Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчеты по лабораторным работам 0-4

Дисциплина: «Технологии сетевого программирования»

Тема «Книжный интернет-магазин»

Выполнила: Никонорова Т.А.

Группа: 6302-010302D

Самара, 2025

**Концепция приложения и сценарии использования**

* Концепция:

Книжный интернет-магазин, позволяющий пользователям регистрироваться, просматривать каталог книг, добавлять их в корзину и оформлять заказы. Также должна быть возможность просматривать историю покупок и обработки заказов.

* Сценарии использования:

Регистрация и вход: Пользователь заходит на сайт. Если у него нет аккаунта, он регистрируется (вводит email, пароль, имя). Если аккаунт есть, он входит, вводя email и пароль.

Просмотр каталога книг: Пользователь переходит на страницу каталога. Ознакамливается с ассортиментом, видит названия, авторов, жанры. Выбирает.

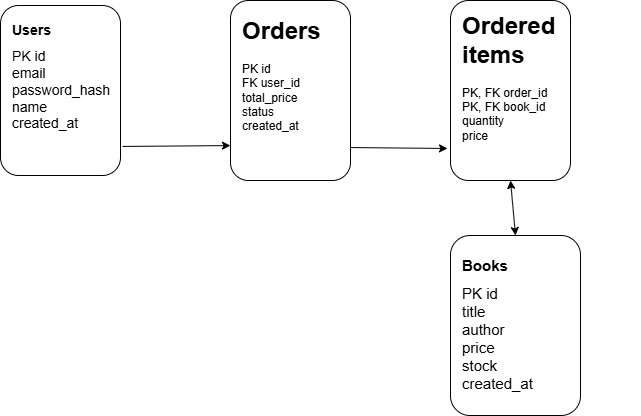
Добавление книги в корзину и покупка: Пользователь выбирает книги и нажимает «Добавить в корзину». Получает уведомление об успешной покупке.

Просмотр истории покупок: Пользователь заходит в личный кабинет. Видит список своих заказов (дата, книги, статус заказа).

Обработка ошибок: Некорректный пароль/email при входе. Ошибка при оформлении заказа (недоступность книги, проблемы с оплатой).

**Схема БД**

* Клиент (Frontend): веб-интерфейс, через который пользователи регистрируются, авторизуются, просматривают каталог книг и совершают покупки.
* Сервер (Backend): обрабатывает запросы от клиента, взаимодействует с базой данных, выполняет аутентификацию.
* База данных: хранит данные о пользователях, книгах и заказах.



**Схема API**

Аутентификация:

* POST /register

Запрос: { "email": "[user@example.com](mailto:user@example.com)", "password": "securepassword", "name": "User Name" }

Ответ: { "message": "User registered successfully" }

* POST /login

Запрос: { "email": "[user@example.com](mailto:user@example.com)", "password": "securepassword" } Ответ: { "token": "jwt\_token\_here" }

Работа с книгами

* GET /books — получить список книг

Ответ: [ { "id": 1, "title": "Book 1", "author": "Author A", "price": 19.99 }, { "id": 2, "title": "Book 2", "author": "Author B", "price": 24.99 } ]

* GET /books/{id} — получить информацию о конкретной книге

Ответ: { "id": 1, "title": "Book 1", "author": "Author A", "price": 19.99, "description": "Book description here" }

Работа с заказами

* POST /orders — создать заказ

Запрос: { "user\_id": 1, "items": [ { "book\_id": 1, "quantity": 2 }, { "book\_id": 3, "quantity": 1 } ] }

Ответ: { "order\_id": 123, "message": "Order created successfully" }

* GET /orders — получить историю заказов пользователя

Ответ: [ { "id": 123, "total\_price": 59.97, "status": "paid", "created\_at": "2025-02-15" } ]

**Список используемых технологий**

* Frontend (шаблоны страниц и стили):

Bootstrap — CSS-фреймворк для стилизации и адаптивной верстки

* Backend (архитектура и серверная логика)

Flask — микрофреймворк для разработки веб-приложений на Python

Jinja2 — встроенный в Flask шаблонизатор для рендеринга HTML

* База данных и управление данными

MySQL — реляционная база данных

MySQL Workbench — инструмент для проектирования и администрирования базы данных Регистрация и авторизация

OAuth — безопасная аутентификация через сторонние сервисы Flask-Login — управление сессиями пользователей в Flask

**Описание базы данных**

Тип ORM: SQLAlchemy

Тип БД: MySQL

Приложение: Flask

Назначение: База данных для хранения информации о продуктах, категориях, пользователях, заказах, отзывах, изображениях.

* Таблица main\_category\_product (модель `MainCategoryProduct`)

Содержит данные о главных категориях товаров.

Поля:

id — уникальный идентификатор категории (первичный ключ).

name — название главной категории (уникальное значение).

Связи:

Один ко многим: одна главная категория содержит множество подкатегорий (`CategoryProduct`).

* Таблица category\_product (модель `CategoryProduct`)

Содержит данные о подкатегориях, которые принадлежат к определённой главной категории.

Поля:

id — уникальный идентификатор подкатегории.

name — название подкатегории.

main\_parameters\_id — необязательное поле для главных параметров.

main\_category\_product\_id — внешний ключ на таблицу main\_category\_product.

Связи:

Один ко многим: одна подкатегория может содержать несколько продуктов (`Product`).

Один ко многим: подкатегория может иметь несколько параметров (`CategoryProductMainParameters`).

* Таблица category\_product\_main\_parameters (модель `CategoryProductMainParameters`)

Содержит параметры, которые могут быть привязаны к конкретной подкатегории.

Поля:

id — идентификатор параметра.

name — название параметра.

category\_product\_id — внешний ключ на таблицу category\_product.

Связи:

Один ко многим: одна подкатегория может иметь несколько параметров.

* Таблица product (модель `Product`)

Хранит информацию о товарах (книгах).

Поля:

id — уникальный идентификатор товара.

name — название товара.

description — описание товара.

cost — цена.

rating — оценка.

category\_product\_id — внешний ключ на таблицу category\_product.

parameters — дополнительная информация о товаре в формате JSON.

Связи:

Один ко многим: один товар может иметь несколько изображений (`ImagesProduct`).

Один ко многим: один товар может иметь несколько отзывов (`Feedback`).

Один ко многим: один товар может быть заказан в разных заказах (`Order`).

* Таблица images\_product (модель `ImagesProduct`)

Содержит изображения, связанные с конкретным товаром.

Поля:

id — идентификатор изображения.

id\_product — внешний ключ на таблицу product.

file — путь к изображению.

priority — приоритет отображения.

Связи:

Один товар может иметь несколько изображений.

* Таблица users (модель `User`)

Содержит информацию о пользователях системы.

Поля:

id — идентификатор пользователя.

login — уникальный логин.

name — имя пользователя.

city — город проживания.

password — пароль (зашифрованный).

Связи:

Один ко многим: один пользователь может сделать много заказов (`Order`).

Один ко многим: один пользователь может оставить много отзывов (`Feedback`).

* Таблица order (модель `Order`)

Содержит информацию о заказах пользователя.

Поля:

id — идентификатор заказа.

user\_id — внешний ключ на таблицу users.

product\_id — внешний ключ на таблицу product.

Связи:

Один заказ — один пользователь и один товар.

* Таблица feedbacks (модель `Feedback`)

Хранит отзывы пользователей о товарах.

Поля:

id — идентификатор отзыва.

user\_id — внешний ключ на таблицу users.

product\_id — внешний ключ на таблицу product.

text — текст отзыва.

rating — оценка товара.

Связи:

Один пользователь может оставить несколько отзывов.

Один товар может иметь несколько отзывов.

**Скрипты для демонстрации работы с данными**

* Пример добавления товара в базу данных:

product = Product(

name='Методы компьютерной оптики',

description='Под редакцией В.А. Сойфера',

cost=1000.00,

category\_product\_id=1,

parameters={

'BibTex': {

'priority': 0,

'parameters': {},

'main\_parameters': {

'GS\_ID': 'волков2000методы',

'title': 'Методы компьютерной оптики',

'author': 'Волков, Алексей В. и др.',

'year': 2000,

'publisher': 'Физико-математическая литература',

}

}

},

rating=4

)

session.add(product)

session.commit()

* Пример получения всех категорий:

main\_categorys = MainCategoryProduct.query.all()

for main in main\_categorys:

print(main.name)

for sub in main.categorys\_product:

print(' ', sub.name)

* Пример запроса категории по ID и получение её продуктов:

category = CategoryProduct.query.filter(CategoryProduct.id == category\_id).first()

if category:

for product in category.products:

print(product.name, product.cost)

* Пример добавления пользователя:

new\_user = User(login='ivan123', name='Иван', city='Москва', password='1234')

session.add(new\_user)

session.commit()

* Пример добавления заказа:

order = Order(user\_id=new\_user.id, product\_id=product.id)

session.add(order)

session.commit(

**Структура API**

API реализован на базе Flask с использованием SQLAlchemy для работы с базой данных. Взаимодействие осуществляется через маршруты (routes), определенные с помощью декоратора @blueprint.route().

Модели, с которыми работает API:

User — пользователь (включает логин, пароль, имя, город, список заказов и отзывов);

Order — заказ (пользователь, продукт);

Product — продукт (название, цена, описание и т.д.);

Feedback — отзыв (текст, оценка, продукт, пользователь);

MainCategoryProduct и CategoryProduct — категории товаров.

**CRUD-операции**

Для каждой модели реализованы базовые операции:

* Продукты (Product)

Create — добавление через скрипты или админку

Read — просмотр списка и отдельного продукта через маршруты

Update/Delete — не реализованы в интерфейсе пользователя, но возможны через SQLAlchemy

* Отзывы (Feedback)

Create — через POST-запрос формы отзыва

Read — отзывы отображаются на страницах продуктов

Update/Delete — не реализованы

* Заказы (Order)

Create — создаются через интерфейс пользователя

Read — просматриваются в личном кабинете

Update/Delete — не реализованы

* Пользователи (User)

Create — регистрация через форму

Read — личный кабинет, отображение имени

Update/Delete — не реализованы

**Описание аутентификации**

Аутентификация реализована с использованием Flask-Login.

Пользователь проходит вход через форму (`/login`), при успешной проверке вызывается login\_user(user).

Выход из аккаунта через /logout.

Зарегистрированные пользователи могут оставлять отзывы, делать заказы и видеть персонализированный интерфейс.

Модель User наследует UserMixin, что обеспечивает работу сессий и проверку current\_user.is\_authenticated.

**Примеры запросов и ответов**

* Пример 1. Регистрация пользователя

POST /register

Form data:

login: testuser

name: Тест

city: Москва

password: 1234

Ответ:

302 Found (редирект на страницу входа)

* Пример 2. Вход пользователя

POST /login

Form data:

login: testuser

password: 1234

Ответ:

302 Found (редирект на главную страницу)

* Получение товаров подкатегории

GET /category/2/

Ответ:

200 OK

(HTML-страница со списком продуктов)

* Пример 4. Отправка отзыва

POST /feedback

Form data:

product\_id: 3

rating: 4

text: Хорошая книга

Ответ:

200 OK — если пользователь авторизован

403 Forbidden — если пользователь не вошел в систему

* Пример 5. Выход пользователя

GET /logout

Ответ:

302 Found (редирект на главную)

1.5 Статусы ответов

200 OK — успешный запрос (например, категория найдена)

302 Found — успешная форма (регистрация/вход), редирект

401 Unauthorized — при неправильных данных входа

403 Forbidden — попытка действий без авторизации

500 Internal Server Error — при ошибке на сервере (например, сломанный маршрут)

Использование Postman/curl

Примеры запросов были проверены в Postman. Через него производились:

Тестирование входа/регистрации

Проверка отображения категорий и продуктов

Отправка отзывов

**Описание модели пользователя и JWT-аутентификации**

Дополнительно реализована аутентификация с помощью JWT:

При успешной авторизации создается токен (срок действия — ограничен).

Токен передается в заголовке Authorization: Bearer <token> при последующих запросах.

При каждом защищенном запросе токен проверяется middleware.

**Эндпоинты, связанные с авторизацией**

* Регистрация пользователя

POST /register

Body (form-data):

login: testuser

name: Тест

city: Москва

password: 1234

Ответ:

201 Created

{

"message": "Пользователь успешно зарегистрирован"

}

* Авторизация (вход) и получение токена

POST /login

Body (form-data):

login: testuser

passord: 1234

Ответ:

200 OK

{

"token": "<JWT-токен>"

}

Защищенный эндпоинт (пример)

GET /user/profile

Headers:

Authorization: Bearer <JWT-токен>

Ответ:

200 OK

{

"login": "testuser",

"name": "Тест",

"city": "Москва"

}

Если токен отсутствует или недействителен:

Ответ:

401 Unauthorized

{

"error": "Токен недействителен или отсутствует"

}

* Изменение пароля

POST /change-password

Headers:

Authorization: Bearer <JWT-токен>

Body (form-data):

old\_password: 1234

new\_password: 5678

Ответ:

200 OK

{

"message": "Пароль успешно изменён"

}

* Тестирование API

Инструменты: Postman и curl

Проверены основные сценарии:

Успешная регистрация и вход

Получение токена и доступ к защищенным маршрутам

Защита маршрутов без токена

Изменение пароля при авторизации

Отправка невалидных токенов (получен 401)

**Примеры curl-запросов**

* Регистрация

curl -X POST http://localhost:5000/register \

-d "login=testuser&password=1234&name=Тест&city=Москва"

* Вход

curl -X POST http://localhost:5000/login \

-d "login=testuser&password=1234"

Доступ к профилю

curl -H "Authorization: Bearer <JWT-токен>" \

http://localhost:5000/user/profile

* Статусы ответов

200 OK — успешный запрос

201 Created — успешная регистрация

400 Bad Request — неверные данные

401 Unauthorized — отсутствует или невалидный токен

403 Forbidden — доступ запрещён

500 Internal Server Error — ошибка сервера

**Структура интерфейса и логика**

* Основные страницы:

1. Главная страница

Список всех категорий и подкатегорий

Отображается вне зависимости от авторизации

2. Страница входа

HTML-форма для логина и пароля

При успешной авторизации пользователь получает токен

3. Страница регистрации

HTML-форма с полями логин, имя, город, пароль

Создается новый пользователь, токен не выдается автоматически

4. Страница просмотра подкатегории

URL: /category/<int:category\_id>/

Запрашиваются все товары подкатегории и отображаются с картинками и описанием

Пример передачи параметра:

@blueprint.route('/category/<int:category\_id>/')

def show\_category(category\_id):

...

return render\_template('subcategory\_products.html', ...)

5. Защищенные страницы (например, личный кабинет)

Доступны только при наличии JWT-токена в сессии

При его отсутствии — редирект на страницу входа

Пример кода рендеринга

@blueprint.route('/category/<int:category\_id>/')

def show\_category(category\_id):

categorys = get\_categorys()

category = CategoryProduct.query.\

filter(CategoryProduct.id == category\_id).first()

if not category:

return render\_template('subcategory\_products.html', categorys=categorys, products=None, user=current\_user)

products\_info = []

for product in category.products:

products\_info.append({

'id': product.id,

'product\_name': product.name,

'rating': product.rating,

'cost': product.cost,

'description': product.description,

'images': product.get\_sorted\_path\_images(config.get('PATH\_IMAGES')),

})

return render\_template('subcategory\_products.html',

categorys=categorys,

products=products\_info,

category=category,

main\_category=category.main\_category\_product,

user=current\_user,

)

Данные передаются в HTML-шаблон, где используются переменные Jinja2 (`{{ user.name }}`, {{ products }}, {{ category.name }} и т.д.).

Аутентификация

После входа пользователю выдается JWT-токен

Токен хранится в session или передается в заголовке (в зависимости от реализации)

Внутри сервер проверяет наличие токена перед доступом к защищенным маршрутам

Пример использования current\_user:

return render\_template('subcategory\_products.html', user=current\_user)

Примеры передачи параметров и динамического контента

Категории: main\_category.categorys\_product

Продукты: category.products

Отзывы: product.feedbacks

Jinja2-шаблон подставляет эти данные на стороне сервера до отправки клиенту.

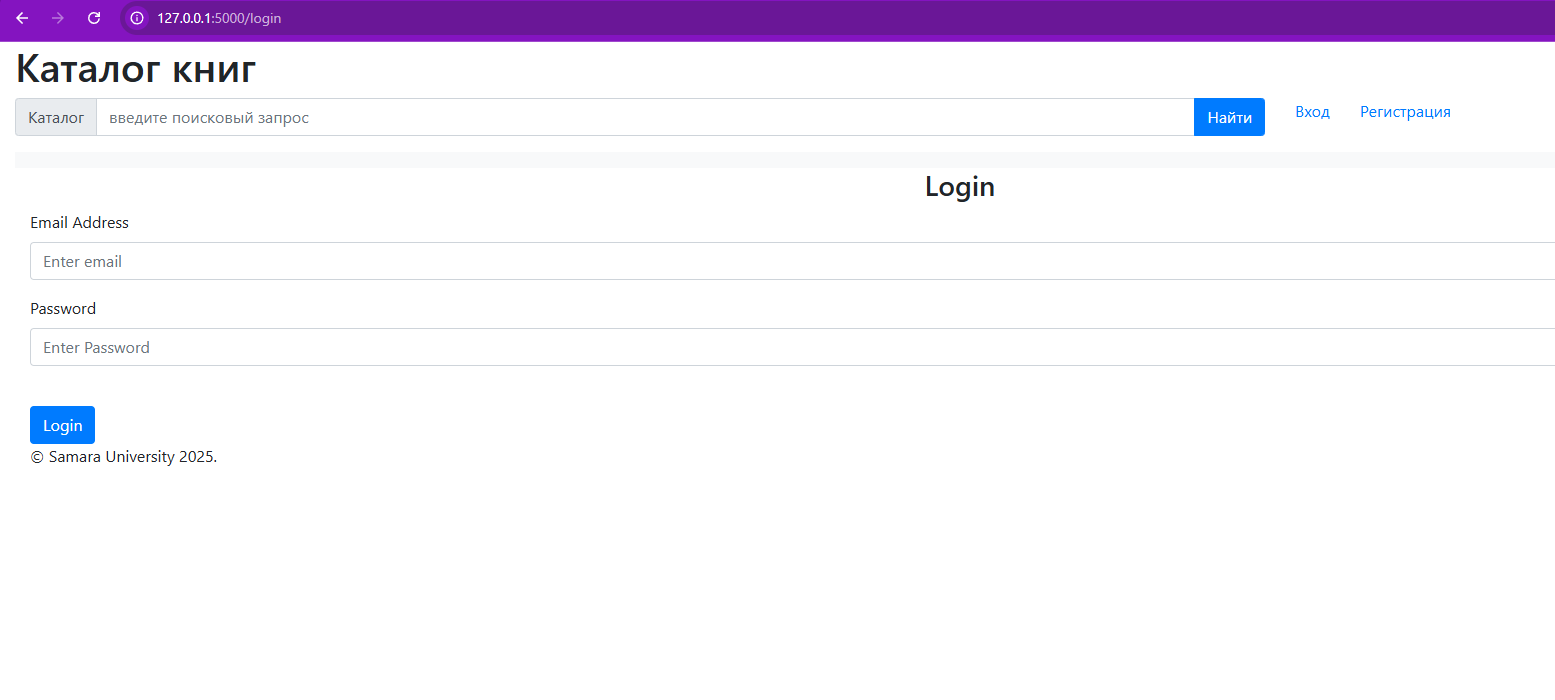


Рисунок 1 – Страница входа

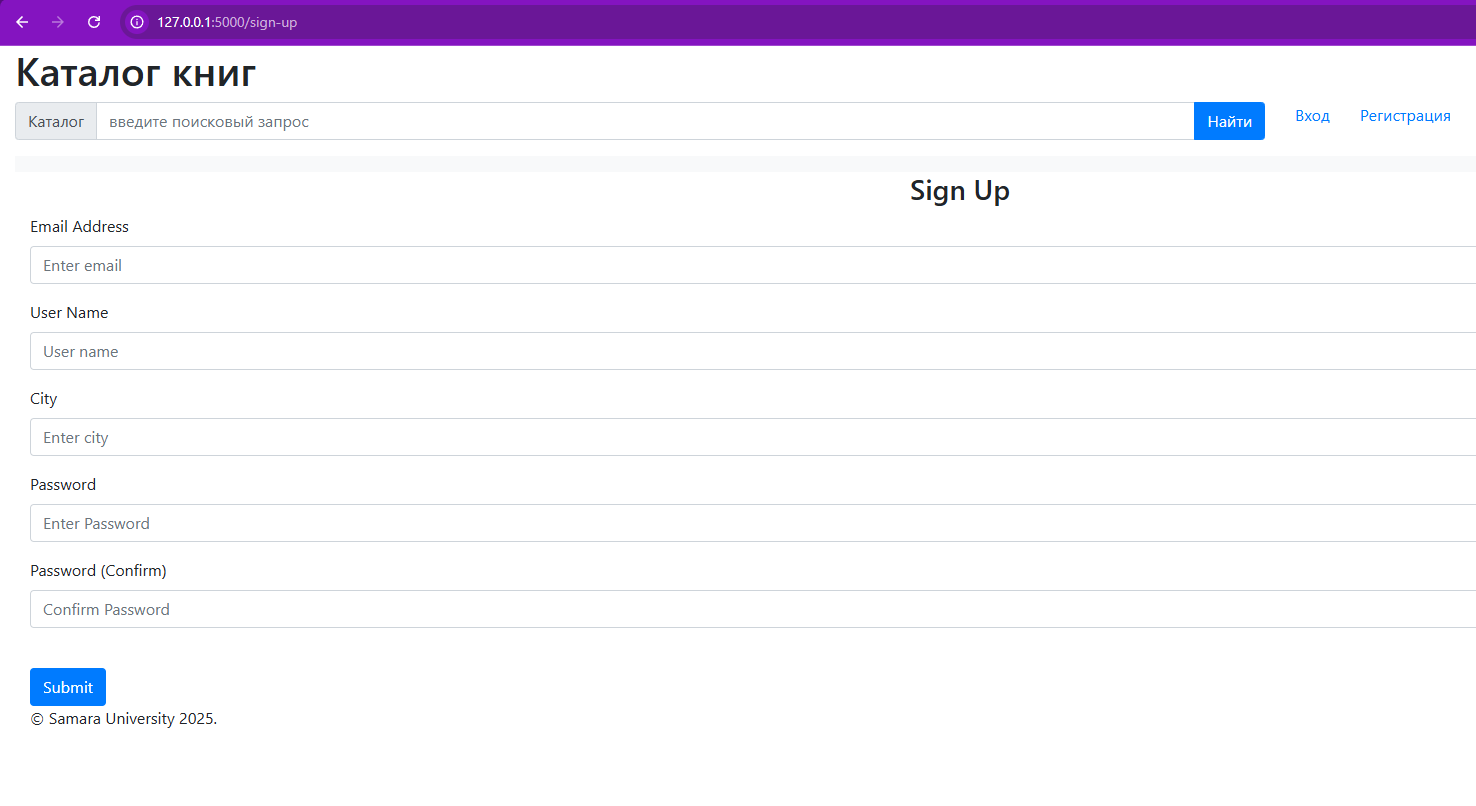


Рисунок 2 – Страница регистрации

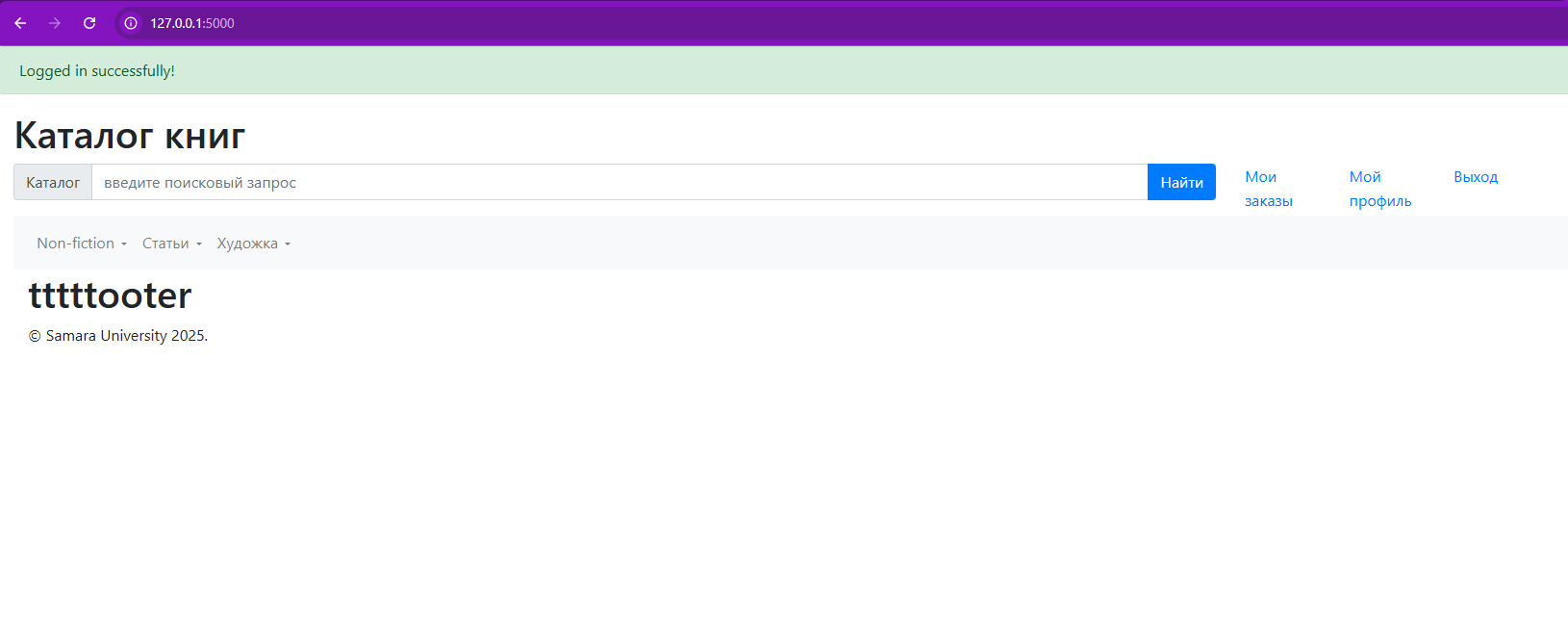


Рисунок 3 – Главная страница

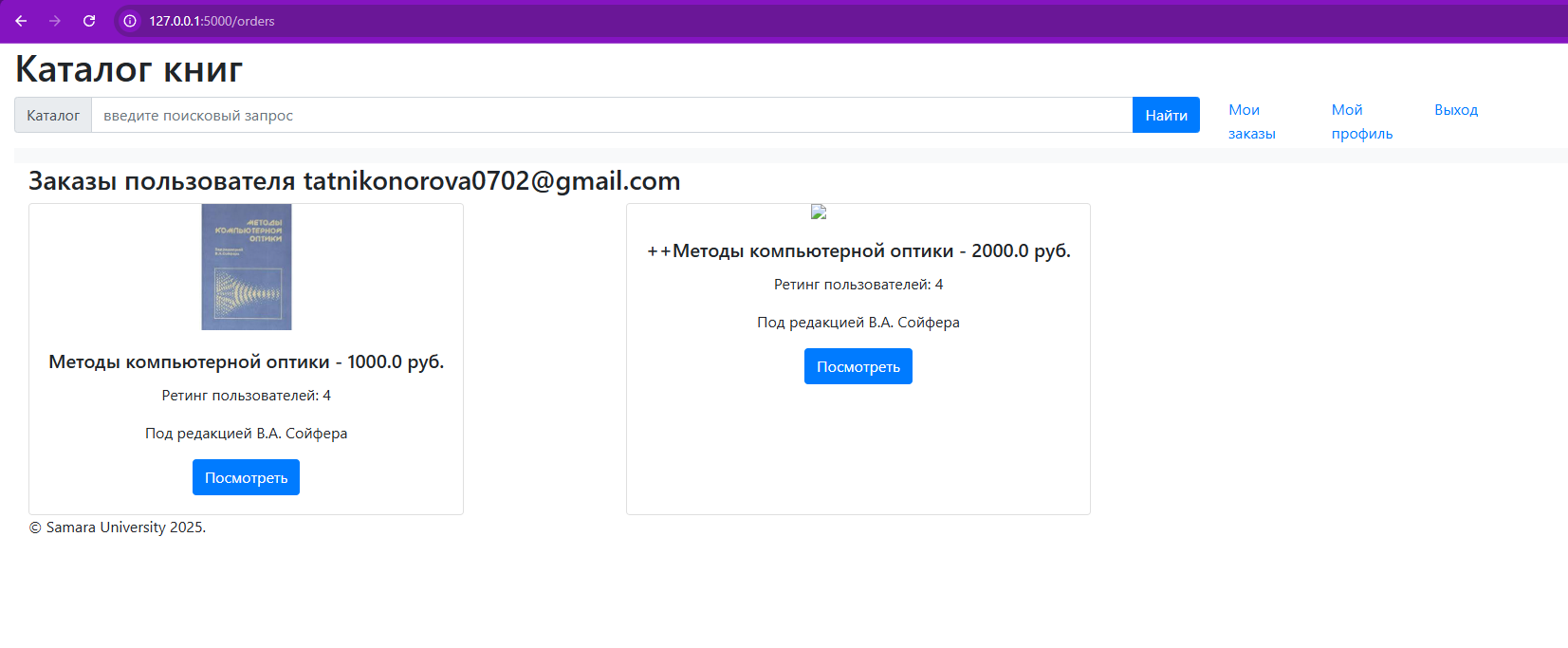


Рисунок 4 – Страница с заказами

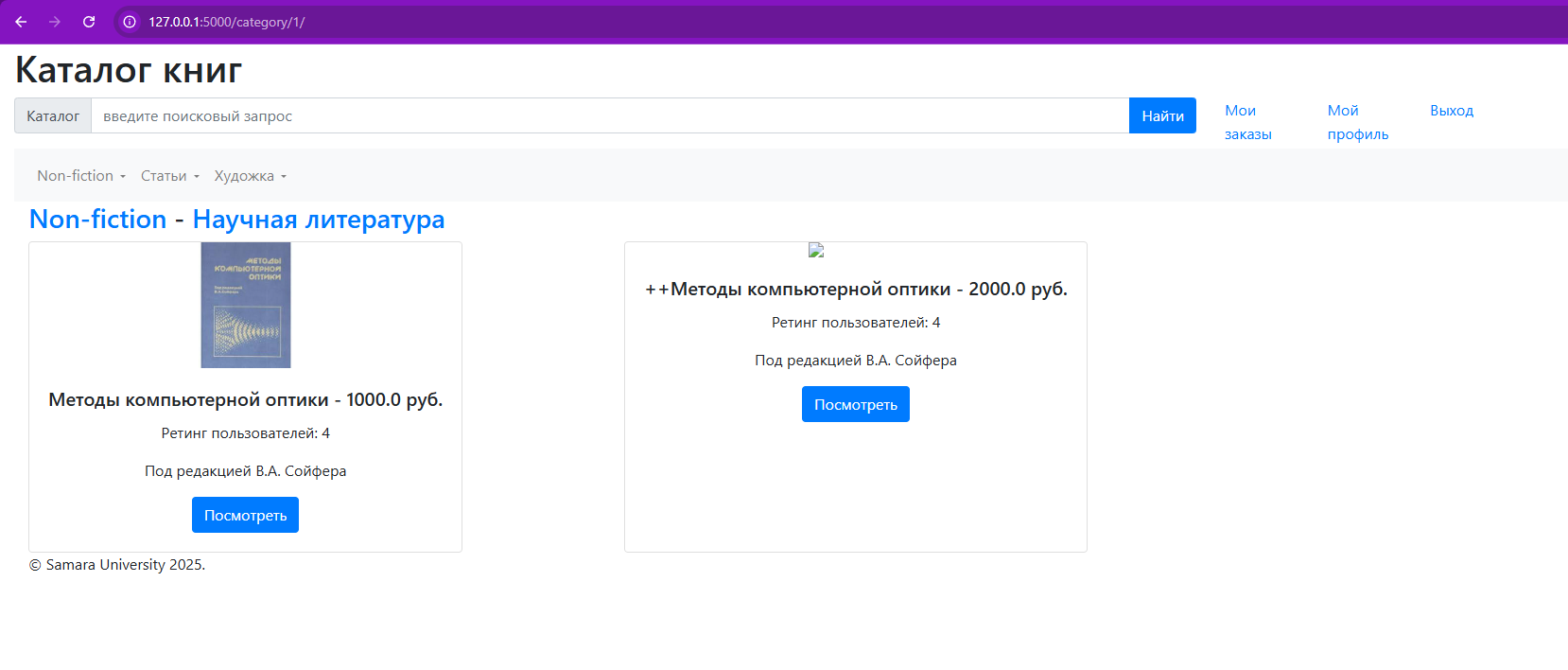


Рисунок 5 – Каталог

Ссылка на гит-репозиторий с проектом: https://github.com/tatnikonorova/TSP