Отчёт по второму заданию

1. Было прочитано и реализовано задание
2. Используемые технологии и их обоснование

2.1. Серверная часть: Python + Flask

Почему выбрано:

- **Простота и скорость разработки**: Flask - минималистичный фреймворк, позволяющий быстро создать REST API
- **Легкость обучения**: Простой синтаксис, понятная структура приложения
- **Гибкость**: легко расширяется дополнительными модулями
- **Встроенная поддержка JSON**: Удобная работа с форматом данных, требуемым по ТЗ

2.2. Клиентская часть: Мультиплатформенный подход

Python-клиент (client.py):

- **Автоматизация тестирования**: Возможность интеграции в CI/CD конвейеры
- **Простота отладки**: легко добавлять логирование и диагностику
- **Кроссплатформенность**: работает на Windows, Linux, macOS

Веб-клиент (HTML/JavaScript):

- **Наглядность**: Визуальное представление данных
- **Доступность**: работает в любом современном браузере
- **Интерактивность**: Удобный пользовательский интерфейс для ручного тестирования

2.3. Формат данных: JSON

Почему выбрано:

- **Стандартизация**: Универсальный формат для веб-API
- **Человекочитаемость**: легко читать и отлаживать
- **Легковесность**: Минимальный оверхед при передаче
- **Широкая поддержка**: Встроенная поддержка в Python и JavaScript

2.4. Архитектура: Модульный подход

- **Разделение ответственности**: Сервер ↔ Клиент ↔ Представление
- **Масштабируемость**: легко добавлять новые endpoints и функции

- **Поддержка CORS**: Возможность интеграции с внешними системами
- **Хранение данных**: JSON-файл вместо базы данных для упрощения

3. Реализованная функциональность

3.1. Серверный модуль (API):

- **GET /api/books** - получение списка всех книг
- **GET /api/books/{id}** - получение конкретной книги
- **POST /api/books** - добавление новой книги
- **PUT /api/books/{id}** - обновление данных книги
- **DELETE /api/books/{id}** - удаление книги
- **GET /api/status** - проверка статуса API

3.2. Клиентские модули:

- **Python-клиент**: Автоматизированное тестирование API
- **Веб-интерфейс**: Интерактивное управление через браузер
- **Диагностические инструменты**: Проверка доступности, логирование

3.3. Обработка ошибок:

- **HTTP 400**: Некорректный запрос (невалидный JSON, отсутствие полей)
- **HTTP 404**: Ресурс не найден
- **HTTP 409**: Конфликт (например, книга уже существует)
- **HTTP 500**: Внутренняя ошибка сервера

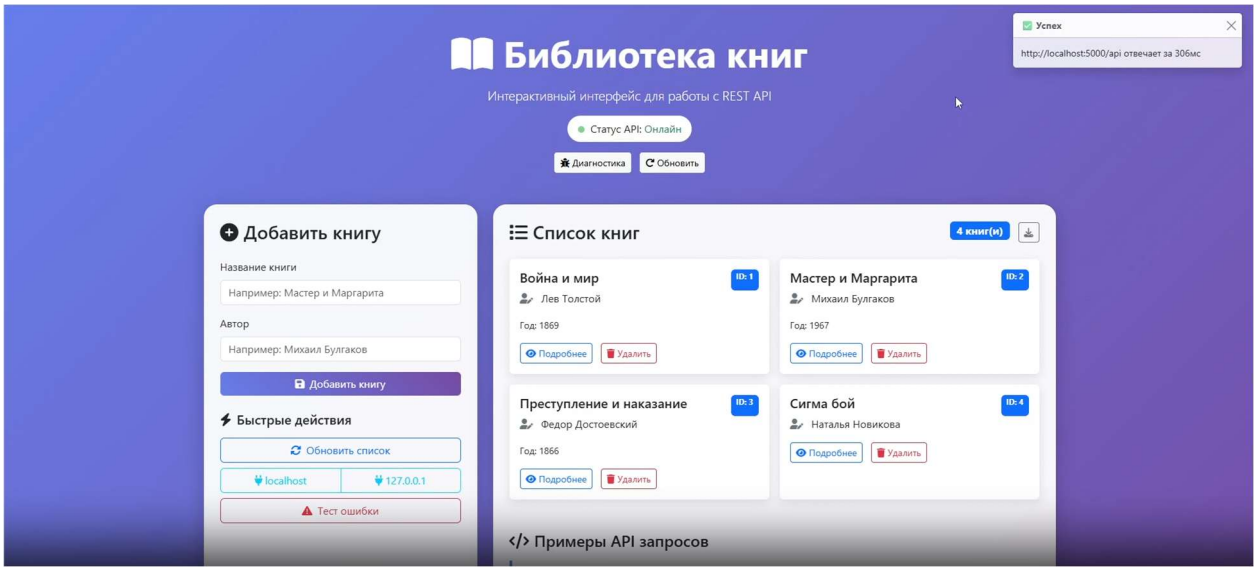

Рис. 1 – главный экран

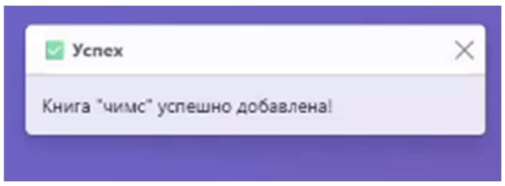

Рис. 2 – успешное добавление

Рис. 3 – успешно проводится диагностика

Рис. 4 – Примеры Api запросов