App.py

import os

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

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

import db as db

import models as m

import constant as const

import functools

from config import \*

from forms import \*

from werkzeug.exceptions import InternalServerError

from collections import defaultdict

from werkzeug.utils import secure\_filename

def organize\_menu\_by\_categories(menu\_items):

    categorized\_menu = defaultdict(list)

    for item in menu\_items:

        categorized\_menu[item['category\_name']].append(item)

    return categorized\_menu

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

app.config['SECRET\_KEY'] = 'key'

login\_manager = LoginManager()

login\_manager.init\_app(app)

@login\_manager.user\_loader

def load\_user(user\_id):

    return db.get\_user\_by\_id(user\_id)

def admin\_required(funct):

    """ Проверка, что пользователь является администратором. """

    try:

        # сохраняет \_\_name\_\_ и \_\_doc\_\_ за функцией.

        @functools.wraps(funct)

        # создаём "обёрнутую" функцию, имплементирующую (реализующую)

        # функционал изначальной, но с проверкой

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

            # если текущий пользователь не администратор

            if current\_user.is\_authenticated and current\_user.get\_role() != const.ADMINISTRATOR:

                # сообщаем об этом

                flash("Access denied", category='error')

                # переадресуем на главную страницу

                return redirect(url\_for('index'))

            # возвращаем результат исполнения "оборачиваемой" функции

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

        # возвращаем "обёрнутую" функцию

        return wrapped

    except Exception as ex:

        logging.error(f"An unexpected error occurred: {ex}")

        raise InternalServerError

def user\_check\_password(funct):

    """ Проверка, что пользователь сменил пароль. """

    try:

        # сохраняет \_\_name\_\_ и \_\_doc\_\_ за функцией.

        @functools.wraps(funct)

        # создаём "обёрнутую" функцию, имплементирующую (реализующую)

        # функционал изначальной, но с проверкой, что пользователь входил уже

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

            if isinstance(current\_user, m.User) and not current\_user.password\_changed:

                return redirect(url\_for('change\_password'))

            # возвращаем результат исполнения "оборачиваемой" функции

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

        # возвращаем "обёрнутую" функцию

        return wrapped

    except Exception as ex:

        logging.error(f"An unexpected error occurred: {ex}")

        raise InternalServerError

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

def login():

    try:

        if not current\_user.is\_authenticated:

            form = LoginForm()

            if form.validate\_on\_submit():

                user = db.get\_user\_by\_email(form.email.data)

                if user is not None:

                    if db.check\_user\_password(

                        user.password, form.password.data):

                        login\_user(user, remember=form.remember\_me.data)

                        flash("Success login", category='success')

                        db.change\_user\_last\_login(current\_user.id)

                        db.history(current\_user.id, 'the user has logged in')

                        return redirect("/")

                    flash("Invalid username or password", category='error')

                    return render\_template('login.html', page='login', form=form)

                flash("The user with this email was not found",

                      category='error')

                return render\_template('login.html', page='login', form=form)

            return render\_template('login.html', page='login', form=form)

        return redirect("/")

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/logout')

@login\_required

def logout():

    if current\_user.get\_id():

        db.history(current\_user.id, 'the user has logged out')

        logout\_user()

    return redirect("/")

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

def registration():

    try:

        if current\_user.is\_authenticated:

            return redirect('/')

        form = RegistrationForm()

        if form.validate\_on\_submit():

            if db.create\_user(

                form.email.data, form.password.data,

                form.lastname.data, form.firstname.data) is not None:

                flash('Registration was completed successfully!',

                    category='success')

                return redirect('/login')

            flash('An error occurred while register!',

                      category='error')

        return render\_template('register.html', page='register', form=form)

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/change\_password', methods=['POST', 'GET'])

@login\_required

def change\_password():

    try:

        form = ResetPasswordForm()

        if form.validate\_on\_submit():

            if form.confirm\_new\_password.data != form.old\_password.data:

                if db.change\_user\_password(

                    current\_user.id,

                    form.confirm\_new\_password.data) is not None:

                    db.history(current\_user.id, 'the user changed the password')

                    flash('The user\'s password has been successfully changed!', category='success')

                    return redirect('/login')

                flash('An error occurred while changing the password!',

                        category='error')

        return render\_template('change\_password.html', page='change\_password', form=form)

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/')

def index():

    try:

        return render\_template('index.html', page='\_')

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/menu')

def menu():

    try:

        categorized\_menu = organize\_menu\_by\_categories(db.get\_menu())

        return render\_template('menu.html', menu\_items=categorized\_menu)

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/add\_to\_cart', methods=['POST'])

def add\_to\_cart():

    food\_id = request.json.get('food\_id')

    price = request.json.get('price')

    try:

        result =  db.add\_food\_to\_cart(current\_user.id, food\_id, price)

        if result:

            return jsonify({'success': True, 'cart\_id': result})

    except Exception as ex:

        logging.error(ex)

        return jsonify({'success': False, 'error': str(ex)})

@app.route('/cart')

@login\_required

@user\_check\_password

def cart():

    try:

        cart\_items = db.get\_cart(current\_user.id)

        if not cart\_items:

            flash('Your cart is currently empty.', 'info')

        return render\_template(

            'cart.html', page='cart', cart\_items=cart\_items)

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/create', methods=['GET', 'POST'])

@login\_required

@user\_check\_password

def create():

    try:

        form = AddDishForm()

        categories = db.get\_category()

        form.category.choices = [(c.get('id'), c.get('name')) for c in categories]

        if form.validate\_on\_submit():

            image = form.image.data

            if image:

                filename = secure\_filename(image.filename)

                file\_path = os.path.join('path/to/save/images', filename)

                image.save(file\_path)

            else:

                file\_path = None

            if db.add\_dish(

                form.name.data, form.description.data,

                form.price.data, form.category.data, file\_path):

                flash('Новое блюдо успешно добавлено!', 'success')

                return redirect(url\_for('create'))

            flash('Новое блюдо успешно добавлено!', 'error')

            return redirect(url\_for('menu'))

        return render\_template('create.html', page='create', form=form)

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/update-quantity', methods=['POST'])

def update\_quantity():

    data = request.get\_json()

    item\_id = data.get('itemId')

    change = data.get('change')

    # Здесь код для обновления количества товара в корзине

    # Предположим, что это обновление происходит успешно

    return jsonify({'success': True})

@app.route('/place-order', methods=['POST'])

@login\_required

@user\_check\_password

def place\_order():

    # Получение данных пользователя и номера стола

    user\_id = current\_user.id

    table\_number = request.form.get('table\_number')

    if not user\_id:

        return jsonify({'success': False, 'message': 'Пользователь не авторизован'}), 401

    if not table\_number:

        return jsonify({'success': False, 'message': 'Номер стола не указан'}), 400

    try:

        # Преобразование номера стола из строки в число, если это необходимо

        table\_number = int(table\_number)

        order\_created = db.create\_order(user\_id, table\_number)

        if order\_created:

            return jsonify({'success': True, 'message': 'Заказ успешно оформлен'})

        return jsonify({'success': False, 'message': 'Не удалось оформить заказ'})

    except (ValueError, TypeError):

        return jsonify({'success': False, 'message': 'Некорректные данные номера стола'}), 400

    except Exception as ex:

        return jsonify({'success': False, 'message': str(ex)}), 500

@app.route('/about')

def about():

    try:

        return render\_template('about.html', page='menu')

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

# ВЫВОД ПРОФИЛЯ ПОЛЬЗОВАТЕЛЯ

@app.route('/profile')

@login\_required

@user\_check\_password

def profile():

    try:

        return render\_template(

            'profile.html', page='profile')

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.route('/orders')

@login\_required

@user\_check\_password

def orders():

    user\_id = session.get('user\_id')

    if not user\_id:

        return redirect(url\_for('login'))

    orders = db.get\_user\_orders(user\_id)

    # Группировка данных заказа по ID заказа

    grouped\_orders = {}

    for order in orders:

        if order['id'] not in grouped\_orders:

            grouped\_orders[order['id']] = {

                'time': order['time'],

                'summa': order['summa'],

                'status': order['status'],

                'items': []

            }

        grouped\_orders[order['id']]['items'].append({

            'food': order['food'],

            'count': order['count'],

            'summ': order['summ']

        })

    return render\_template('orders.html', orders=grouped\_orders)

@app.route('/users')

@admin\_required

@user\_check\_password

@login\_required

def users():

    try:

        return render\_template('users.html', page='users', users=db.get\_users())

    except Exception as ex:

        logging.error(ex)

        raise InternalServerError

@app.errorhandler(404)

def not\_found\_error(error):

    try:

        return render\_template('404.html', page='404', error=error), 404

    except Exception as ex:

        logging.error(f"An unexpected error occurred: {ex}")

@app.errorhandler(InternalServerError)

def internal\_error(error):

    try:

        return render\_template('500.html', page='500', error=error), 500

    except Exception as ex:

        logging.error(f"An unexpected error occurred: {ex}")

@login\_manager.unauthorized\_handler

def unauthorized():

    try:

        flash("Log in to visit this page.","error")

        return redirect(url\_for('login'))

    except Exception as ex:

        logging.error(f"An unexpected error occurred: {ex}")

        raise InternalServerError

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

    app.run(debug=True)

db.py

import config

import datetime

import psycopg2

import psycopg2.extras

import models as m

from werkzeug.security import generate\_password\_hash, check\_password\_hash

def get\_data(query, values=None):

    """Функция для получения данных из базы данных (SELECT \* FROM table\_name).

    Создается соединение (данные config считаются глобальными)

    Args:

        query (str): Запрос, исполняемый в бд.

        values (tuple, optional): Данные, по которым можно конкретизировать

        запрос (например, указание id). По умолчанию None.

    Returns:

        list: Возвращает массив с данными из базы данных

    """

    try:

        connection = psycopg2.connect(\*\*config.DB\_CONFIG)

        cursor = connection.cursor(cursor\_factory=psycopg2.extras.DictCursor)

        cursor.execute(query, values)

        return cursor.fetchall()

    except psycopg2.DatabaseError as ex:

        config.logging.error(ex)

    finally:

        connection.close()

def set\_data(query, values=None):

    """Функция, в результате которой отправляюся данные в базу данных

    (INSERT, UPDATE, DELETE)

    Args:

        query (str): Запрос, исполняемый в бд.

        values (tuple, optional): Данные, которые будут отправлены

        в базу данных в качестве изменяемых. Defaults to None.

    Returns:

        int: В случае, если необходимо вернуть id внесенной записи

        "RETURNING id;", то  возвращает это значение

    """

    try:

        connection = psycopg2.connect(\*\*config.DB\_CONFIG)

        cursor = connection.cursor()

        cursor.execute(query, values)

        connection.commit()

        if "RETURNING id;" in query:

            return cursor.fetchone()[0]

    except (Exception, psycopg2.DatabaseError) as ex:

        config.logging.error(ex)

    finally:

        connection.close()

def get\_user\_by\_id(id):

    try:

        user = get\_data("""SELECT u.id, u.email, u.password, u.lastname,

                        u.firstname, r.role, u.last\_activity,

                        u.password\_changed, u.status

                        FROM users u JOIN roles r on u.role = r.id

                        WHERE u.id = %s""", (id,))

        if user != []:

            user = user[0]

            return m.User(

                user.get('id'), user.get('email'), user.get('password'),

                user.get('lastname'), user.get('firstname'), user.get('role'),

                user.get('last\_activity'), user.get('password\_changed'),

                user.get('status'))

    except Exception as ex:

        config.logging.error(ex)

def get\_user\_by\_email(email):

    try:

        user = get\_data("""SELECT u.id, u.email, u.password, u.lastname,

                        u.firstname, r.role, u.last\_activity,

                        u.password\_changed, u.status

                        FROM users u JOIN roles r on u.role = r.id

                        WHERE u.email = %s""", (email,))

        if user != []:

            user = user[0]

            return m.User(

                user.get('id'), user.get('email'), user.get('password'),

                user.get('lastname'), user.get('firstname'), user.get('role'),

                user.get('last\_activity'), user.get('password\_changed'),

                user.get('status'))

    except Exception as ex:

        config.logging.error(ex)

def set\_user\_last\_activity(user\_id):

    """\_summary\_

    Args:

        user\_id (\_type\_): \_description\_

    Returns:

        \_type\_: \_description\_

    """

    last\_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    try:

        return set\_data("""UPDATE users SET last\_activity = %s WHERE id = %s

                        RETURNING id;""", (last\_date, user\_id))

    except Exception as ex:

        config.logging.error(ex)

def get\_users():

    try:

        return get\_data("""SELECT u.id, u.email, u.lastname,

                        u.firstname, r.role, u.status, u.last\_activity

                        FROM users u JOIN roles r on u.role = r.id

                        ORDER BY u.lastname""")

    except Exception as ex:

        config.logging.error(ex)

def create\_user(email, password, lastname, firstname):

    password\_hash = generate\_password\_hash(password)

    try:

        role = get\_data("SELECT id FROM roles WHERE role = 'user';")

        if role:

            role\_id = role[0].get('id')

            return set\_data(

                '''INSERT INTO users (email, password, lastname, firstname,

                 role, status, password\_changed) VALUES

                (%s, %s, %s, %s, %s, True, False) RETURNING id;''',

                (email, password\_hash, lastname, firstname, role\_id))

    except Exception as error:

        config.logging.error(error)

def change\_user\_password(user\_id, new\_password):

    password\_hash = generate\_password\_hash(new\_password)

    try:

        return set\_data(

            '''UPDATE users SET password = %s,

            password\_changed = true WHERE id = %s RETURNING id;''',

            (password\_hash, user\_id,))

    except Exception as error:

        config.logging.error(error)

def change\_user\_last\_login(user\_id):

    now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    try:

        return set\_data(

            '''UPDATE users SET last\_activity = %s

            WHERE id = %s RETURNING id;''',

            (now, user\_id,))

    except Exception as error:

        config.logging.error(error)

def check\_user\_password(password\_hash, password):

    try:

        return check\_password\_hash(password\_hash, password)

    except Exception as ex:

        config.logging.error(ex)

def blocked\_password(user\_id):

    try:

        return set\_data(

            '''UPDATE users SET status = False

            WHERE users.id = %s RETURNING id;''',

            (user\_id,))

    except Exception as error:

        config.logging.error(error)

def unblocked\_password(user\_id):

    try:

        return set\_data(

            '''UPDATE users SET status = True

            WHERE users.id = %s RETURNING id;''',

            (user\_id,))

    except Exception as error:

        config.logging.error(error)

def get\_roles():

    try:

        return get\_data('SELECT \* FROM roles')

    except Exception as ex:

        config.logging.error(ex)

# def edit\_user(email, lastname, firstname, phone, role, id):

#     try:

#         return set\_data("""

#                         UPDATE users SET email = %s,  lastname = %s,

#                         firstname = %s, phone = %s, role = %s

#                         WHERE id = %s RETURNING id;""",

#                         (email, lastname, firstname, phone, role, id))

#     except Exception as ex:

#         config.logging.error(ex)

def get\_admins\_count():

    try:

        role = get\_data("SELECT id FROM roles WHERE role = 'admin';")

        if role:

            role\_id = role[0].get('id')

        return get\_data('SELECT count(\*) FROM users WHERE role = %s;', (role\_id,))

    except Exception as ex:

        config.logging.error(ex)

def history(user, action):

    try:

        now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        return set\_data(

            '''INSERT INTO history ("user", action, time) VALUES (%s, %s, %s)

            RETURNING id;''',

            (user, action, now))

    except Exception as ex:

        config.logging.error(ex)

def get\_menu():

    try:

        return get\_data("""

        SELECT m.\*, c.name as category\_name

        FROM menu m

        JOIN category c ON m.category = c.id

        ORDER BY m.category;

        """)

    except Exception as ex:

        config.logging.error(ex)

def add\_food\_to\_cart(user\_id, food\_id, price):

    try:

        return set\_data(

            '''INSERT INTO cart ("user", food, count, summ)

            VALUES (%s, %s, %s, %s) RETURNING id;''',

            (user\_id, food\_id, 1, price))

    except Exception as ex:

        config.logging.error(ex)

def get\_cart(user\_id):

    try:

        return get\_data('''SELECT c.id, m.name, c.count, c.summ FROM cart as c

                        JOIN menu as m ON m.id = c.food

                        WHERE "user" = %s;''', (user\_id,))

    except Exception as ex:

        config.logging.error(ex)

def get\_category():

    try:

        return get\_data('''SELECT \* FROM category;''', ())

    except Exception as ex:

        config.logging.error(ex)

def add\_dish(name, description, price, category, photo):

    try:

        return set\_data('''INSERT INTO menu

                        (name, description, price, category, photo)

                        VALUES (%s, %s, %s, %s, %s) RETURNING id;''',

                        (name, description, price, category, photo))

    except Exception as ex:

        config.logging.error(ex)

def update\_cart\_count(item\_id, change, user\_id):

    """

    Обновляет количество указанного товара в корзине пользователя.

    Args:

        item\_id (int): ID товара в корзине.

        change (int): Изменение количества товара (может быть положительным или отрицательным).

        user\_id (int): ID пользователя.

    Returns:

        tuple: (bool, str) успех операции и сообщение.

    """

    try:

        connection = psycopg2.connect(\*\*config.DB\_CONFIG)

        cursor = connection.cursor()

        # Начало транзакции

        connection.autocommit = False

        # Получить текущее количество товара в корзине

        cursor.execute("SELECT count FROM cart WHERE user\_id = %s AND food = %s", (user\_id, item\_id))

        result = cursor.fetchone()

        if result:

            current\_quantity = result[0]

            new\_quantity = current\_quantity + change

            if new\_quantity <= 0:

                # Если количество становится нулевым или меньше, удаляем товар из корзины

                cursor.execute("DELETE FROM cart WHERE user\_id = %s AND food = %s", (user\_id, item\_id))

            else:

                # Обновляем количество товара в корзине

                cursor.execute("UPDATE cart SET count = %s WHERE user\_id = %s AND food = %s", (new\_quantity, user\_id, item\_id))

            # Подтверждаем изменения

            connection.commit()

            return True, "Количество успешно обновлено"

        else:

            return False, "Товар не найден в корзине"

    except Exception as e:

        if connection:

            connection.rollback()  # Откатываем изменения в случае ошибки

        config.logging.error(str(e))

        return False, str(e)

    finally:

        if connection:

            connection.close()

def create\_order(user\_id, table\_number):

    """

    Создает новый заказ, копирует содержимое корзины в список заказов и очищает корзину.

    Args:

        user\_id (int): ID пользователя, который делает заказ.

    Returns:

        bool: True, если заказ успешно создан и корзина очищена; False в случае ошибки.

    """

    try:

        connection = psycopg2.connect(\*\*config.DB\_CONFIG)

        cursor = connection.cursor()

        # Начать транзакцию

        connection.autocommit = False

        # Получить данные из корзины

        cursor.execute("SELECT food, count, count \* summ AS total FROM cart WHERE user\_id = %s", (user\_id,))

        cart\_items = cursor.fetchall()

        if not cart\_items:

            raise Exception("Корзина пуста")

        # Расчет суммы заказа

        total\_sum = sum(item['total'] for item in cart\_items)

        # Создание заказа

        cursor.execute("INSERT INTO orders (user, time, summa, table, status) VALUES (%s, NOW(), %s, %s, 1) RETURNING id;",

                       (user\_id, total\_sum, table\_number))

        order\_id = cursor.fetchone()[0]

        # Добавление элементов в order\_list

        for item in cart\_items:

            cursor.execute("INSERT INTO order\_list (order, food, count, summ) VALUES (%s, %s, %s, %s);",

                           (order\_id, item['food'], item['count'], item['total']))

        # Очистка корзины

        cursor.execute("DELETE FROM cart WHERE user\_id = %s;", (user\_id,))

        # Подтверждение транзакции

        connection.commit()

        return True

    except Exception as ex:

        config.logging.error(ex)

        if connection:

            connection.rollback()

        return False

    finally:

        if connection:

            connection.close()

def get\_user\_orders():

    try:

        return get\_data("""

    SELECT o.id, o.time, o.summa, o.status, ol.food, ol.count, ol.summ

    FROM orders o

    JOIN order\_list ol ON o.id = ol.order

    WHERE o.user = %s

    ORDER BY o.id;

    """, ())

    except Exception as ex:

        config.logging.error(ex)

forms.py

from flask\_wtf import FlaskForm

from wtforms import (

    DecimalField, StringField, PasswordField, SubmitField,

    BooleanField, SelectField, TextAreaField)

from wtforms.validators import (

    DataRequired, EqualTo, ValidationError, Length, Email, NumberRange)

from wtforms.widgets import TextArea

from flask\_wtf.file import FileField, FileAllowed, FileRequired

import re

class PasswordValidator:

    def \_\_init\_\_(self, password):

        self.password = password

    def is\_valid\_length(self, min\_length):

        return len(self.password) >= min\_length

    def has\_uppercase(self):

        return any(c.isupper() for c in self.password)

    def has\_lowercase(self):

        return any(c.islower() for c in self.password)

    def has\_digit(self):

        return any(c.isdigit() for c in self.password)

    def has\_special\_character(self):

        special\_chars = r"[~\!@#\$%\^&\\*\(\)\_\+{}\":;'\[\]]"

        return bool(re.search(special\_chars, self.password))

class LoginForm(FlaskForm):

    email = StringField('Электронная почта', validators=[DataRequired()])

    password = PasswordField('Пароль', validators=[DataRequired()])

    remember\_me = BooleanField('Запомнить меня')

    submit = SubmitField('Войти')

    def validate\_password(self, password):

        validator = PasswordValidator(password.data)

        if not validator.is\_valid\_length(8):

            raise ValidationError('Пароль должен быть не менее 8 символов')

        if not validator.has\_uppercase():

            raise ValidationError('Пароль должен содержать хотя бы одну заглавную букву')

        if not validator.has\_lowercase():

            raise ValidationError('Пароль должен содержать хотя бы одну строчную букву')

        if not validator.has\_digit():

            raise ValidationError('Пароль должен содержать хотя бы одну цифру')

        if not validator.has\_special\_character():

            raise ValidationError('Пароль должен содержать хотя бы один специальный символ')

class RegistrationForm(FlaskForm):

    email = StringField('Электронная почта', validators=[DataRequired(), Email()])

    lastname = StringField('Фамилия', validators=[DataRequired()])

    firstname = StringField('Имя', validators=[DataRequired()])

    password = PasswordField('Пароль', validators=[DataRequired()])

    confirm\_password = PasswordField('Подтвердите пароль',

                                     validators=[DataRequired(), EqualTo('password')])

    submit = SubmitField('Зарегистрироваться')

    def validate\_password(self, password):

        validator = PasswordValidator(password.data)

        if not validator.is\_valid\_length(8) or not validator.has\_uppercase() or not validator.has\_lowercase() or not validator.has\_digit() or not validator.has\_special\_character():

            raise ValidationError(

                '''Пароль должен соответствовать следующим требованиям:\n

                Пароль должен быть не менее 8 символов длиной.\n

                Пароль должен содержать хотя бы одну заглавную букву.\n

                Пароль должен содержать хотя бы одну строчную букву.\n

                Пароль должен содержать хотя бы одну цифру.\n

                Пароль должен содержать хотя бы один специальный символ.\n''')

class ResetPasswordForm(FlaskForm):

    old\_password = PasswordField('Старый пароль', validators=[DataRequired()])

    new\_password = PasswordField('Новый пароль', validators=[DataRequired()])

    confirm\_new\_password = PasswordField(

        'Подтвердите новый пароль', validators=[DataRequired(), EqualTo('new\_password')])

    submit = SubmitField('Сменить пароль')

    def validate\_new\_password(self, new\_password):

        validator = PasswordValidator(new\_password.data)

        if not validator.is\_valid\_length(8) or not validator.has\_uppercase() or not validator.has\_lowercase() or not validator.has\_digit() or not validator.has\_special\_character():

            raise ValidationError('Новый пароль должен соответствовать определенным требованиям')

    def validate\_old\_password(self, old\_password\_field):

        if old\_password\_field.data == self.new\_password.data:

            raise ValidationError('Новый пароль должен отличаться от старого')

class AddDishForm(FlaskForm):

    name = StringField('Название', validators=[DataRequired(), Length(min=2, max=100)])

    description = StringField('Описание', validators=[DataRequired(), Length(min=10, max=500)])

    price = DecimalField('Цена', validators=[DataRequired(), NumberRange(min=0)])

    category = SelectField('Категория', choices=[], coerce=int, validators=[DataRequired()])

    image = FileField('Фото', validators=[FileAllowed(['jpg', 'png'], 'Images only!')])

const.py

import os

# роли пользователей в системе

ADMINISTRATOR = 'admin'

# Путь к текущему каталогу

CURRENT\_DIRECTORY = os.path.dirname(os.path.abspath(\_\_file\_\_))

# Путь к папке "static" в текущем каталоге

STATIC\_FOLDER = os.path.join(CURRENT\_DIRECTORY, 'static')

# Путь к папке 'logs" в текущем каталоге

LOG\_FOLDER = os.path.join(CURRENT\_DIRECTORY, 'logs')

# Путь к папке "images" в текущем каталоге

IMAGES\_FOLDER = os.path.join(STATIC\_FOLDER, 'img')

Base.html

<!DOCTYPE html>

<!-- {{ url\_for('static', filename='') }}-->

<html lang="en">

    <head>

        <meta charset="UTF-8">

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

        <link rel="preconnect" href="https://fonts.googleapis.com">

        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

        <link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@300;400;500;600;700;800&display=swap"

              rel="stylesheet">

        <link rel="preconnect" href="https://fonts.googleapis.com">

        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

        <link href="https://fonts.googleapis.com/css2?family=Tinos:wght@400;700&display=swap"

              rel="stylesheet">

        <link rel="preconnect" href="https://fonts.googleapis.com">

        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

        <link href="https://fonts.googleapis.com/css2?family=Open+Sans:ital@1&display=swap"

              rel="stylesheet">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>{% block title %}

            Про рыбу

        {% endblock title %}</title>

    </head>

    <body>

        <div class='header'>

            <div class='container'>

                <div class='header-line'>

                    <div class='header-logo'>

                        <img src="{{ url\_for('static', filename='img/logo.png') }}" alt=" ">

                    </div>

                    <div class='nav'>

                        <a class='nav-item' href="/">ГЛАВНАЯ</a>

                        <a class='nav-item' href="/menu">МЕНЮ</a>

                        <a class='nav-item' href="/about">О НАС</a>

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

                            <a class='nav-item' href="/profile">КАБИНЕТ</a>

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

                            <a class='nav-item' href="/users">ПОЛЬЗОВАТЕЛИ</a>

                            {% endif%}

                            {% if current\_user.role =='staff' %}

                            <a class='nav-item' href="/users">ПОЛЬЗОВАТЕЛИ</a>

                            {% endif%}

                            {% if current\_user.role in ['admin', 'staff'] %}

                            <a class='nav-item' href="/orders">ЗАКАЗЫ</a>

                            {% endif%}

                            <a class='nav-item' href="/logout">ВЫХОД</a>

                        {%else%}

                            <a class='nav-item' href="/login">ВХОД</a>

                            {%endif%}

                    </div>

                    <div class='cart'>

                        <a href="/cart">

                            <img src="{{ url\_for('static', filename='img/cart.png') }}" alt=" " class="cart-img">

                        </a>

                    </div>

                    <div class='phone'>

                        <div class='phone-holder'>

                            <div class='phone-img'>

                            <img src="{{ url\_for('static', filename='img/phone.png') }}" alt=" ">

                            </div>

                            <div class='number'>

                                <a class='num' href='#'>+7(917)-313-37-14</a>

                            </div>

                        </div>

                        <div class='phone-text'>

                            Свяжитесь с нами для

                            <br>

                            бронирования

                        </div>

                    </div>

                    <div class='btn'>

                        <a class='button' href='/cart'>КОРЗИНА</a>

                    </div>

                    <div class='burger-menu'>

                        <button id='burger'>

                            <img src="{{ url\_for('static', filename='img/burger-img.png') }}" alt=" " >

                        </button>

                    </div>

                </div>

                <div class='header-down'>

                    {% block header\_down %}

                    <div class='header-title'>

                        Добро пожаловать в

                        <div class='header-subtitle'>Про Рыбу</div>

                        <div class='header-suptitle'>ДОМ ЛУЧШЕЙ ЕДЫ</div>

                        <div class='header-bth'>

                            <a href='/menu' class='header-button'>VIEW MENU</a>

                        </div>

                    </div>

                    {% endblock %}

                </div>

            </div>

        </div>

        {% for cat, msg in get\_flashed\_messages(True) %}

            <div class="flash {{ cat }}">

                <h3>{{ msg }}</h3>

                <span class="close-button">✕</span>

            </div>

        {% endfor %}

        {% block content %}

        {% endblock content %}

        <script>

            // Функция для скрытия сообщений Flash при нажатии на кнопку закрытия

            function hideFlashMessage(flashMessage) {

               flashMessage.style.display = 'none';

           }

           // Функция для добавления обработчиков событий на кнопки закрытия

           function addCloseButtonHandlers() {

               var closeButtons = document.getElementsByClassName('close-button');

               for (var i = 0; i<closeButtons.length; i++) {

                   var closeButton = closeButtons[i];

                   closeButton.addEventListener('click', function () {

                       hideFlashMessage(this.parentNode);

                   });

               }

           }

           // Функция для скрытия сообщений Flash после определенного времени

           /\*function hideFlashMessages() {

               var flashMessages = document.getElementsByClassName('flash');

               for (var i = 0; i < flashMessages.length; i++) {

                   var flashMessage = flashMessages[i];

                   setTimeout(function () {

                       flashMessage.style.display = 'none';

                   }, 15000); // Изменить значение 5000 на нужное вам время в миллисекундах (здесь 5 секунд)

               }

           }\*/

           // Вызов функции для скрытия сообщений Flash при загрузке страницы

           window.onload = function () {

               //hideFlashMessages();

               addCloseButtonHandlers();

           };

        </script>

        <div class='footer'>Про Рыбу</div>

        <script src="{{ url\_for('static', filename='js/main.js') }}"></script>

    </body>

</html>