ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"

Факультет «Институт международного развития и партнерства»
Направление подготовки «45.03.04 Интеллектуальные системы в гуманитарной сфере»
Бакалаврская программа «Иностранные языки и информационные технологии»

Отчет

по дисциплине «Web Programm»

Тема Экспериментальный отчет по проекту "Сервис персонального финансового управления"

Выполнил: Ван Цзынин

Группа D3310

Санкт-Петербург

2025

I. Обзор проекта

Данный проект направлен на разработку сервиса личных финансовых операций. Он предоставляет функции регистрации и входа пользователей, а также управление доходами и расходами, чтобы помочь пользователям лучше контролировать свои личные финансы. Проект разработан с использованием фреймворка FastAPI, для операций с базой данных применяется SQLModel, а для аутентификации пользователей - JWT-токены.

II. Реализованные конечные точки

1. Конечные точки, связанные с пользователями

- Конечная точка регистрации
    - Путь: /register
    - Метод запроса: POST
    - Функция: Принимает информацию о регистрации пользователя (имя пользователя, электронная почта, пароль) и сохраняет данные пользователя в базе данных. Пароль подвергается хэшированию для усиления безопасности.
    - Реализация кода:

```
python
@app.post("/register", response_model=UserRead)
async def register(request: Request, session: Session = Depends(get_session)):
    form_data = await request.form()
    user_data = UserCreate(
        username=form_data.get("username"),
        email=form_data.get("email"),
        password=form_data.get("password")
    )
    db_user = create_user(session, user_data)
    return db_user
```

- Конечная точка входа
    - Путь: /login

- Метод запроса: POST
- Функция: Проверяет введенное пользователем имя пользователя и пароль. При успешной проверке генерирует JWT-токен доступа и возвращает его пользователю.
- Реализация кода:

```
python
@app.post("/login", response_model=Token)
async def login(request: Request, session: Session = Depends(get_session)):
    form_data = await request.form()
    form = OAuth2PasswordRequestForm(
        username=form_data.get("username"),
        password=form_data.get("password")
    )
    user = authenticate_user(session, form.username, form.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW - Authenticate": "Bearer"},
        )
    access_token = create_access_token(
        data={"sub": user.username}
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

- Конечная точка получения информации о текущем пользователе
  - Путь: /users/me
  - Метод запроса: GET
  - Функция: Проверяет JWT-токен, который пользователь передаёт, и возвращает информацию о текущем вошедшем пользователе.
  - Реализация кода:

```
python
@app.get("/users/me", response_model=UserRead)
def read_users_me(current_user: User = Depends(get_current_user)):
    return current_user
```

- Конечная точка получения списка всех пользователей (добавлена для демонстрации, может быть изменена в соответствии с реальными потребностями)
    - Путь: /users
    - Метод запроса: GET
    - Функция: Извлекает информацию о всех пользователях из базы данных и отображает её в виде HTML-таблицы (или может быть настроена на возврат в формате JSON, в зависимости от конкретного сценария использования).
    - Реализация кода:

```
python
@app.get("/users", response_class=HTMLResponse)
def get_users(session: Session = Depends(get_session)):
    users = get_all_users(session)
    html_content = """
    <html>
        <body>
            <h1>User List</h1>
            <table border="1">
                <tr>
                    <th>ID</th>
                    <th>Username</th>
                    <th>Email</th>
                </tr>
    """
    for user in users:
        html_content += f"""
                <tr>
                    <td>{user.id}</td>
                    <td>{user.username}</td>
                    <td>{user.email}</td>
                </tr>
        """
    html_content += """
            </table>
        </body>
    </html>
    """
    return html_content
```

2. Конечные точки, связанные с финансовым управлением

- Конечная точка создания записи о доходах
    - Путь: /incomes
    - Метод запроса: POST
    - Функция: Принимает информацию о создании записи о доходах пользователя (сумма, источник, дата) и сохраняет запись в базе данных, связывая её с идентификатором текущего вошедшего пользователя.
    - Реализация кода:

```
python
@app.post("/incomes", response_model=IncomeRead)
def create_income_endpoint(income: IncomeCreate, current_user: User =
Depends(get_current_user),
                           session: Session = Depends(get_session)):
    db_income = create_income(session, income, current_user.id)
    return db_income
```

- Конечная точка получения списка записей о доходах текущего пользователя
    - Путь: /incomes
    - Метод запроса: GET
    - Функция: Запрашивает все записи о доходах текущего вошедшего пользователя из базы данных и возвращает их.
    - Реализация кода:

```
python
@app.get("/incomes", response_model=list[IncomeRead])
def get_incomes_endpoint(current_user: User = Depends(get_current_user), session:
Session = Depends(get_session)):
    return get_incomes(session, current_user.id)
```

- Конечная точка создания записи о расходах
    - Путь: /expenses
    - Метод запроса: POST
    - Функция: Принимает информацию о создании записи о расходах пользователя (сумма, категория, дата) и сохраняет запись в базе данных, связывая её с идентификатором текущего вошедшего пользователя.
    - Реализация кода:

```
python
@app.post("/expenses", response_model=ExpenseRead)
def create_expense_endpoint(expense: ExpenseCreate, current_user: User =
Depends(get_current_user),
                            session: Session = Depends(get_session)):
    db_expense = create_expense(session, expense, current_user.id)
    return db_expense
```

- Конечная точка получения списка записей о расходах текущего пользователя
    - Путь: /expenses
    - Метод запроса: GET
    - Функция: Запрашивает все записи о расходах текущего вошедшего пользователя из базы данных и возвращает их.
    - Реализация кода:

```
python
@app.get("/expenses", response_model=list[ExpenseRead])
def get_expenses_endpoint(current_user: User = Depends(get_current_user), session:
Session = Depends(get_session)):
    return get_expenses(session, current_user.id)
```

III. Модели

1. Модель пользователя (User)

- Определение:

```
python
class User(SQLModel, table=True):
    id: int = Field(default=None, primary_key=True)
    username: str = Field(unique=True, index=True)
    email: str = Field(unique=True, index=True)
    hashed_password: str
    incomes: List["Income"] = Relationship(back_populates="user")
    expenses: List["Expense"] = Relationship(back_populates="user")
```

- Описание: Эта модель представляет информацию о пользователе, включая идентификатор пользователя, имя пользователя, электронную почту и хэшированный пароль. Также через поля отношений `incomes` и `expenses` устанавливается связь с моделями доходов и расходов, что упрощает запросы записей о доходах и расходах, связанных с пользователем.

2. Модель доходов (Income)

- Определение:

```
python
class Income(SQLModel, table=True):
    id: int = Field(default=None, primary_key=True)
    amount: float
    source: str
    date: datetime = Field(default=datetime.utcnow)
    user_id: int = Field(foreign_key="user.id")
    user: Optional[User] = Relationship(back_populates="incomes")
```

- Описание: Используется для записи информации о доходах пользователя, включая идентификатор дохода, сумму, источник и дату. Через внешний ключ `user_id` связана с моделью пользователя, а поле `user` позволяет обратно запросить связанного пользователя.

3. Модель расходов (Expense)

- Определение:

```
python
class Expense(SQLModel, table=True):
    id: int = Field(default=None, primary_key=True)
    amount: float
    category: str
    date: datetime = Field(default=datetime.utcnow)
    user_id: int = Field(foreign_key="user.id")
    user: Optional[User] = Relationship(back_populates="expenses")
```

- Описание: Используется для записи информации о расходах пользователя, включая идентификатор расхода, сумму, категорию и дату. Через внешний ключ `user_id` связана с моделью пользователя, а поле `user` позволяет обратно запросить связанного пользователя.

IV. Код для подключения к базе данных

1. Конфигурация подключения к базе данных

В файле `database.py` настраивается информация о подключении к базе данных:

```
python
from sqlmodel import SQLModel, create_engine
from dotenv import load_dotenv
import os

load_dotenv()
DATABASE_URL = "mysql+pymysql://root:wzn001021@localhost:3306/personal_finance"
engine = create_engine(DATABASE_URL, echo=True)


def create_db_and_tables():
    SQLModel.metadata.create_all(engine)
```

2. Объяснение

Во - первых, с помощью `load_dotenv()` загружаются переменные окружения (если используются для управления информацией о подключении к базе данных). Затем определяется `DATABASE_URL`, в котором указывается тип базы данных (в данном случае MySQL), имя пользователя, пароль, адрес хоста и имя базы данных. Создается базовый движок с использованием `create_engine`, а параметр `echo = True` позволяет выводить SQL - запросы в консоль, что облегчает отладку. Функция `create_db_and_tables` используется для создания всех определенных в моделях таблиц в базе данных (на основе метаданных SQLModel).

V. Ссылка на репозиторий проекта

Исходный код проекта размещен на GitHub по адресу: https://github.com/lt9rookie/Finance_Project. В корневой директории репозитория находится рабочая версия кода, а история коммитов наглядно демонстрирует процесс итеративной разработки. Для ознакомления с финальной версией рекомендуется переключиться на ветку [название ветки].

VI. Заключение

В рамках проекта была успешно разработана серверная часть сервиса для управления личными финансами, включающая:

Реализацию API-эндпоинтов

Систему управления пользователями

Модуль учета доходов/расходов

Интеграцию с базой данных

Использование GitHub для контроля версий обеспечивает:

✓ Возможность командной разработки

✓ Прозрачность изменений кода

✓ Удобное развертывание обновлений

Перспективы развития:

- **Генерация финансовых отчетов**
- **Внедрение системы бюджетного планирования**
- **Оптимизация производительности API**