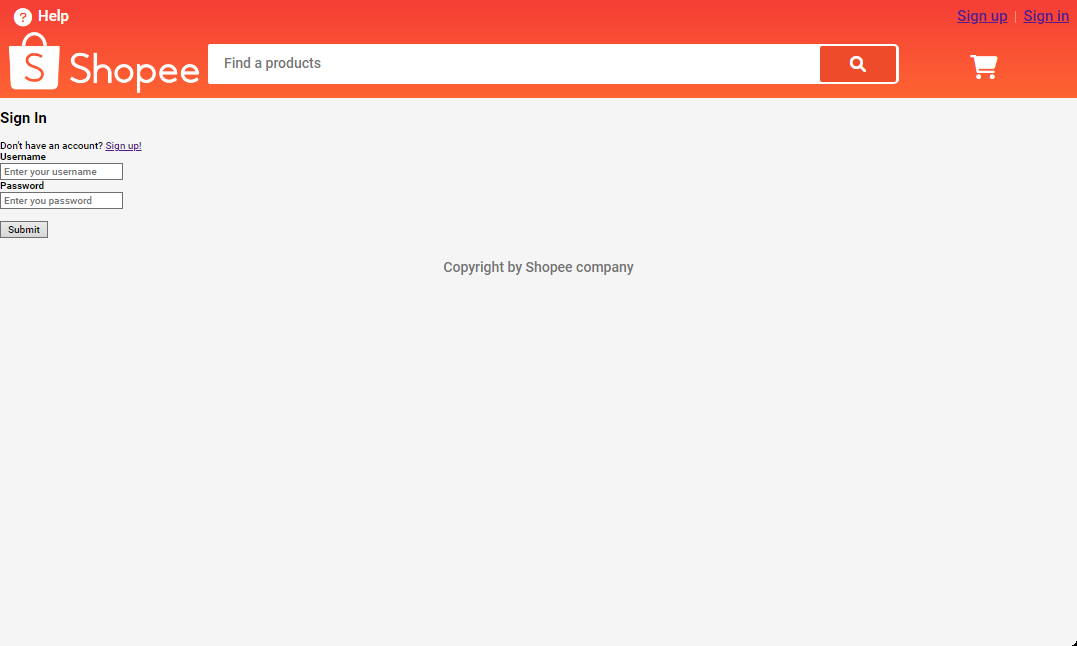
Приложение 13. Код для функции signup.



Приложение 14. Вывод для “/signup”.

|  |
| --- |
| **Код для функции signin** |
| @app.route("/signin", methods=["GET", "POST"])  def signin():  error1 = False # user is not registered  error2 = False # wrong password  if request.method == "POST":  username = request.form.get("usr")  password = request.form.get("pwd")  user = User(username, password)  error1 = not user.in\_db()  error2 = not user.pwd\_check()  if not error1 and not error2:  session["user\_id"] = user.get\_id()  return redirect(url\_for("success"))  return render\_template("signin.html", error1=error1, error2=error2) |

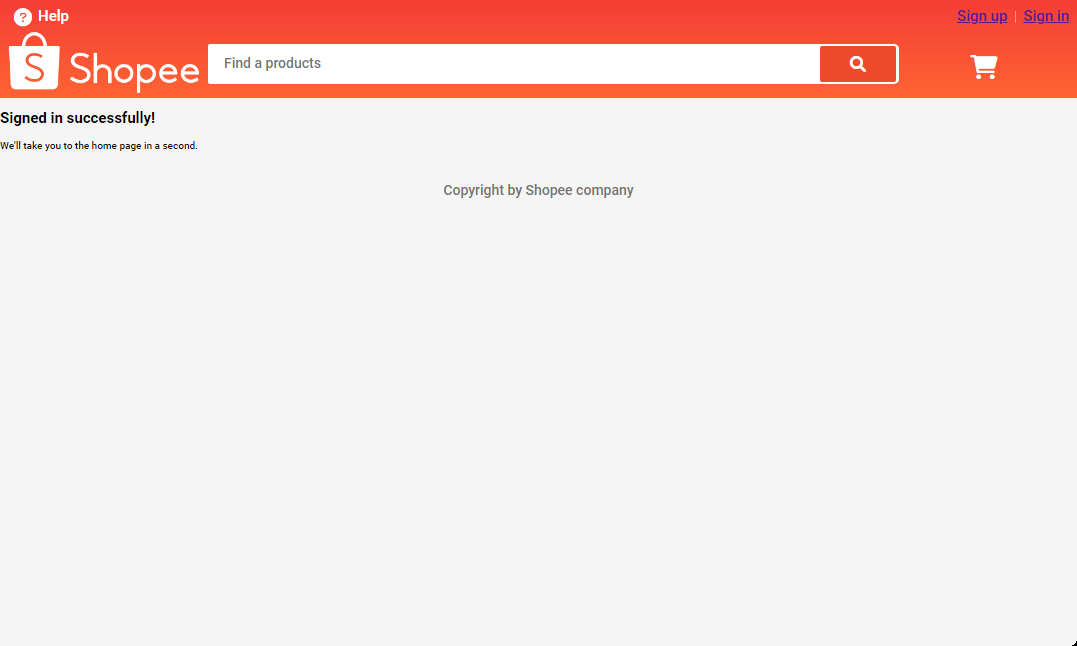
Приложение 15. Код для функции signin.



Приложение 16. Вывод для “/signin”.

|  |
| --- |
| **Код для функции success** |
| @app.route("/success", methods=["GET"])  def success():  if request.method == "GET":  return render\_template("success.html") |

Приложение 17. Код для функции success.



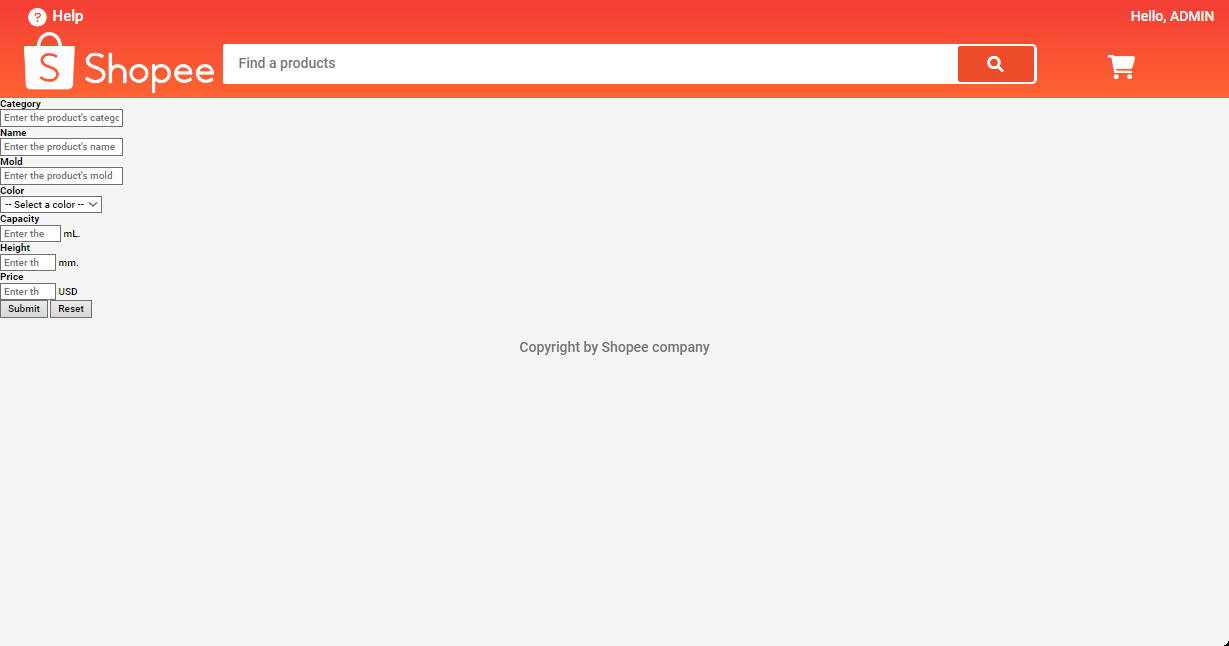
Приложение 18. Вывод для “/success”.

|  |
| --- |
| **Код для функции logout** |
| @app.route("/logout")  def logout():  session.pop("user\_id", None)  return redirect(url\_for("index")) |

Приложение 19. Код для функции logout.

|  |
| --- |
| **Код для функции add\_product** |
| @app.route("/add\_product", methods=["GET", "POST"])  def add\_product():  error = None  if request.method == "POST":  ctg = request.form.get("ctg")  name = request.form.get("name")  mold = request.form.get("mold")  clr = request.form.get("clr")  cap = request.form.get("cap")  height = request.form.get("height")  price = request.form.get("price")  product = Product(ctg, name, mold, clr, cap, height, price)  error = not product.db\_add()  return render\_template("add\_product.html", error=error) |

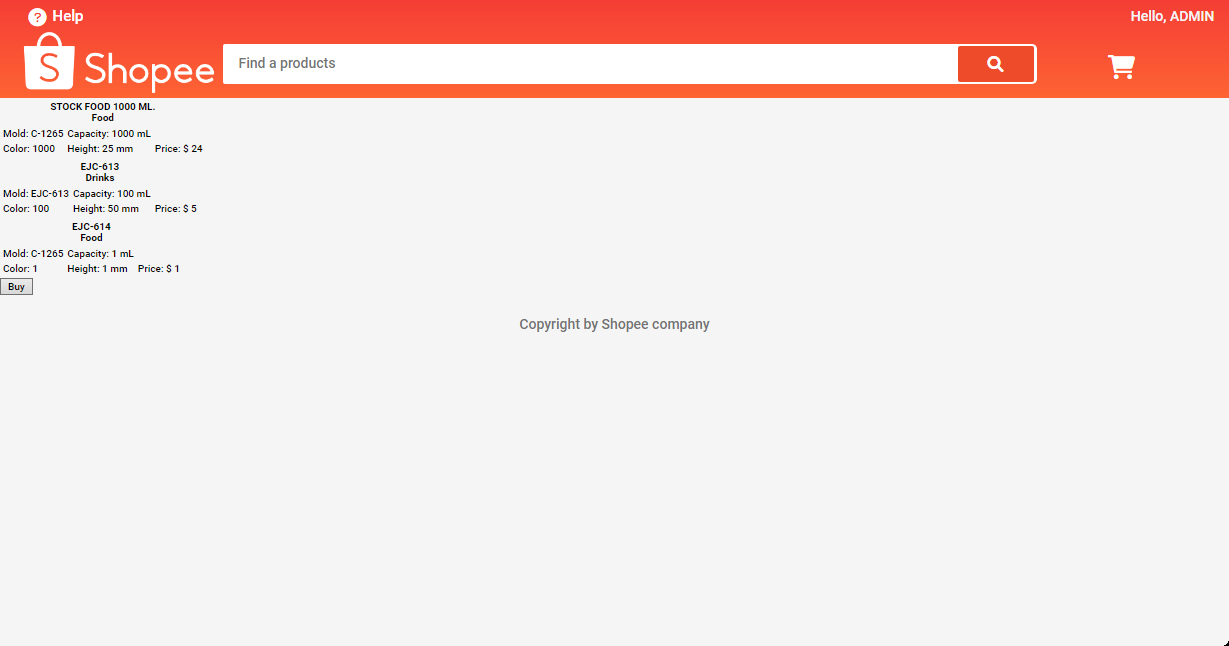
Приложение 20. Код для функции add\_product.



Приложение 21. Вывод для “/add\_product”.

|  |
| --- |
| **Код для функции cart** |
| @app.route("/cart")  def cart():  global base\_cart, base\_product  return render\_template(  "cart.html",  carts=base\_cart.db\_view(),  products=base\_product.db\_view()  ) |

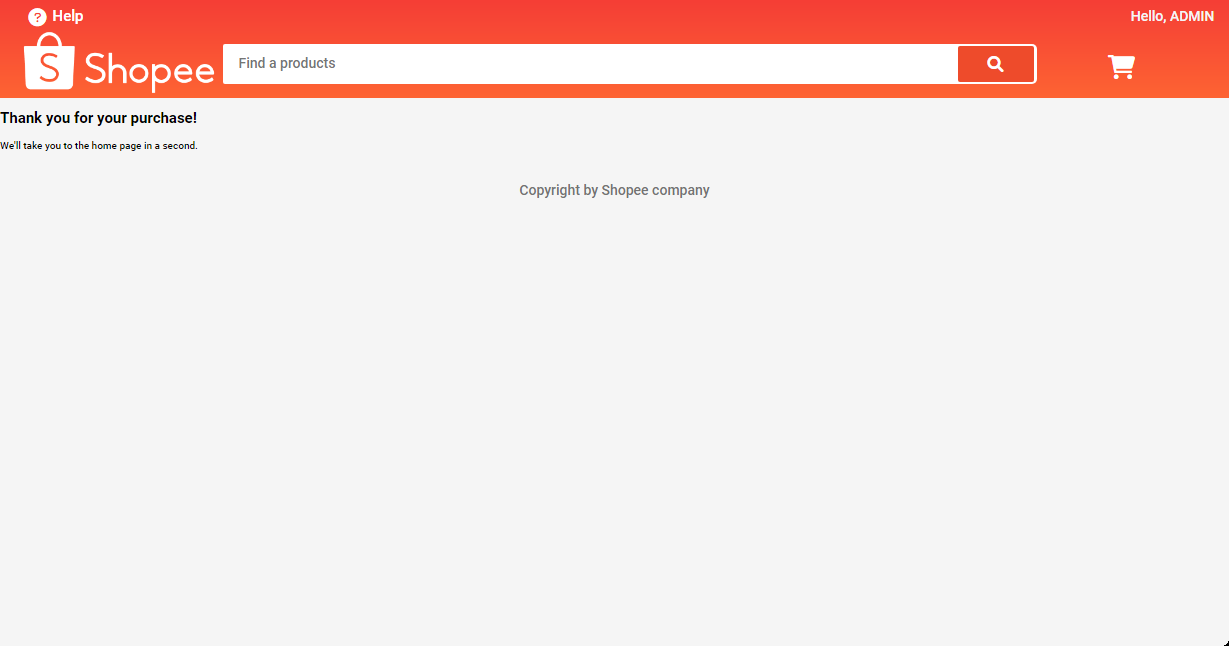
Приложение 22. Код для функции cart.



Приложение 23. Вывод для “/cart”.

|  |
| --- |
| **Код для функции thankyou** |
| @app.route("/thankyou", methods=["GET"])  def thankyou():  if "user\_id" in session:  Cart().clear(session["user\_id"])  if request.method == "GET":  return render\_template("thankyou.html") |

Приложение 24. Код для функции thankyou.



Приложение 25. Вывод для “/thankyou”.