

Министерство образования и науки РФ
ФГБОУ ВПО «Омский государственный технический университет»
Кафедра «Прикладная математика и фундаментальная информатика»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине «Алгоритмизация и программирование»

Выполнил студент гр. ФИТ-222
Чепурко Артём
Проверил
ст. преподаватель Федотова И.В.

Омск 2023

Содержание

Теоретическая часть по фреймворку Oat++.....	3
Теоретическая часть по базе данных SQLite.....	4
Теоретическая часть по Postman.....	5
Теоретическая часть по Swagger UI.....	6
Скриншоты запросов в Swagger UI.....	7
Скриншоты запросов в Postman.....	8
Приложение с кодом API-контроллера.....	11

Теоретическая часть по фреймворку Oat++

Фреймворк Oat++ предназначен для написания Backend приложения на языке C++. предоставляет разработчикам широкий набор инструментов для реализации различных функций веб-приложений: маршрутизацию запросов, обработку HTTP-запросов и ответов, работу с базами данных и многое другое. Кроме того, фреймворк имеет интегрированный набор инструментов для разработки и отладки приложений, что значительно упрощает процесс разработки и сокращает время на развертывание приложения в продакшене.

Одной из особенностей oat++ является его модульная архитектура, позволяющая использовать только те компоненты, которые необходимы для конкретного проекта. Благодаря этому, разработчики могут создавать масштабируемые и гибкие приложения, которые легко адаптируются к изменяющимся потребностям пользователей.

Теоретическая часть по базе данных SQLite

SQLite является самым простым вариантом реализации базы данных. Он поддерживает большинство функций и команд, которые доступны в других реляционных базах данных, таких как MySQL и PostgreSQL. Он позволяет хранить данные в таблицах, которые могут быть связаны друг с другом посредством внешних ключей. SQLite также поддерживает индексирование данных, что ускоряет процесс поиска и сортировки.

SQLite может быть использован для различных целей, включая хранение данных приложения, кэширование данных, хранение данных сенсоров и многое другое. Он также может быть использован в сочетании с другими технологиями, такими как JSON, для упрощения обмена данными между приложениями.

Теоретическая часть по Postman

Postman - это инструмент для тестирования API и разработки программного обеспечения, который облегчает создание, отправку, тестирование и отладку HTTP-запросов. Postman позволяет разработчикам быстро и удобно проверять работоспособность API и взаимодействовать с ними без необходимости написания собственных скриптов или использования других инструментов.

Postman имеет простой и интуитивно понятный интерфейс, который позволяет создавать и редактировать запросы, устанавливать параметры запросов и просматривать ответы от серверов. Он также поддерживает автоматическую генерацию кода для разных языков программирования, что упрощает интеграцию с различными приложениями.

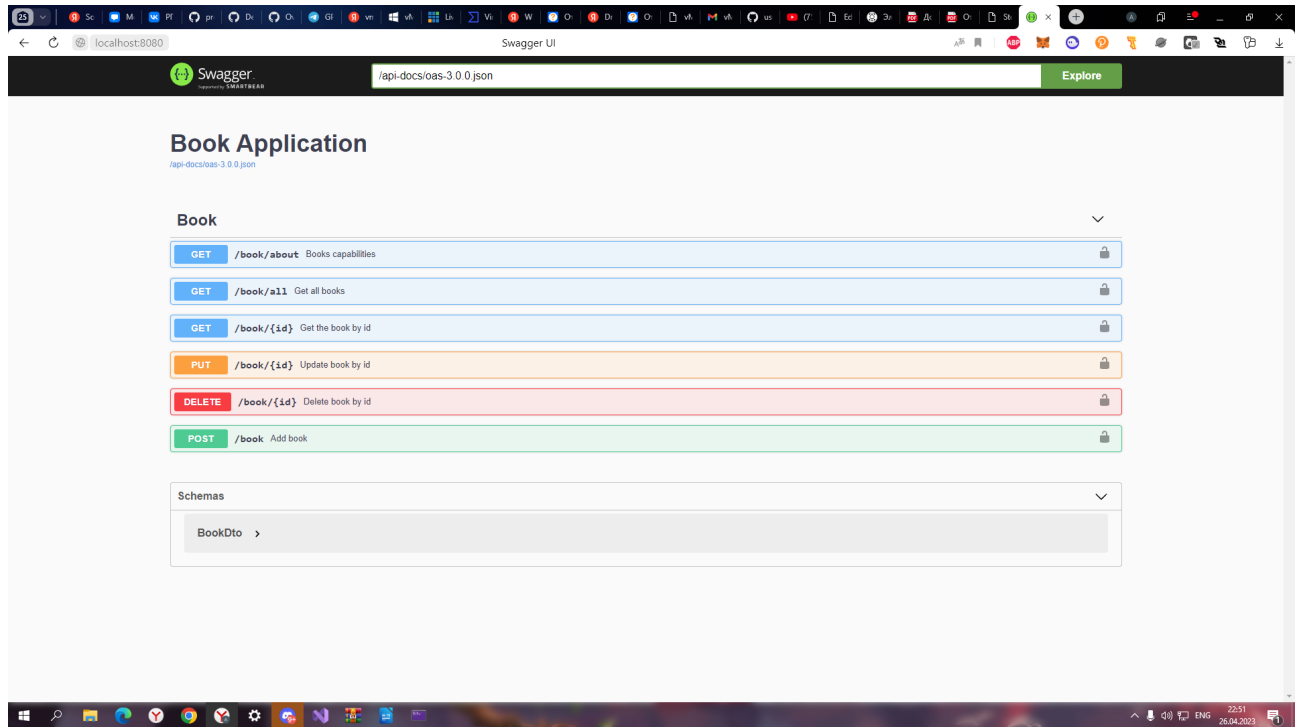
Postman также имеет множество функций, которые помогают улучшить процесс разработки и тестирования, такие как сохранение истории запросов, создание коллекций запросов и переменных окружения, использование коллаборации и многое другое. Он также позволяет тестировать авторизацию и безопасность API, включая тестирование JWT-токенов и проверку SSL-сертификатов.

Теоретическая часть по Swagger UI

Swagger UI предоставляет веб-интерфейс для просмотра документации по API, включая информацию о конечных точках, параметрах, типах данных и примерах запросов и ответов. Swagger UI использует спецификацию OpenAPI (ранее известную как Swagger), которая описывает работу API и позволяет генерировать документацию и код для разных языков программирования.

Он также позволяет тестировать API непосредственно из интерфейса Swagger UI, что делает процесс разработки и отладки API более простым и удобным. Swagger UI поддерживает автоматическое обновление документации при изменении API, что облегчает поддержку и развитие API.

Скриншоты запросов в Swagger UI



Скриншоты запросов в Postman

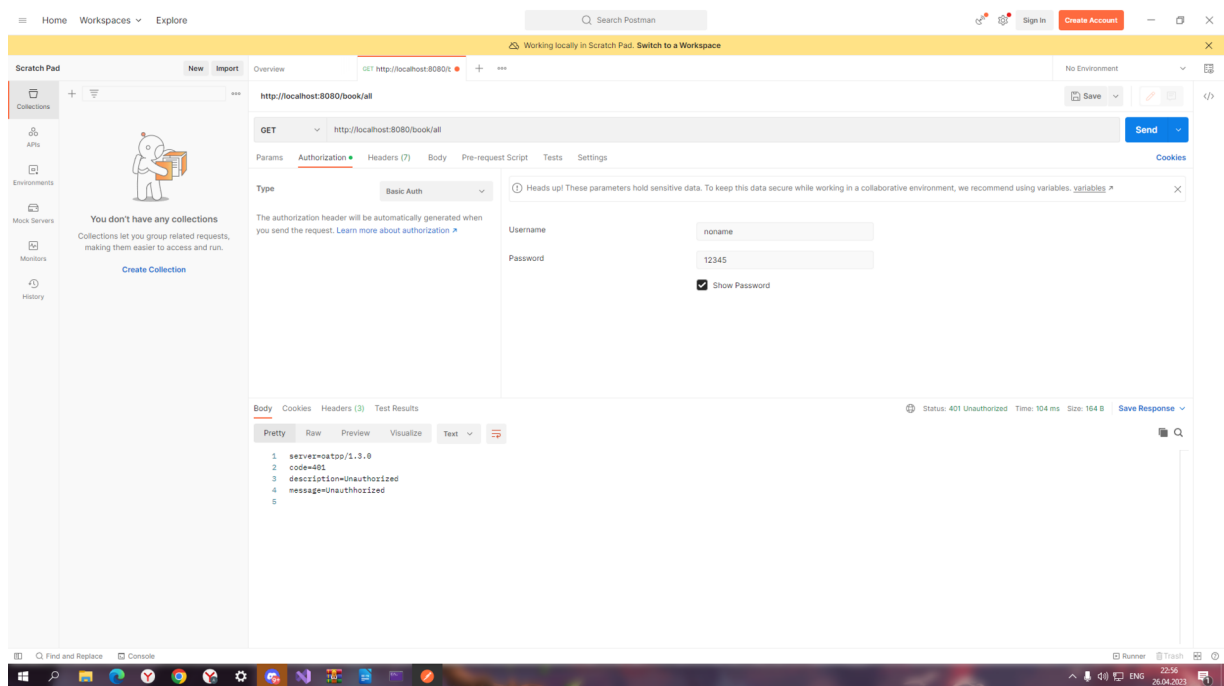


Рисунок 1 - Попытка посмотреть все существующие книги незарегистрированным в базе данных пользователем

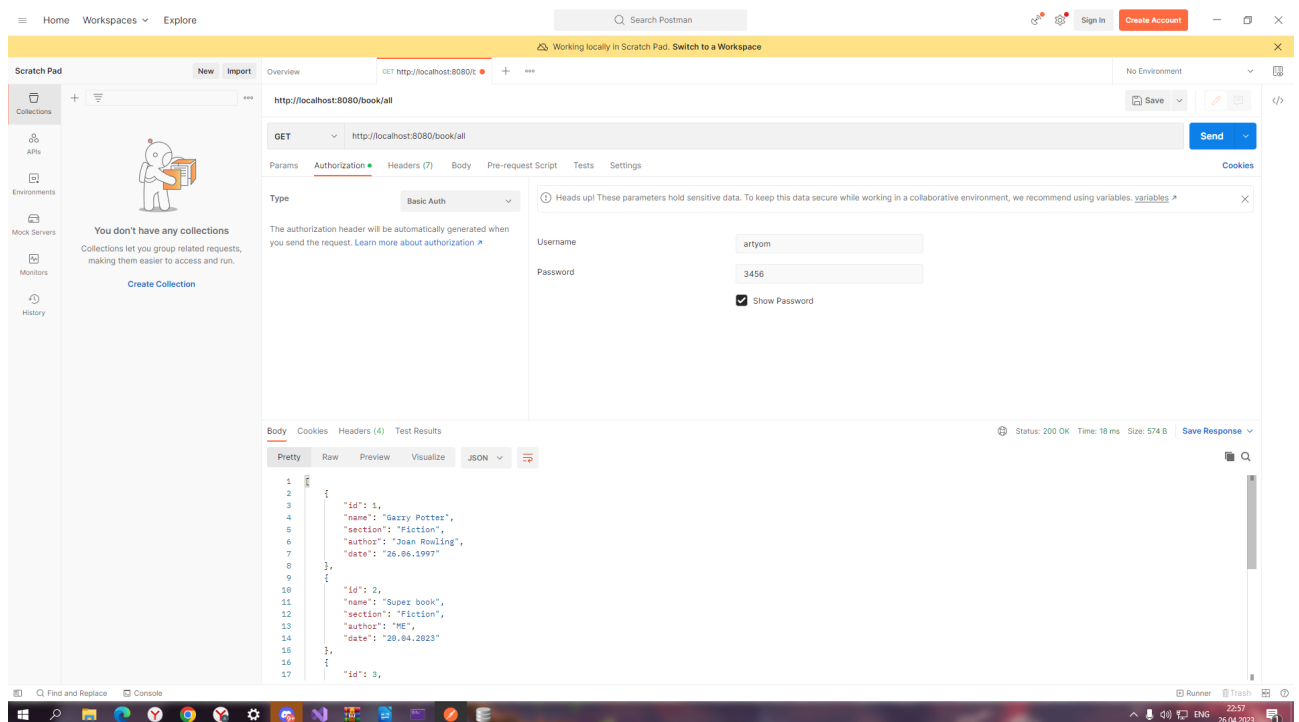


Рисунок 2 - Просмотр всех существующих книг зарегистрированным в базе данных пользователем

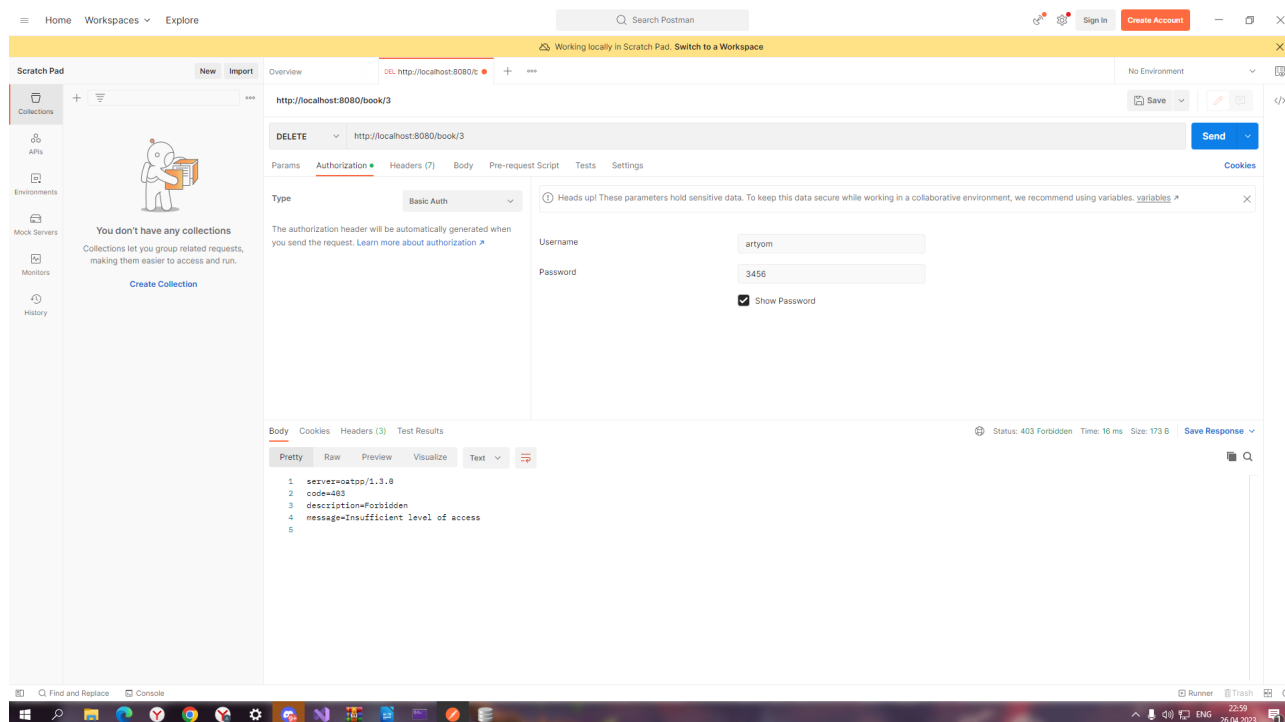


Рисунок 3 — Попытка удаления книги из базы данных пользователем, с низкими правами доступа

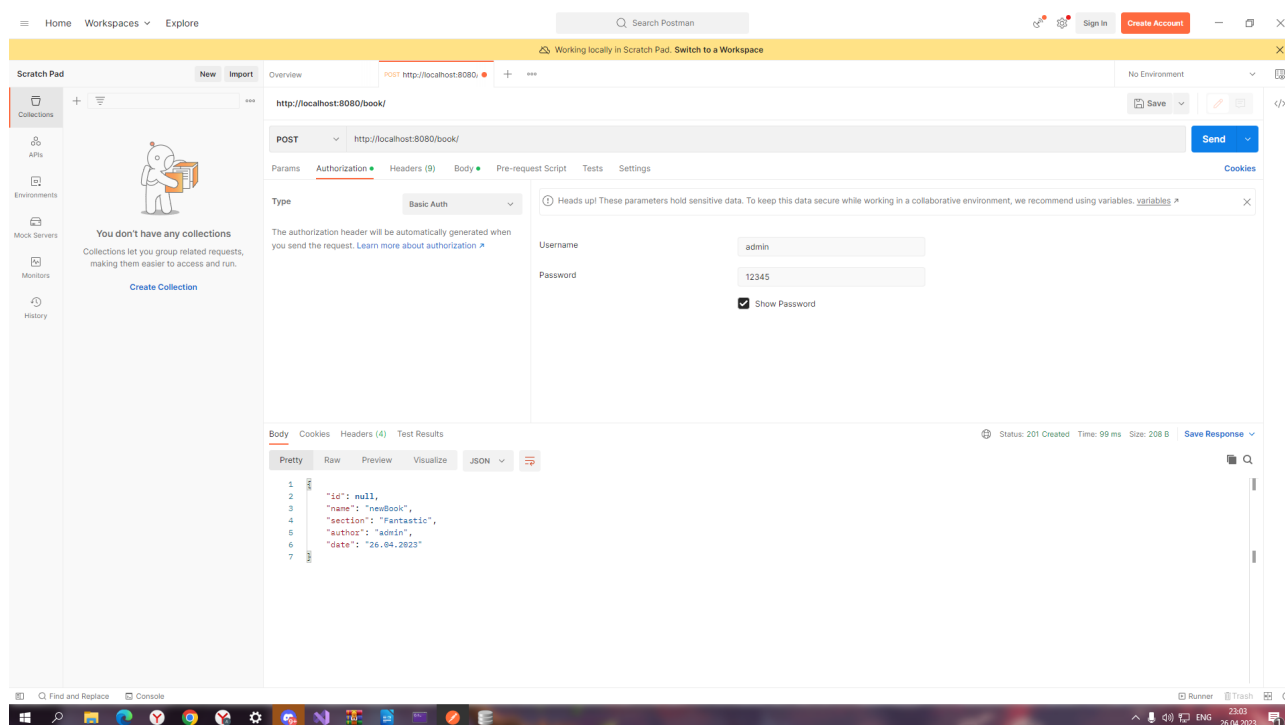


Рисунок 4 — Добавление в базу данных книги пользователем, с высокими правами доступа

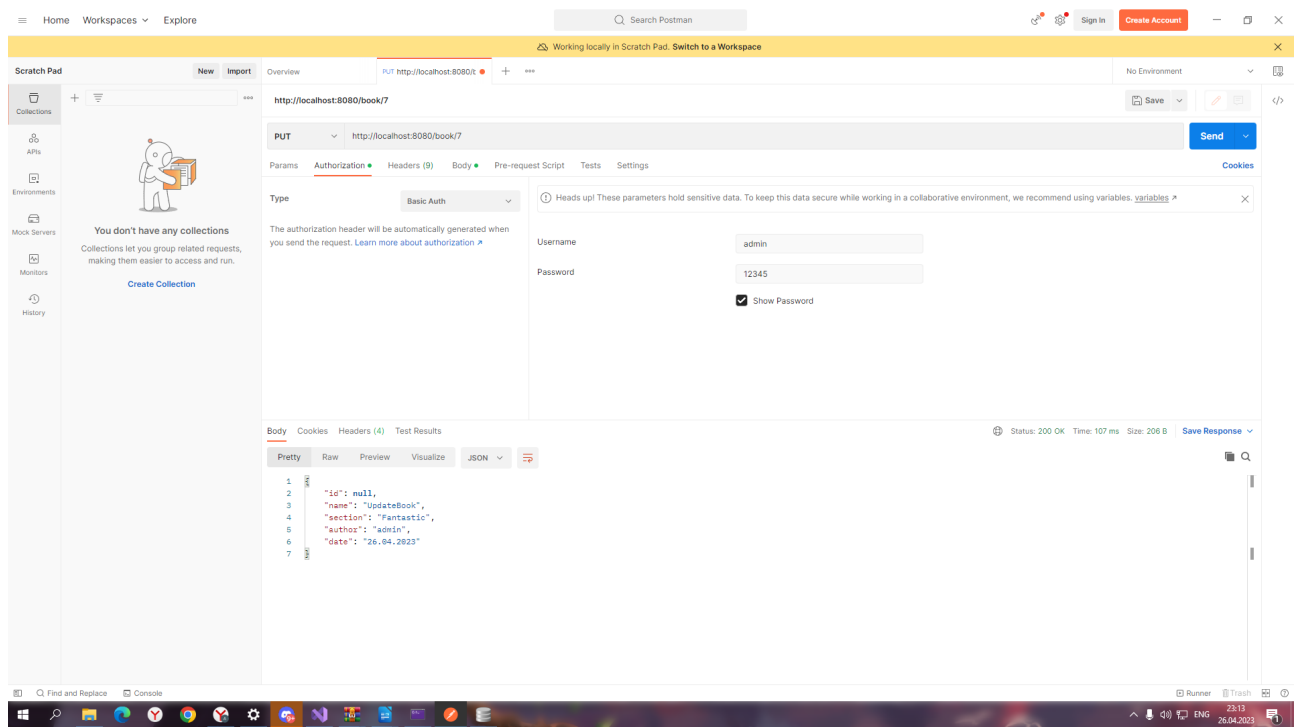


Рисунок 5 — Обновление книги в базе данных пользователем, с высокими правами доступа

Приложение с кодом API-контроллера

```
#include "oatpp/web/server/api/ApiController.hpp"

#include "oatpp/core/macro/codegen.hpp"

#include "oatpp/core/macro/component.hpp"

#include "oatpp/core/Types.hpp"

#include "../dto/ToDoDto.hpp"

#include "../dto/UserDto.hpp"

#include "../dto/ToDoDbDto.hpp"

#include OATPP_CODEGEN_BEGIN(ApiController)

//контроллер для работы с заметками

class BookController :public oatpp::web::server::api::ApiController {
private:

    //получаем ObjectMapper
    OATPP_COMPONENT(std::shared_ptr<oatpp::data::mapping::ObjectMapper>, objectMapper);

    //получаем БД клиента
    OATPP_COMPONENT(std::shared_ptr<MyClient>, dbClient);

    UserDto::Wrapper
    checkAuth(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>
authObject) {

        //получаем всех пользователей

        auto result = dbClient->getAllUsers();

        //преобразуем объект QueryResult в вектор UserDTO

        auto users = result->fetch<oatpp::Vector<oatpp::Object<UserDto>>>>();

        //преобразование QueryResult в JSON

        OATPP_LOGI("", objectMapper->writeToString(users)->c_str());

        //проверка логина и пароля

        for (auto i = 0; i < users->size(); i++) {

            if (users[i]->name == authObject->userId && users[i]->password ==
authObject->password) {

                return users[i];

            }

        }

    }

}
```

```

        //если не совпало, то иди лесом

        throw
oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE_401,
"Unauthorized", {});

    }

    BookDbDto::Wrapper getBook(int id, int userId) {

        //получаем данные о заметках в виде QueryResult
        auto result = dbClient->getOneBookById(id);

        //преобразование в вектор
        auto books = result->fetch<oatpp::Vector<Object<BookDbDto>>>();

        if (books->size() == 0)

            throw
oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE_404, "Заметка
не найдена", {});

        //берем единственную заметку
        auto book = books[0];

        return books[0];

    }

public:

    BookController(OATPP_COMPONENT(std::shared_ptr<ObjectMapper>, objectMapper))

        :oatpp::web::server::api::ApiController(objectMapper)

    {

setDefaultAuthorizationHandler(std::make_shared<oatpp::web::server::handler:
:BasicAuthorizationHandler>("auth-handler"));

    }

    ENDPOINT_INFO(booksAbout) {

        info->tags = std::list<oatpp::String>{"Book"};

        info->summary = "Books capabilities";

        info->addSecurityRequirement("Auth required");

    }

    //о функциях заметок

    ENDPOINT("GET", "/book/about", booksAbout,
AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
authObject)) {

        std::string header = "<h1>Book Controller</h1>";

```

```

        std::string list = "<ul><li>Create books</li><li>Update books</li><li>Delete
books</li><li>Search books</li></ul>";

        return ResponseFactory::createResponse(Status::CODE_200, header + list);
    }

    //получить список всех заметок
    ENDPOINT_INFO(booksAll) {
        info->tags = std::list<oatpp::String>{ "Book" };
        info->summary = "Get all books";
        info->addSecurityRequirement("Auth required");
    }

    ENDPOINT("GET", "/book/all", booksAll,

AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
authObject)) {

        //проверка логина и пароля
        auto userID = checkAuth(authObject);

        //получаем данные о заметках в виде QueryResult
        auto result = dbClient->getAllBook();

        //преобразование в вектор
        auto books = result->fetch<oatpp::Vector<Object<BookDbDto>>>();

        return ResponseFactory::createResponse(Status::CODE_200, books, objectMapper);
    }

    //получить заметку по id
    ENDPOINT_INFO(booksOne) {
        info->tags = std::list<oatpp::String>{ "Book" };
        info->summary = "Get the book by id";
        info->addSecurityRequirement("Auth required");
    }

    ENDPOINT("GET", "/book/{id}", booksOne, PATH(Int16, id),

AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
authObject)) {

        auto userID = checkAuth(authObject);

        auto book = getBook(id, userID->id);

        return ResponseFactory::createResponse(Status::CODE_200, book, objectMapper);
    }

```

```

    }

    //добавление заметки
    ENDPOINT_INFO(booksAdd) {
        info->tags = std::list<oatpp::String>{ "Book" };
        info->summary = "Add book";
        info->addSecurityRequirement("Auth required");
    }

    ENDPOINT("POST", "/book", booksAdd, BODY_DTO(Object<BookDto>, bookDto),
    ),
    AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
    authObject)) {
        auto userID = checkAuth(authObject);
        if (userID->priority == 0) {
            throw
oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE_403,
"Insufficient level of access", {});
        }

        //добавляем DTO в вектор
        dbClient->addBook(bookDto);

        //возвращаем в формате json (201 - объект добавлен)
        return ResponseFactory::createResponse(Status::CODE_201, bookDto,
objectMapper);
    }

    //обновление заметки
    ENDPOINT_INFO(booksUpdate) {
        info->tags = std::list<oatpp::String>{ "Book" };
        info->summary = "Update book by id";
        info->addSecurityRequirement("Auth required");
    }

    ENDPOINT("PUT", "/book/{id}", booksUpdate, BODY_DTO(Object<BookDto>, bookDto),
    PATH(Int16, id),

    AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
    authObject)) {
        auto userID = checkAuth(authObject);
        auto book = getBook(id, userID->id);
        if (userID->priority == 0) {

```

```

        throw
oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE_403,
    "Insufficient level of access", {}));

    }

    dbClient->updateBook(bookDto, id);

    return ResponseFactory::createResponse(Status::CODE_200, bookDto,
objectMapper);
}

//удаление заметки
ENDPOINT_INFO(booksDelete) {
    info->tags = std::list<oatpp::String>{ "Book" };
    info->summary = "Delete book by id";
    info->addSecurityRequirement("Auth required");
}

ENDPOINT("DELETE", "/book/{id}", booksDelete, PATH(Int16, id),

AUTHORIZATION(std::shared_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>,
authObject)) {
    auto userID = checkAuth(authObject);

    //если такого нет, то 404
    auto book = getBook(id, userID->id);

    if (userID->priority == 0) {

        throw
oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE_403,
    "Insufficient level of access", {}));

    }

    //удаление
    dbClient->deleteBook(id);

    //возвращаем в формате json (204 - пустой ответ)
    return ResponseFactory::createResponse(Status::CODE_204, "");
}

};

#include OATPP_CODEGEN_END(ApiController)

```