**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Информационная безопасность баз данных»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

«Реализация сервиса для взаимодействия с разработанной базой данных»

**Выполнили:**

Бардышев Артём Антонович,

студент группы N3346

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Салихов Максим Русланович,

преподаватель, ФБИТ

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Содержание

[Введение 3](#_Toc210326334)

[1 Ход выполнения 4](#_Toc210326335)

[1.1 Мониторинг (логирование операций) 4](#_Toc210326336)

[1.2 Шифрование секретных данных 5](#_Toc210326337)

[1.3 Разграничение доступа 7](#_Toc210326338)

[Заключение 9](#_Toc210326339)

[Список использованных источников 10](#_Toc210326340)

Введение

Цель работы – Получение навыков использования серверных языков программирования, фреймворков по работе с БД, ORM-систем.

# Ход выполнения

## Стек и подготовка (Windows/PowerShell)

mkdir lr4\_apple; cd lr4\_apple

python -m venv .venv

. .\.venv\Scripts\Activate.ps1

pip install flask sqlalchemy psycopg2-binary flask-login python-dotenv passlib[bcrypt]

pip freeze > requirements.txt

Создаем **.env** в корне:

DB\_USER=root

DB\_PASS=root

DB\_NAME=apple\_store

DB\_HOST=localhost

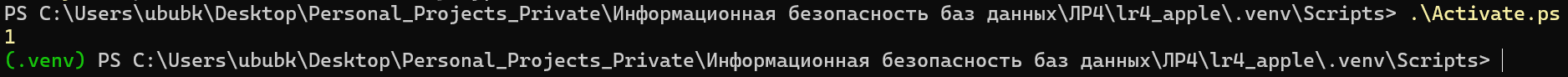
SECRET\_KEY=supersecret

## Создаем виртуальное окружение

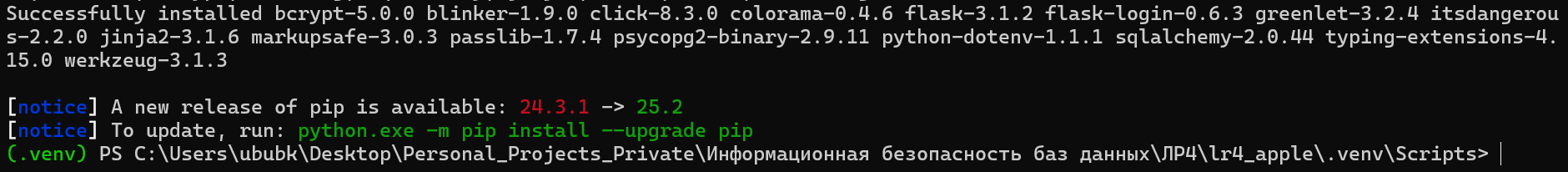
if (-not (Test-Path .venv)) { python -m venv .venv }

После выполнения появится папка .venv внутри проекта.

## Активируем окружение

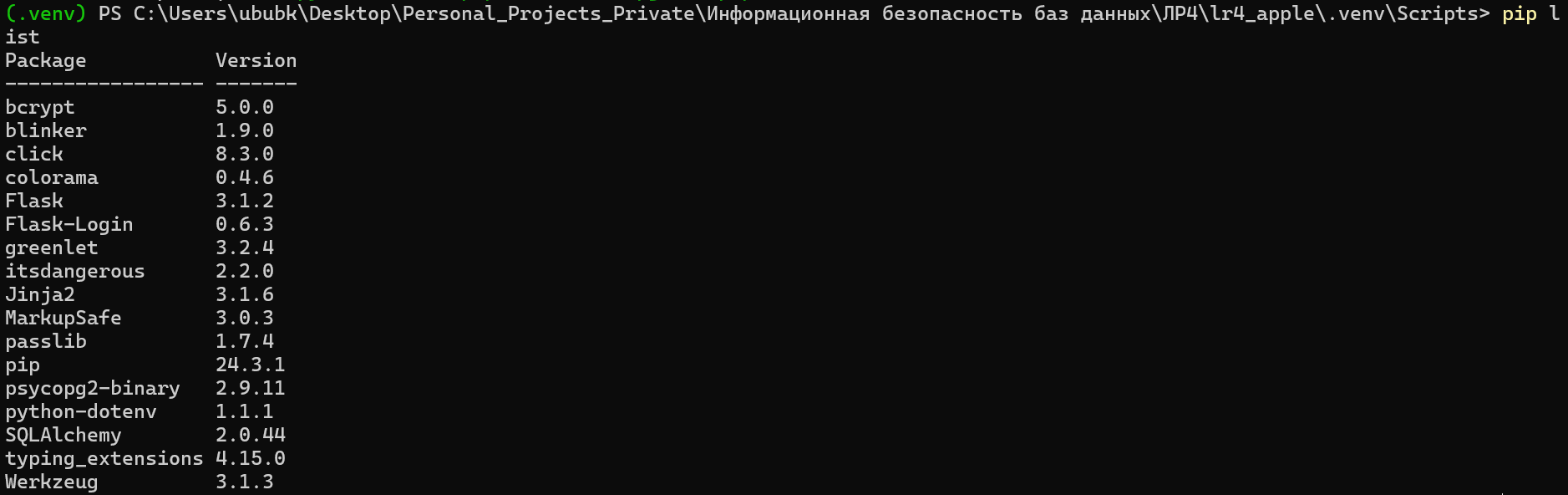
.\Activate.ps1

## Устанавливаем библиотеки

pip install flask sqlalchemy psycopg2-binary flask-login python-dotenv passlib[bcrypt]

Так же можем доподнительно проверить все ли верно установилось

pip list



## Создаем requirements.txt

flask

sqlalchemy

psycopg2-binary

flask-login

python-dotenv

passlib[bcrypt]

## Создаем security.py — хэширование паролей

# security.py

from passlib.hash import bcrypt

def hash\_password(p: str) -> str:

return bcrypt.hash(p)

def verify\_password(p: str, h: str) -> bool:

return bcrypt.verify(p, h)

## Создаем models\_apple.py — модели БД и соединение

# models\_apple.py

import os, enum

from dotenv import load\_dotenv

from sqlalchemy import (create\_engine, Column, Integer, Text, String,

Date, DateTime, Numeric, ForeignKey, Enum)

from sqlalchemy.orm import declarative\_base, sessionmaker, relationship

from flask\_login import UserMixin

load\_dotenv()

Base = declarative\_base()

# Роли ИМЕННО в приложении (не в СУБД)

class AppRole(enum.Enum):

customer = "customer"

manager = "manager"

admin = "admin"

# Пользователи приложения (для Flask-Login)

class AppUser(Base, UserMixin):

\_\_tablename\_\_ = "app\_users"

id = Column(Integer, primary\_key=True)

username = Column(String(120), unique=True, nullable=False) # можно хранить email

password\_hash = Column(String(255), nullable=False)

role = Column(Enum(AppRole), nullable=False)

# ===== Модели apple\_store (минимально-достаточные) =====

# Если имена/типы в твоей БД отличаются — поправь под свою схему.

class Customer(Base):

\_\_tablename\_\_ = "customer"

customerid = Column(Integer, primary\_key=True)

fullname = Column(Text)

email = Column(Text)

class Product(Base):

\_\_tablename\_\_ = "product"

productid = Column(Integer, primary\_key=True)

title = Column(Text)

price = Column(Numeric(10,2))

stock = Column(Integer)

class Order(Base):

\_\_tablename\_\_ = "orders"

orderid = Column(Integer, primary\_key=True)

customerid = Column(Integer, ForeignKey("customer.customerid"))

orderdate = Column(Date)

status = Column(Text)

totalsum = Column(Numeric(12,2))

itemscount = Column(Integer)

lastpaymentstatus= Column(Text)

customer = relationship("Customer")

class OrderItem(Base):

\_\_tablename\_\_ = "order\_items"

id = Column(Integer, primary\_key=True)

orderid = Column(Integer, ForeignKey("orders.orderid"))

productid = Column(Integer, ForeignKey("product.productid"))

quantity = Column(Integer)

price\_each= Column(Numeric(10,2))

class Payment(Base):

\_\_tablename\_\_ = "payments"

paymentid = Column(Integer, primary\_key=True)

orderid = Column(Integer, ForeignKey("orders.orderid"))

method = Column(Text) # card/sbp/etc

status = Column(Text) # pending/success/refunded

paid\_at = Column(DateTime)

# --- Соединение ---

def get\_engine():

user = os.getenv("DB\_USER")

pw = os.getenv("DB\_PASS")

host = os.getenv("DB\_HOST", "localhost")

db = os.getenv("DB\_NAME")

return create\_engine(f"postgresql+psycopg2://{user}:{pw}@{host}/{db}", echo=False)

engine = get\_engine()

SessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False)

# Локальный тест соединения (по желанию)

if \_\_name\_\_ == "\_\_main\_\_":

from sqlalchemy import text

with engine.connect() as conn:

print("✅ DB version:", conn.execute(text("SELECT version()")).scalar())

## Создаем auth.py — регистрация/вход/выход

# auth.py

from flask import Blueprint, render\_template, request, redirect, url\_for, flash

from flask\_login import LoginManager, login\_user, logout\_user, login\_required

from models\_apple import AppUser, AppRole, SessionLocal

from security import hash\_password, verify\_password

auth\_bp = Blueprint("auth", \_\_name\_\_)

login\_manager = LoginManager()

login\_manager.login\_view = "auth.login"

@login\_manager.user\_loader

def load\_user(user\_id):

db = SessionLocal()

return db.get(AppUser, int(user\_id))

@auth\_bp.route("/register", methods=["GET","POST"])

def register():

if request.method == "POST":

db = SessionLocal()

if db.query(AppUser).filter\_by(username=request.form["username"]).first():

flash("Пользователь уже существует"); return redirect(url\_for("auth.register"))

role = AppRole(request.form["role"])

u = AppUser(username=request.form["username"],

password\_hash=hash\_password(request.form["password"]),

role=role)

db.add(u); db.commit()

return redirect(url\_for("auth.login"))

return render\_template("register.html")

@auth\_bp.route("/login", methods=["GET","POST"])

def login():

if request.method == "POST":

db = SessionLocal()

u = db.query(AppUser).filter\_by(username=request.form["username"]).first()

if u and verify\_password(request.form["password"], u.password\_hash):

login\_user(u); return redirect(url\_for("home"))

flash("Неверные логин/пароль")

return render\_template("login.html")

@auth\_bp.route("/logout")

@login\_required

def logout():

logout\_user(); return redirect(url\_for("home"))

## Создаем crud.py — операции над товарами (для админа)

# crud.py

from models\_apple import SessionLocal, Product, Order, Payment

def create\_product(title:str, price, stock:int):

db = SessionLocal()

p = Product(title=title, price=price, stock=stock)

db.add(p); db.commit()

def update\_product\_price(productid:int, new\_price):

db = SessionLocal()

p = db.get(Product, productid)

if p:

p.price = new\_price

db.commit()

def delete\_product(productid:int):

db = SessionLocal()

p = db.get(Product, productid)

if p:

db.delete(p); db.commit()

def list\_orders(limit=50):

db = SessionLocal()

return db.query(Order).order\_by(Order.orderdate.desc()).limit(limit).all()

def list\_payments(limit=50):

db = SessionLocal()

return db.query(Payment).order\_by(Payment.paid\_at.desc()).limit(limit).all()

## Создаем app.py — маршруты, ACL и запуск

# app.py

import os, functools

from dotenv import load\_dotenv

from flask import Flask, render\_template, request, redirect, url\_for, abort, flash

from flask\_login import current\_user

from auth import auth\_bp, login\_manager

from models\_apple import (Base, engine, SessionLocal, AppRole, AppUser,

Product, Order, Payment, Customer)

from crud import create\_product, update\_product\_price, delete\_product, list\_orders, list\_payments

load\_dotenv()

app = Flask(\_\_name\_\_)

app.config["SECRET\_KEY"] = os.getenv("SECRET\_KEY")

# Flask-Login

login\_manager.init\_app(app)

app.register\_blueprint(auth\_bp)

# Создаём только таблицу app\_users (остальные — уже в apple\_store)

Base.metadata.create\_all(bind=engine)

# Матрица доступа

ACL = {

"products": {AppRole.admin, AppRole.manager},

"orders": {AppRole.admin, AppRole.manager},

"payments": {AppRole.admin, AppRole.manager},

"my\_orders": {AppRole.customer},

}

def role\_required(allowed\_roles:set[AppRole]):

def deco(fn):

@functools.wraps(fn)

def inner(\*args, \*\*kwargs):

if not current\_user.is\_authenticated:

return abort(401)

if current\_user.role not in allowed\_roles:

return abort(403)

return fn(\*args, \*\*kwargs)

return inner

return deco

# Страницы ошибок

@app.errorhandler(401)

def err401(\_):

return render\_template("errors/401.html"), 401

@app.errorhandler(403)

def err403(\_):

return render\_template("errors/403.html"), 403

@app.route("/")

def home():

return render\_template("home.html", acl=ACL)

# ====== Каталог (просмотр всем manager/admin; CRUD — только admin)

@app.route("/products", methods=["GET","POST"])

@role\_required(ACL["products"])

def products():

db = SessionLocal()

rows = db.query(Product).order\_by(Product.productid).all()

if request.method == "POST":

if current\_user.role != AppRole.admin:

flash("Только admin может изменять каталог")

return redirect(url\_for("products"))

action = request.form.get("action")

try:

if action == "create":

create\_product(request.form["title"], request.form["price"], int(request.form["stock"]))

elif action == "update":

update\_product\_price(int(request.form["productid"]), request.form["price"])

elif action == "delete":

delete\_product(int(request.form["productid"]))

except Exception as e:

flash(f"Ошибка: {e}")

return redirect(url\_for("products"))

return render\_template("tables/products.html", rows=rows)

@app.route("/orders")

@role\_required(ACL["orders"])

def orders():

rows = list\_orders()

return render\_template("tables/orders.html", rows=rows)

@app.route("/payments")

@role\_required(ACL["payments"])

def payments():

rows = list\_payments()

return render\_template("tables/payments.html", rows=rows)

# ====== Мои заказы (customer)

@app.route("/my\_orders")

@role\_required(ACL["my\_orders"])

def my\_orders():

db = SessionLocal()

# упрощённая привязка: username в app\_users = email клиента

cust = db.query(Customer).filter\_by(email=current\_user.username).first()

if not cust:

rows = []

else:

rows = db.query(Order).filter\_by(customerid=cust.customerid).order\_by(Order.orderdate.desc()).all()

return render\_template("tables/my\_orders.html", rows=rows)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

# HTML Шаблоны

## Templates

Создаем папку templates и складываем туда все нужные шаблоны

## base.html

<!doctype html>

<html lang="ru">

<head>

<meta charset="utf-8"><title>apple\_store</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<nav>

<a href="{{ url\_for('home') }}">Главная</a>

{% if current\_user.is\_authenticated %}

<span> | Вы: {{ current\_user.username }} ({{ current\_user.role.value }})</span>

<a href="{{ url\_for('auth.logout') }}">Выход</a>

{% else %}

<a href="{{ url\_for('auth.login') }}">Вход</a>

<a href="{{ url\_for('auth.register') }}">Регистрация</a>

{% endif %}

</nav>

<hr>

{% with messages = get\_flashed\_messages() %}

{% if messages %}<ul>{% for m in messages %}<li>{{ m }}</li>{% endfor %}</ul>{% endif %}

{% endwith %}

{% block content %}{% endblock %}

</body>

</html>

## home.html

{% extends "base.html" %}

{% block content %}

<h2>apple\_store — демо сервис ЛР-4</h2>

<ul>

<li><a href="{{ url\_for('products') }}">Каталог (manager/admin)</a></li>

<li><a href="{{ url\_for('orders') }}">Заказы (manager/admin)</a></li>

<li><a href="{{ url\_for('payments') }}">Платежи (manager/admin)</a></li>

<li><a href="{{ url\_for('my\_orders') }}">Мои заказы (customer)</a></li>

</ul>

{% endblock %}

## login.html

{% extends "base.html" %}

{% block content %}

<h3>Вход</h3>

<form method="post">

<input name="username" placeholder="username/email">

<input name="password" placeholder="password" type="password">

<button type="submit">Войти</button>

</form>

{% endblock %}

## register.html

{% extends "base.html" %}

{% block content %}

<h3>Регистрация</h3>

<form method="post">

<input name="username" placeholder="username/email">

<input name="password" placeholder="password" type="password">

<select name="role">

<option value="customer">customer</option>

<option value="manager">manager</option>

<option value="admin">admin</option>

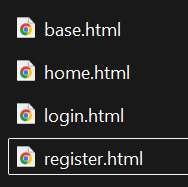
</select>

<button type="submit">Создать</button>

</form>

{% endblock %}

Как пример выполненной работы:



## Создаем папку tables и вставляем туда существующие таблицы из БД

### products.html

{% extends "base.html" %}

{% block content %}

<h3>Каталог</h3>

<table border="1" cellpadding="4">

<tr><th>ID</th><th>Название</th><th>Цена</th><th>Остаток</th></tr>

{% for r in rows %}

<tr><td>{{ r.productid }}</td><td>{{ r.title }}</td><td>{{ r.price }}</td><td>{{ r.stock }}</td></tr>

{% endfor %}

</table>

{% if current\_user.role.value == 'admin' %}

<h4>Создать</h4>

<form method="post">

<input type="hidden" name="action" value="create">

<input name="title" placeholder="Название">

<input name="price" placeholder="Цена">

<input name="stock" placeholder="Остаток">

<button>Добавить</button>

</form>

<h4>Обновить цену</h4>

<form method="post">

<input type="hidden" name="action" value="update">

<input name="productid" placeholder="ID товара">

<input name="price" placeholder="Новая цена">

<button>Обновить</button>

</form>

<h4>Удалить</h4>

<form method="post">

<input type="hidden" name="action" value="delete">

<input name="productid" placeholder="ID товара">

<button>Удалить</button>

</form>

{% endif %}

{% endblock %}

### orders.html

{% extends "base.html" %}

{% block content %}

<h3>Заказы</h3>

<table border="1" cellpadding="4">

<tr>

<th>orderid</th><th>date</th><th>status</th>

<th>totalsum</th><th>items</th><th>last payment</th><th>customer</th>

</tr>

{% for r in rows %}

<tr>

<td>{{ r.orderid }}</td><td>{{ r.orderdate }}</td><td>{{ r.status }}</td>

<td>{{ r.totalsum }}</td><td>{{ r.itemscount }}</td><td>{{ r.lastpaymentstatus }}</td>

<td>{{ r.customer.fullname if r.customer else '' }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

### payments.html

{% extends "base.html" %}

{% block content %}

<h3>Платежи</h3>

<table border="1" cellpadding="4">

<tr><th>paymentid</th><th>orderid</th><th>method</th><th>status</th><th>paid\_at</th></tr>

{% for r in rows %}

<tr><td>{{ r.paymentid }}</td><td>{{ r.orderid }}</td><td>{{ r.method }}</td><td>{{ r.status }}</td><td>{{ r.paid\_at }}</td></tr>

{% endfor %}

</table>

{% endblock %}

### my\_orders.html

{% extends "base.html" %}

{% block content %}

<h3>Мои заказы</h3>

{% if rows|length == 0 %}<p>Нет заказов или учётка не связана с клиентом.</p>{% endif %}

<table border="1" cellpadding="4">

<tr><th>orderid</th><th>date</th><th>status</th><th>sum</th></tr>

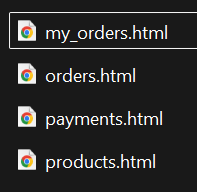
{% for r in rows %}

<tr><td>{{ r.orderid }}</td><td>{{ r.orderdate }}</td><td>{{ r.status }}</td><td>{{ r.totalsum }}</td></tr>

{% endfor %}

</table>

{% endblock %}



## Создаем папку errrors

### 401.html

{% extends "base.html" %}

{% block content %}

<h3>401 — Требуется вход</h3>

<p>Пожалуйста, выполните вход.</p>

{% endblock %}

### 403.html

{% extends "base.html" %}

{% block content %}

<h3>403 — Доступ запрещён</h3>

<p>У вашей роли нет прав на просмотр этой страницы.</p>

{% endblock %}

# Вспомогательные файлы (ручной запуск проверок)

test\_db\_connection.py - проверка соединения:

from dotenv import load\_dotenv; load\_dotenv()

import os

from sqlalchemy import create\_engine, text

u=os.getenv("DB\_USER"); p=os.getenv("DB\_PASS"); h=os.getenv("DB\_HOST"); d=os.getenv("DB\_NAME")

print(f"Подключаемся к {d} от {u}...")

try:

engine=create\_engine(f"postgresql+psycopg2://{u}:{p}@{h}/{d}")

with engine.connect() as c:

print("✅", c.execute(text("SELECT version()")).scalar())

except Exception as e:

print("❌ Ошибка:", e)

init\_users.py - создать пользователей приложения:

from models\_apple import Base, engine, SessionLocal, AppUser, AppRole

from security import hash\_password

Base.metadata.create\_all(bind=engine)

db = SessionLocal()

def ensure(u,p,r):

if not db.query(AppUser).filter\_by(username=u).first():

db.add(AppUser(username=u, password\_hash=hash\_password(p), role=r)); db.commit()

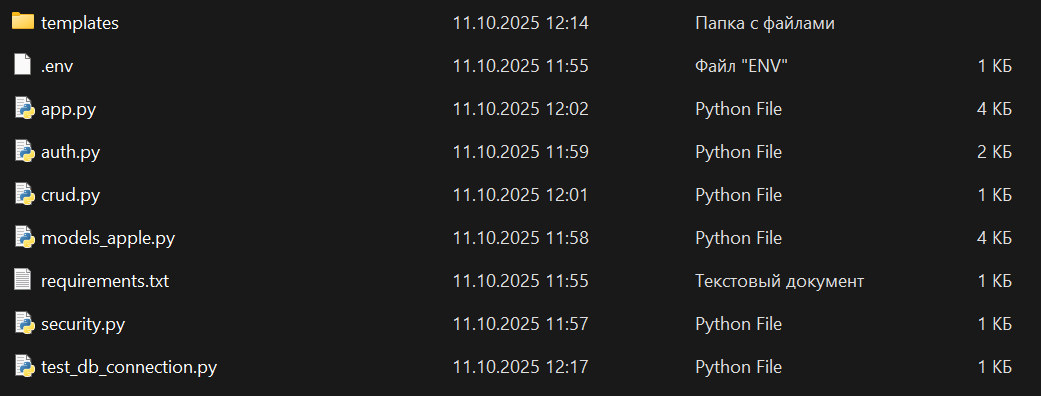
ensure("admin1","admin",AppRole.admin)

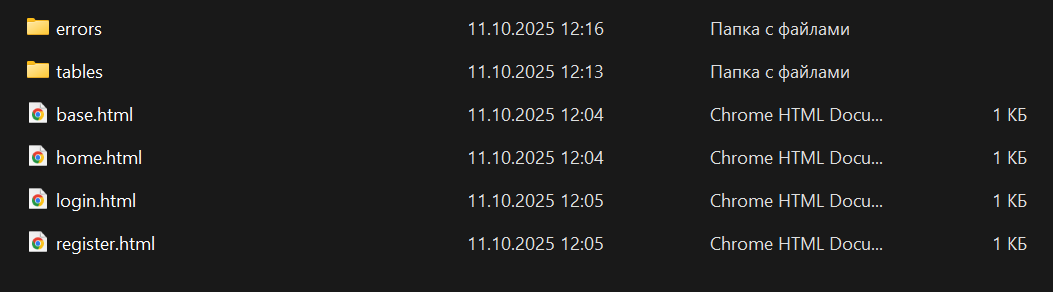
ensure("manager1","manager",AppRole.manager)

ensure("cust1@example.com","cust",AppRole.customer)

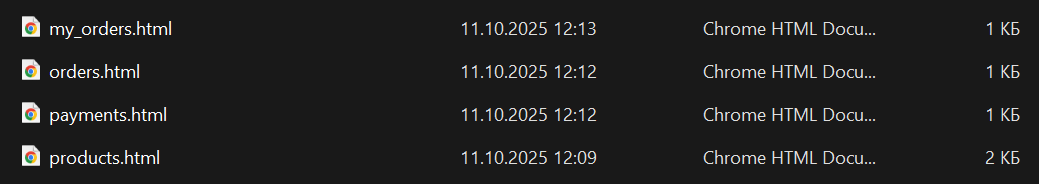
print("Готово: admin1/admin, manager1/manager, [cust1@example.com/cust](mailto:cust1@example.com/cust)")

Скриншоты:

Общий вид:

Папка templates:

Папка errors:

Папка tables:

# Запуск

1. Активируем .venv и ставим зависимости:

. .\.venv\Scripts\Activate.ps1

pip install -r requirements.txt

1. Проверяем подключение:

python test\_db\_connection.py

1. Создаем пользователей приложения:

python init\_users.py

1. Запускаем сайт:

python app.py

Открываем http://127.0.0.1:5000  
Логины для проверки:

* admin1 / **admin** → /products (CRUD), /orders, /payments
* manager1 / **manager** → /products (только просмотр), /orders, /payments
* cust1@example.com / **cust** → /my\_orders (если email клиента = этому username)

Заключение

В ходе работы разработана система защиты БД средствами PostgreSQL.

* Реализовано логирование операций с данными с помощью триггеров.
* Создана таблица секретных данных с симметричным шифрованием на основе pgcrypto.
* Настроено разграничение доступа на основе ролей пользователей, реализован принцип минимальных привилегий.

Таким образом, выполнены базовые задачи по обеспечению безопасности базы данных: контроль действий, защита чувствительных данных и управление правами доступа.

Список использованных источников

1. Новиков Б. А., Горшкова Е. А., Графеева Н. Г. **Основы технологий баз данных.** – 2-е изд. – М.: ДМК Пресс, 2020. – 582 с.
2. Хомоненко А. Д. (ред.). **Базы данных.** – 6-е изд., доп. – СПб.: КОРОНА-Век, 2009. – 736 с.
3. Документация PostgreSQL: <https://www.postgresql.org/docs/>
4. Модуль pgcrypto: <https://www.postgresql.org/docs/current/pgcrypto.html>