Отчёт о проделанной работе

Отчёт демонстрирует структуру проекта и тестовые запуски с проверкой работоспособности заявленного функционала.

1) Структура проекта

Проект построен в соответствии с принципами микросервисного подхода. Все компоненты разделены по функциональным областям. Структура показана на рисунке 1.

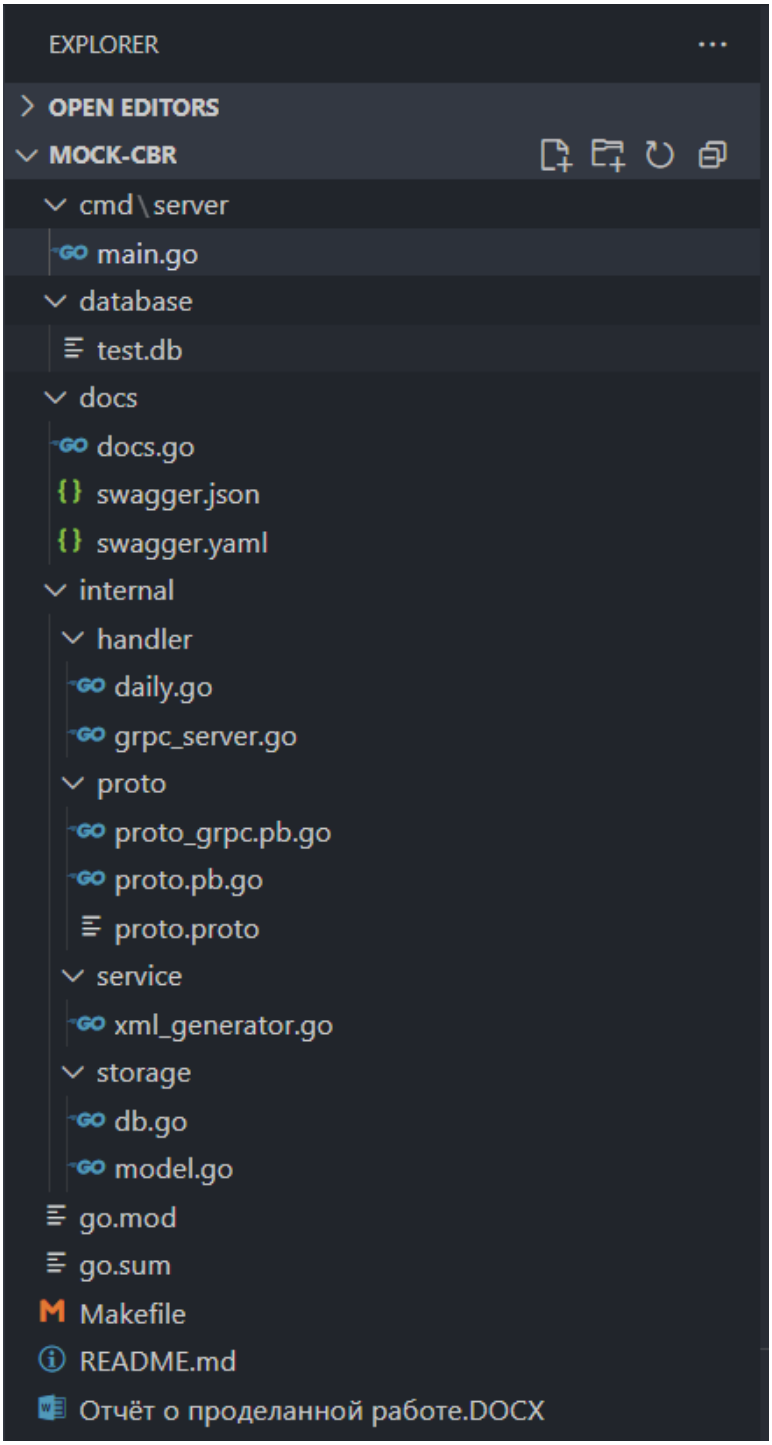

Рисунок 1 — Структура проекта

2) Makefile

Для удобства взаимодействия с проектом реализован Makefile, включающий основные команды сборки и запуска. Это позволяет запускать проект и обновлять документацию одной строкой. Сам файл представлен на рисунке 2.

Makefile

```
# Управление проектом
# Запуск сервера
run:
    go run cmd/server/main.go

# Очистка бд и временных файлов
clean:
    @if exist database\test.db ( \
        del /Q database\test.db && \
        echo Database was deleted \
    ) else ( \
        echo File database\test.db not found \
    )

# Установка зависимостей
deps:
    go mod tidy

# Свагтер
swagger:
    swag init --generalInfo cmd/server/main.go

# Проверка кода
lint:
    go fmt ./...

# Навигатор команд
help:
    @echo "make run    - Server start"
    @echo "make clean  - Delete database and temporary files"
    @echo "make deps   - Installing dependencies"
    @echo "make swagger   - Creating swagger"
    @echo "make lint   - Code formatting"
```

Рисунок 2 — Структура Makefile

3) Swagger

Для REST API реализована автоматическая Swagger-документация. Она генерируется с помощью библиотеки swaggo/swag и находится по http://localhost:8080/swagger/index.html. Документация описывает маршрут GET /daily, параметры запроса (date) и возможные ответы (успешный XML или ошибка 500). Конечный вид Swagger представлен на рисунке 3.

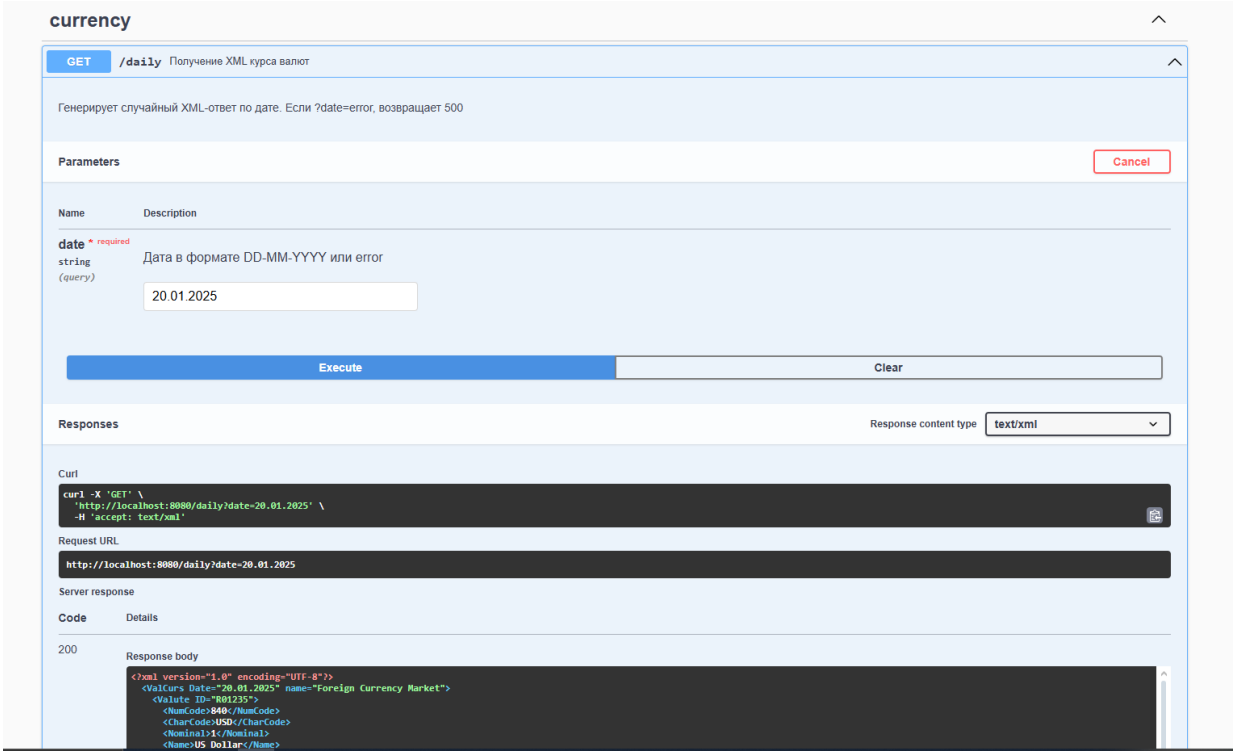

Рисунок 3 — Демонстрация кода 200 в Swagger

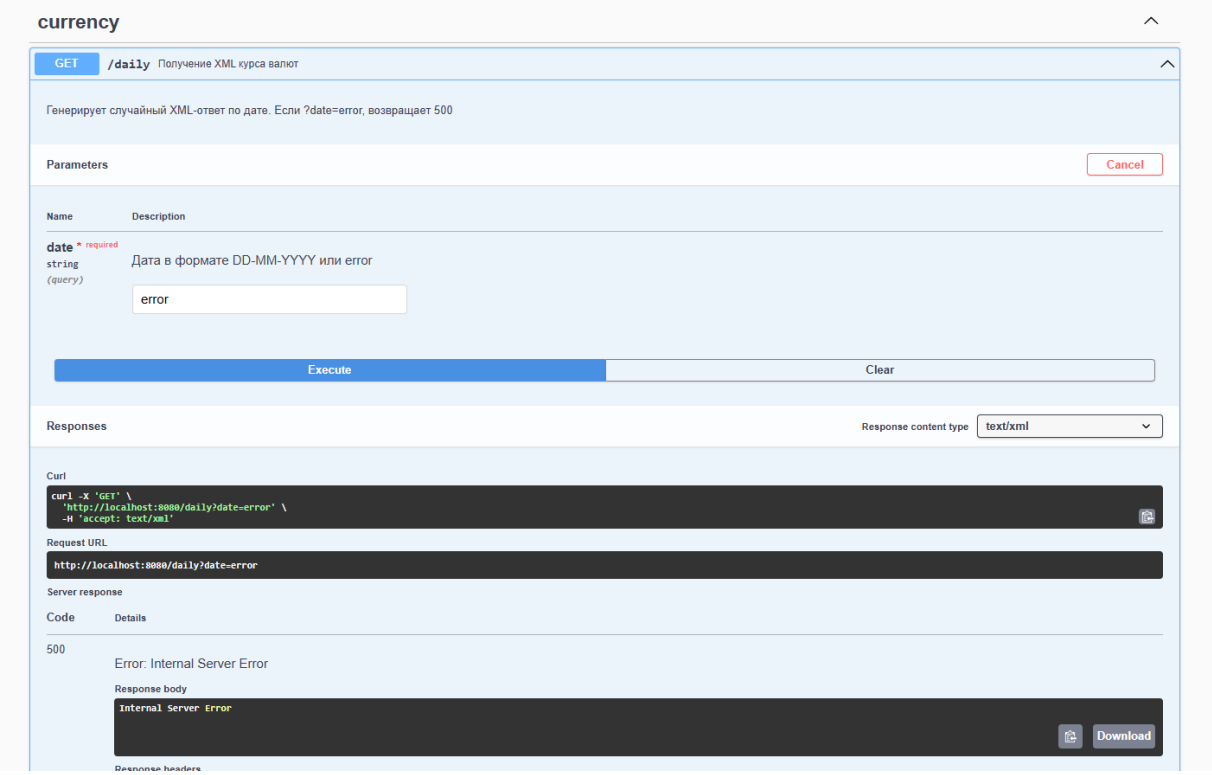

Рисунок 4 — Демонстрация кода 500 в Swagger

4) Реализация основного функционала

На рисунках 5-6 показан вызов GET с передачей значения /daily?date=20-07-2025 через Postman или error. В первом случае сервис возвращает валидный XML-ответ с текущими курсами валют. Данные генерируются динамически при каждом вызове. Во втором случае в качестве значения параметра date указать error, сервис возвращает код 500 Internal Server Error, имитируя отказ.

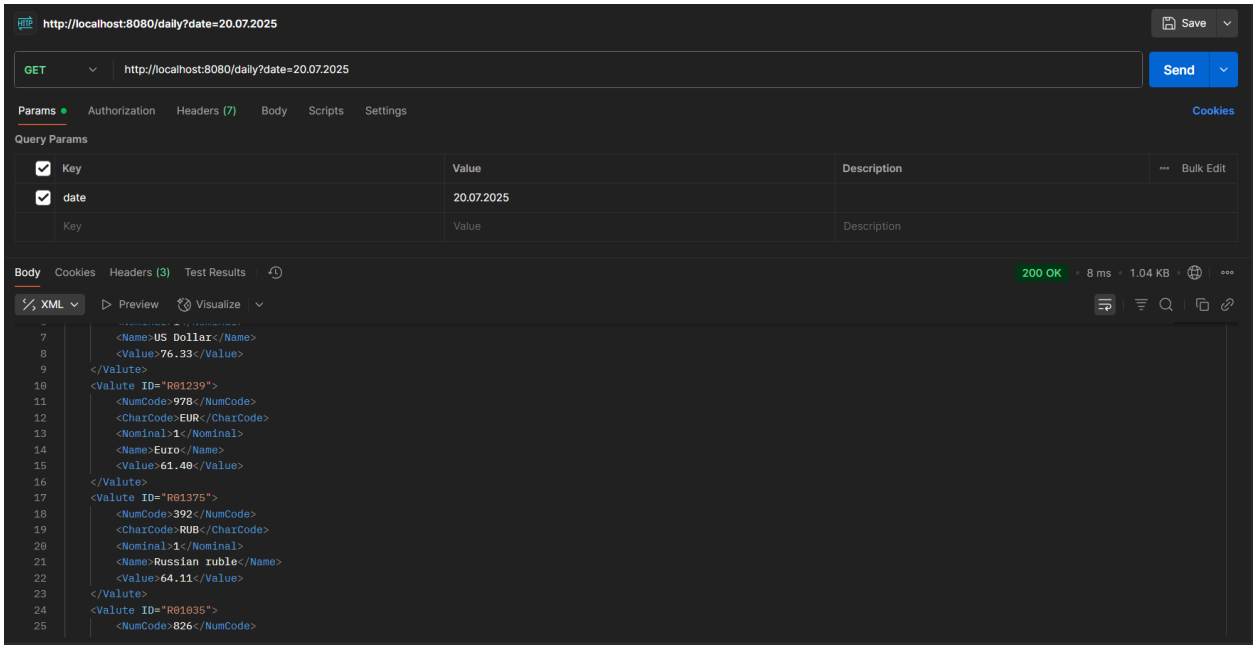

Рисунок 5 — Демонстрация успешного запроса

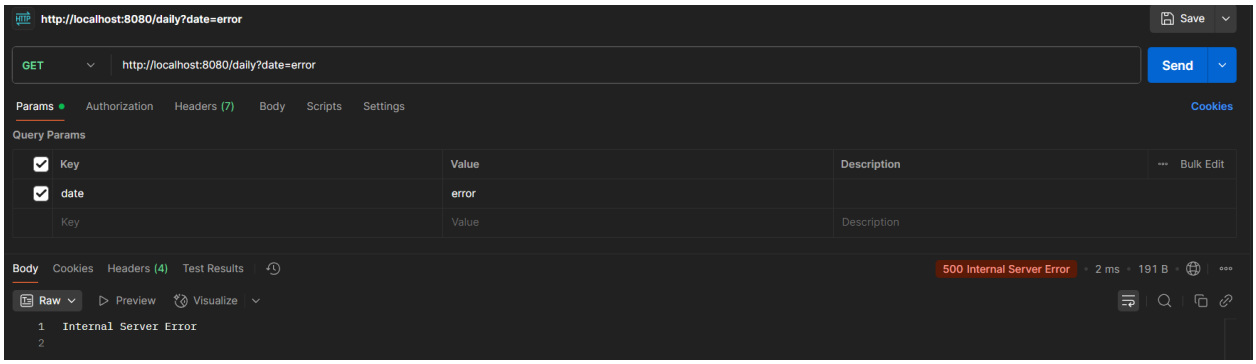

Рисунок 6 — Демонстрация не успешного запроса

5) gRPC

Была проделана работа по созданию gRPC. Использование метода GetDaily с параметром {"date": "21-07-2025"} позволяет получить нам строку с XML и структуру valutes, которая показывает массив валют с номерами, названиями и курсами. Данный метод удобен для автотестов т.к. можно сравнивать конкретные поля без ручного разбора XML. Также при передаче параметра error получим ошибку. Демонстрация работы gRPC показана на рисунках 7 и 8.

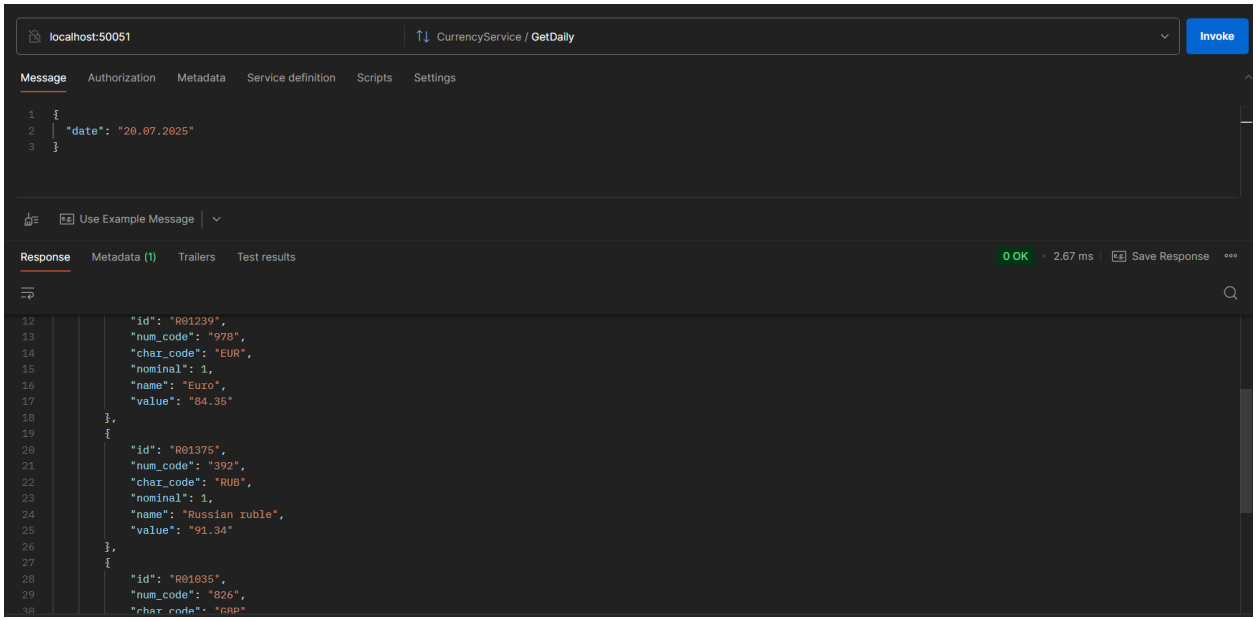

Рисунок 5 — Демонстрация получения данных

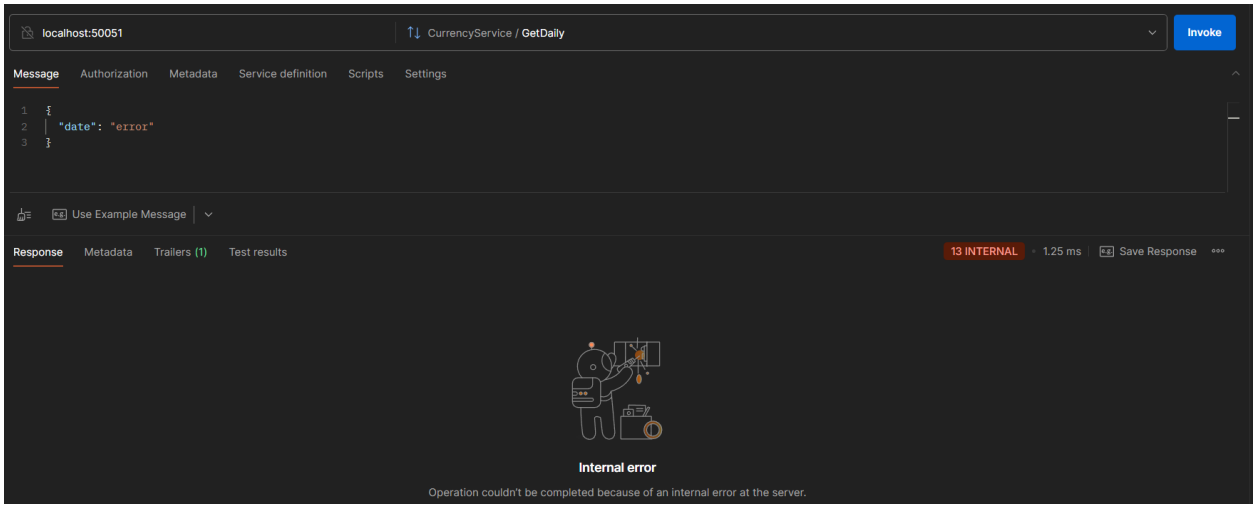

Рисунок 5 — Демонстрация ошибки

6) Вывод

В процессе выполнения тестового задания был создан микросервис на языке Golang, который имитирует работу публичного API Центрального Банка России, предоставляющего информацию о курсах валют. Сервис разработан с нуля и включает в себя:

- REST API с генерацией уникальных XML-ответов;
- gRPC-интерфейс с возвратом как XML-строки, так и структурированных валютных данных;
- Поддержка позитивных и негативных сценариев (успешный ответ и ошибка);
- Использование SQLite для сохранения истории запросов;
- Автогенерация Swagger-документации;
- Запуск серверов (REST + gRPC) одновременно;
- Makefile для управления проектом.

Структура проекта соответствует микросервисному подходу и предполагает лёгкую масштабируемость.