#### 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 Сагалбаев Дамир

#### Проверил

#### ст. преподаватель Федотова И.В.

#### Омск 2023

### Содержание

[Теоретическая часть по фреймворку Oat++ 3](#_TOC_250006)

[Теоретическая часть по базе данных SQLite 4](#_TOC_250005)

[Теоретическая часть по Postman 5](#_TOC_250004)

[Теоретическая часть по Swagger UI 6](#_TOC_250003)

[Скриншоты запросов в Swagger UI 7](#_TOC_250002)

[Скриншоты запросов в Postman 8](#_TOC_250001)

[Приложение с кодом API-контроллера 11](#_TOC_250000)

# 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

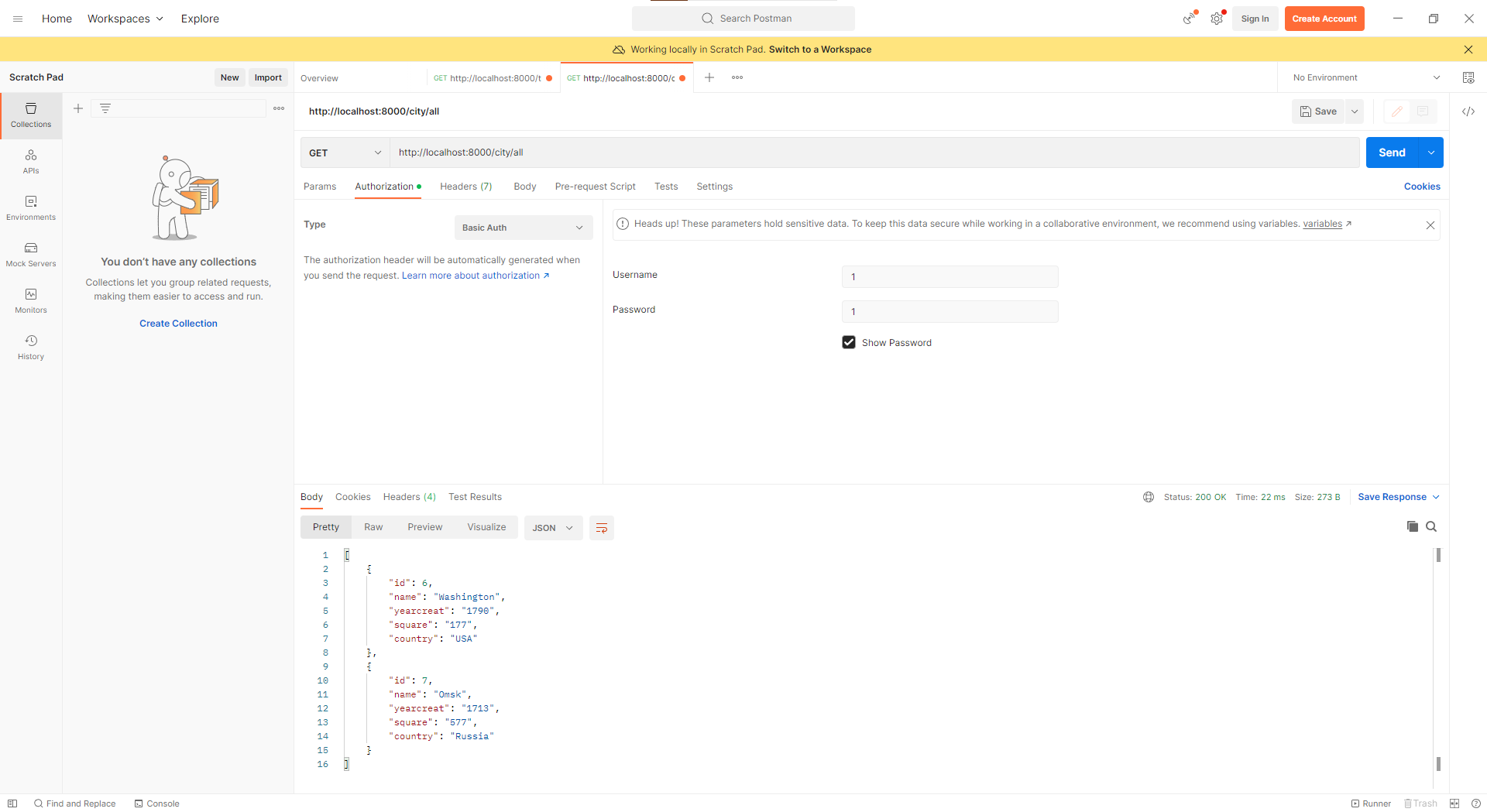
#### Swagger UI предоставляет веб-интерфейс для просмотра документации по API, включая информацию о конечных точках, параметрах, типах данных и примерах запросов и ответов. Swagger UI использует спецификацию OpenAPI (ранее известную как Swagger), которая описывает работу API и позволяет генерировать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

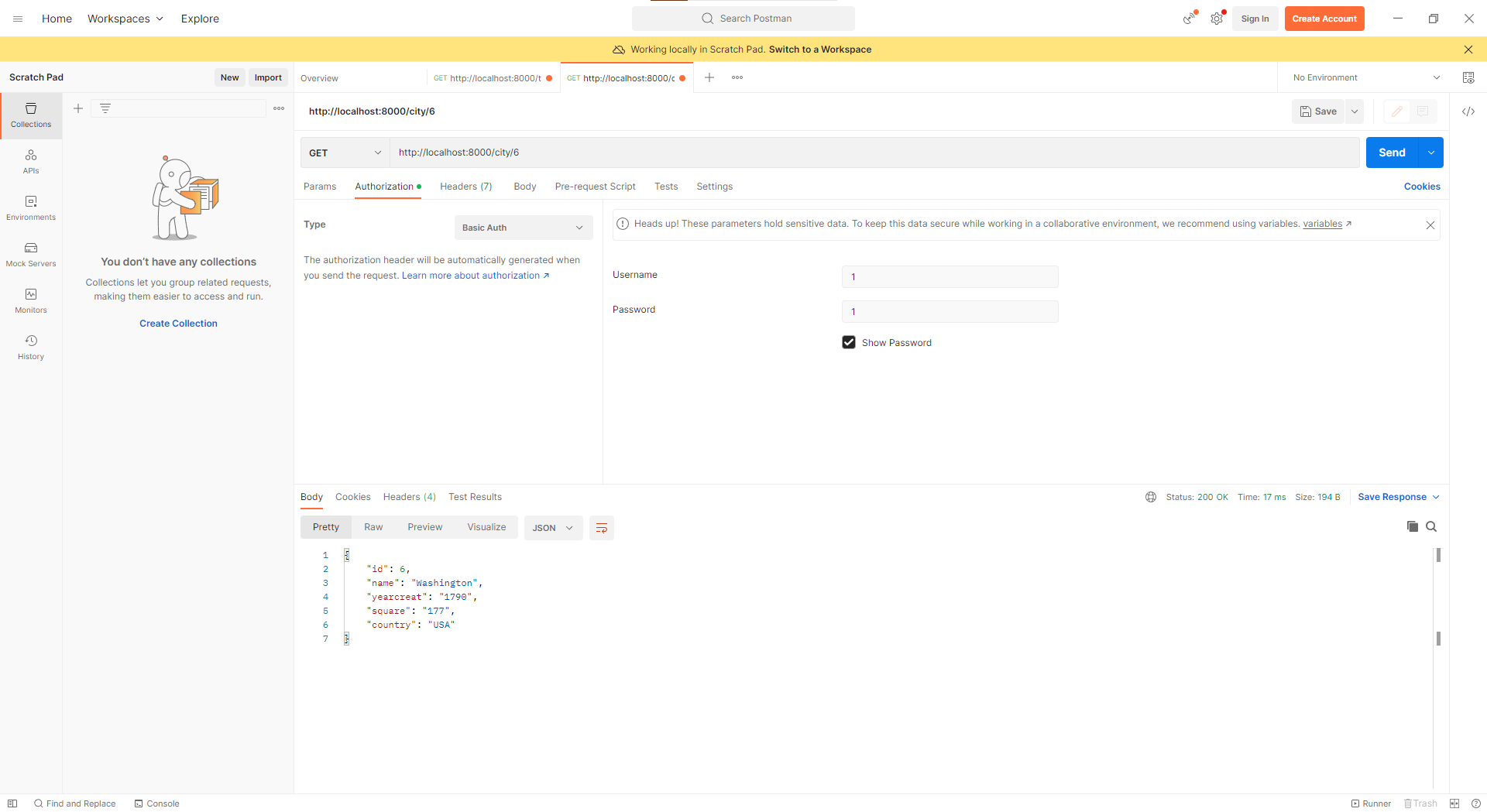
# Скриншоты запросов в Swagger UI

# 

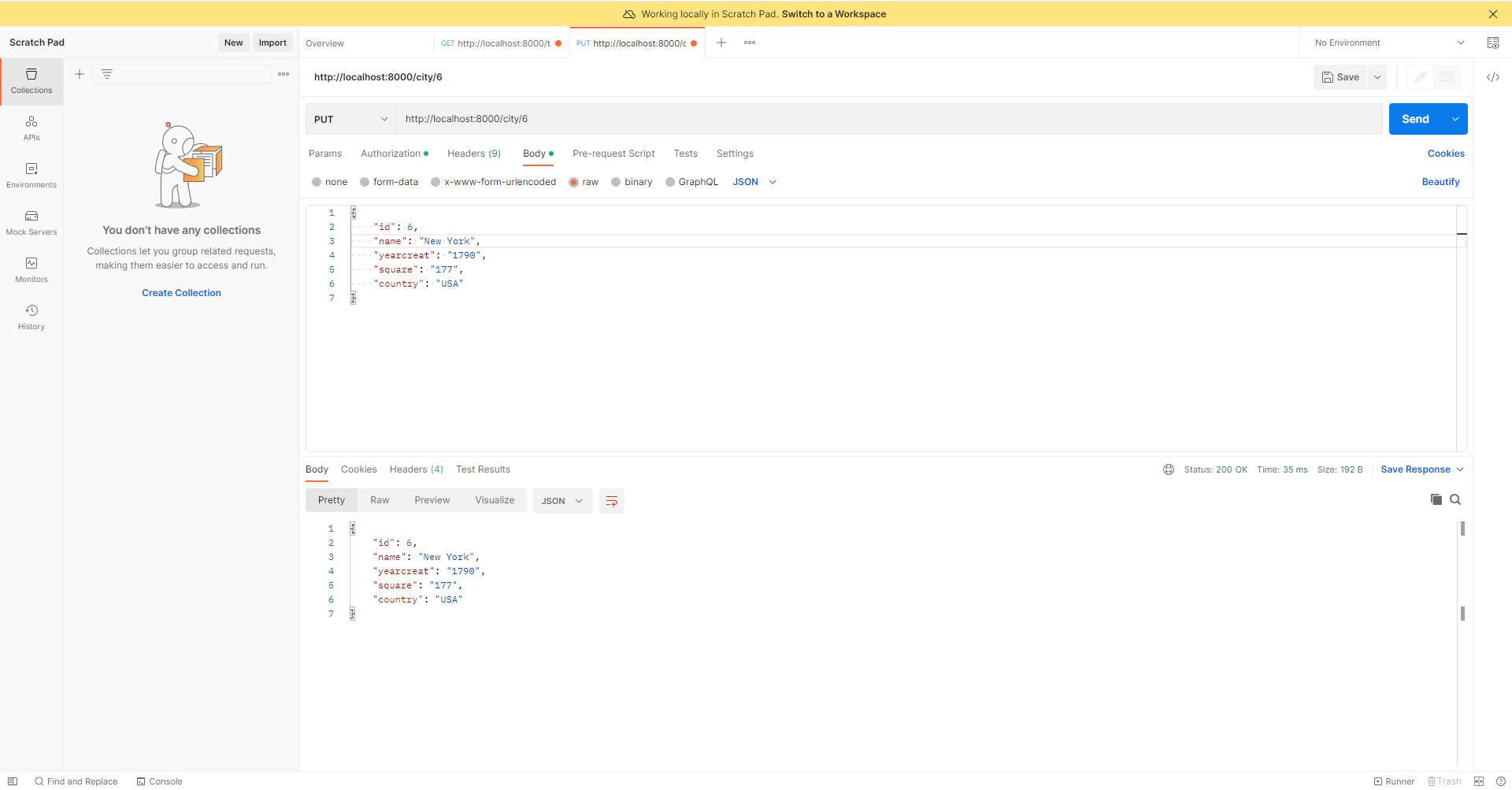
# Скриншоты запросов в Postman



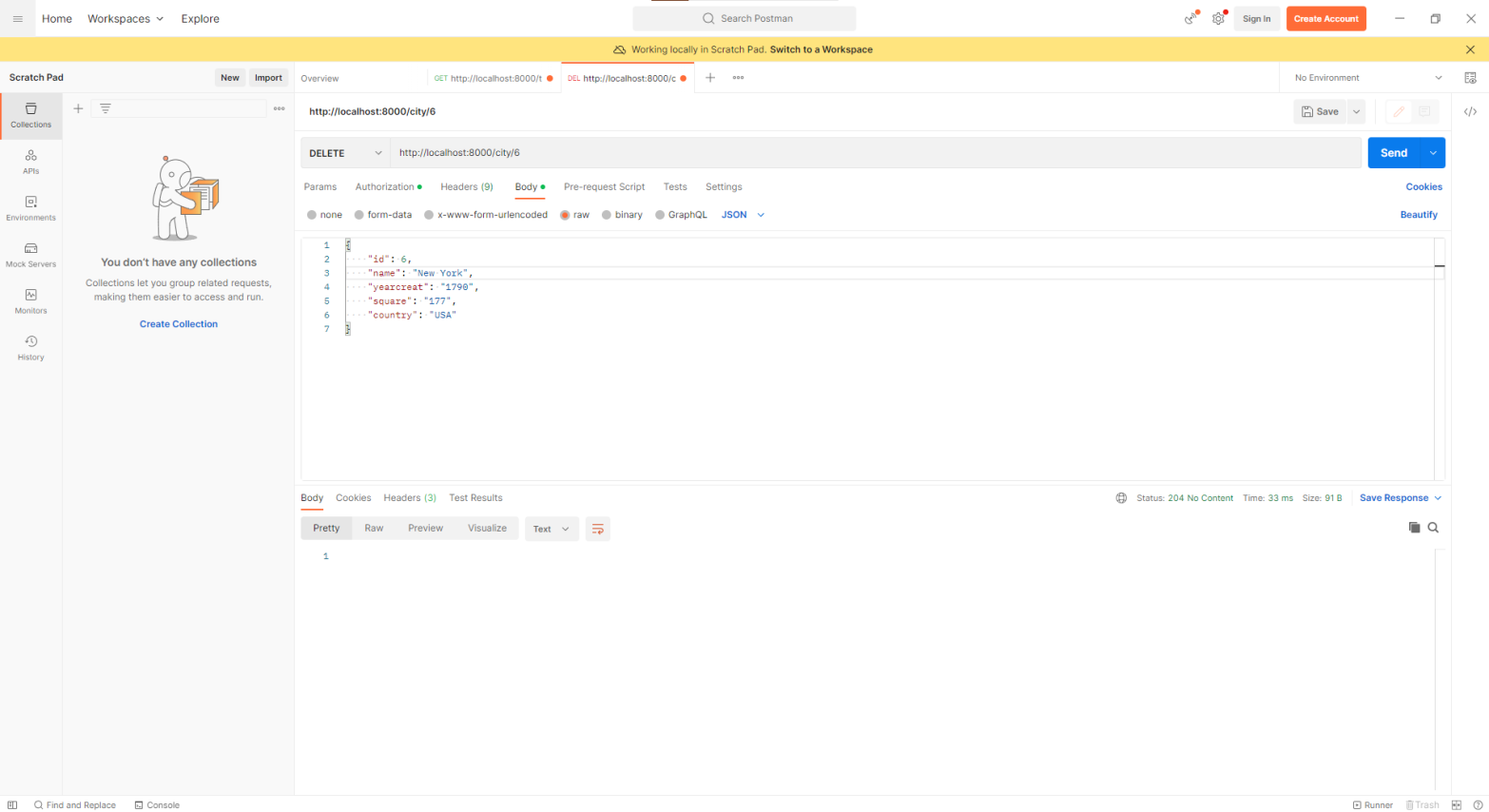
*Рисунок 1. Все города(логин 1, пароль 1)*



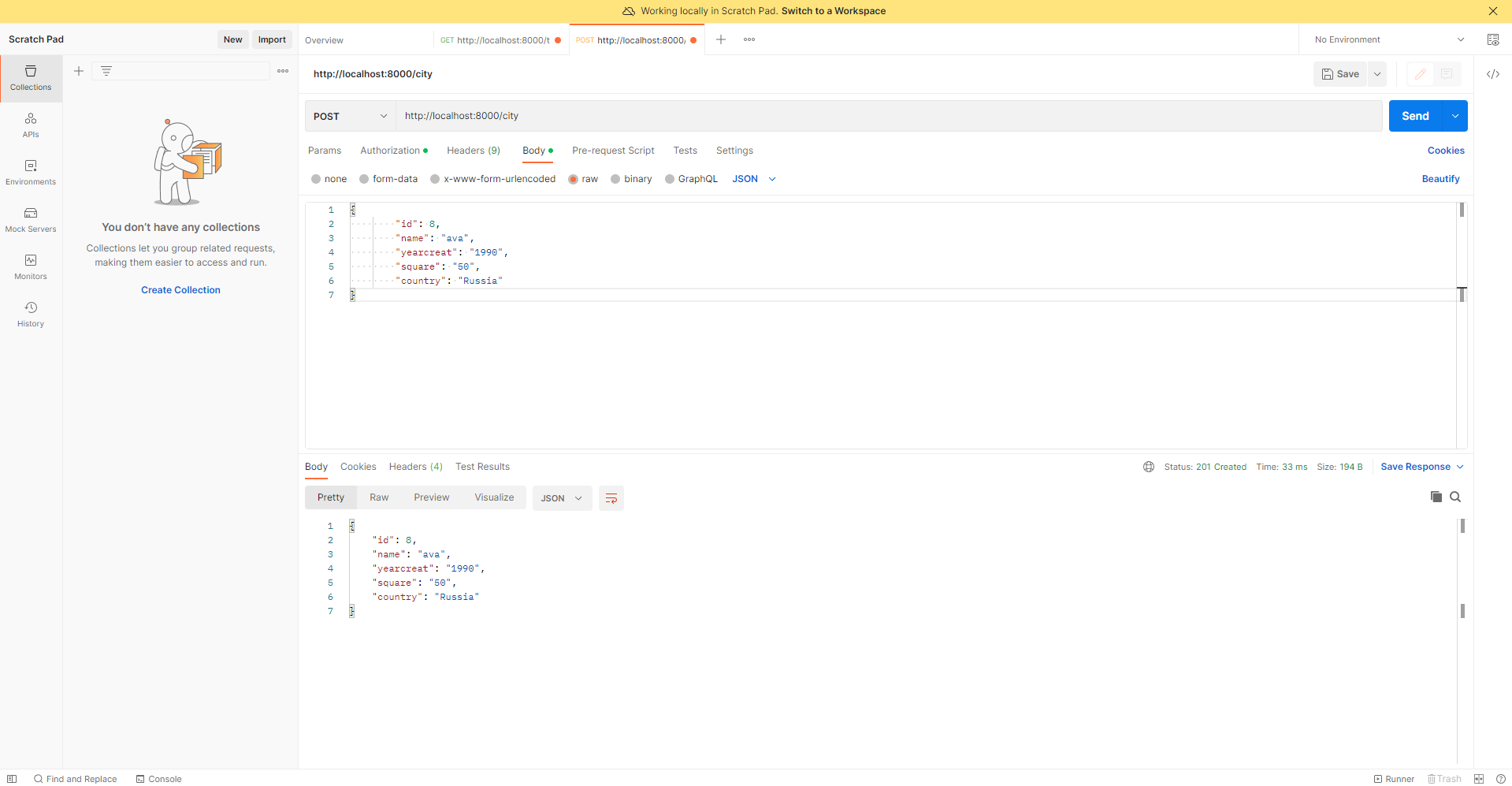
*Рисунок 2. Город по id*

**

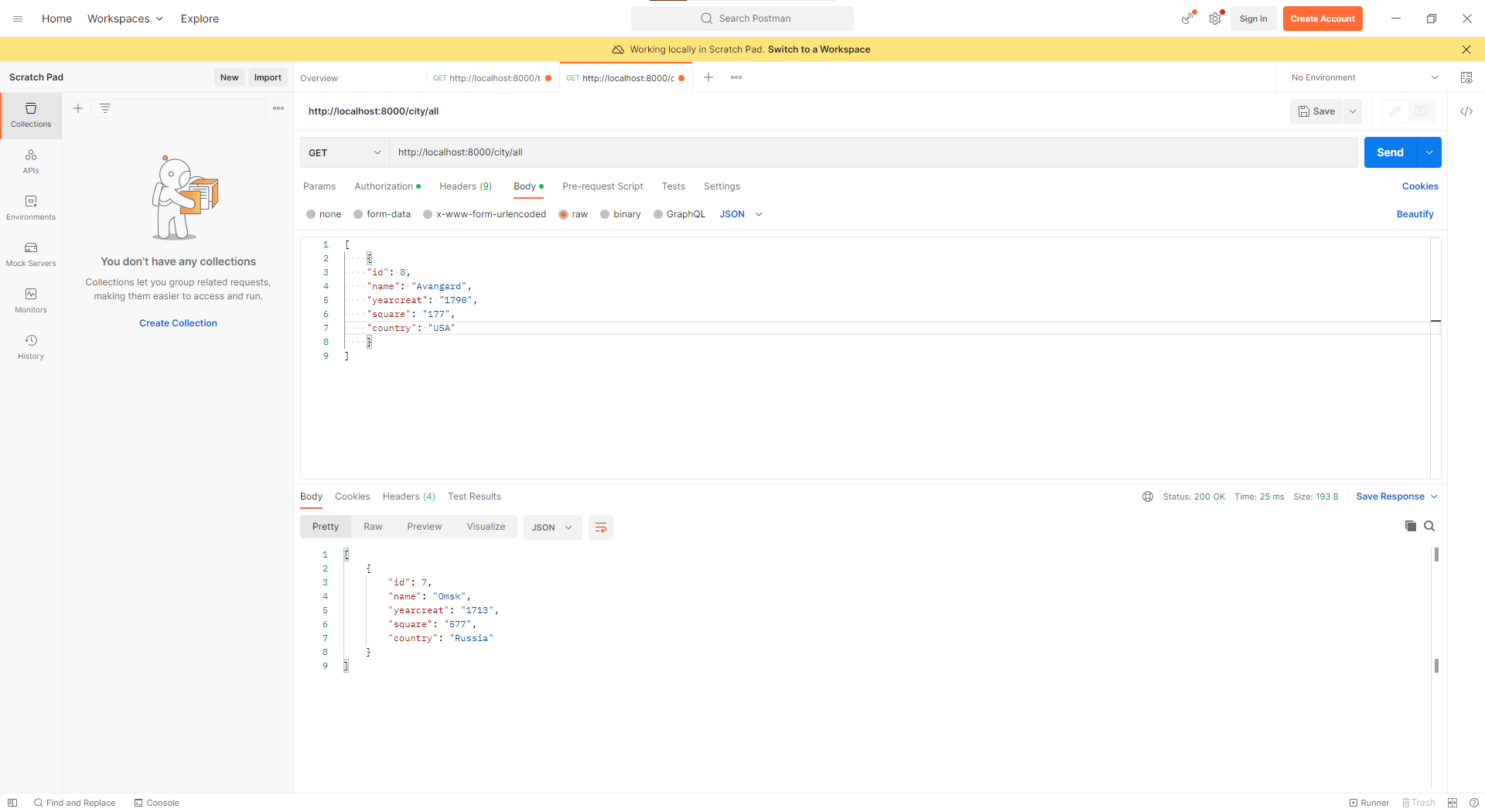
*Рисунок 3. Изменить по id города*

**

*Рисунок 4. Удалить город по id*

**

*Рисунок 5. Добавить город*

**

*Рисунок 6. В итоге(Все города)*

# 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 CityController :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);

//функция для проверки данных авторизации

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, "Unauthhorized", {});

}

CityDbDto::Wrapper getCity(int id, int userId) {

//получаем данные о заметках в виде QueryResult

auto result = dbClient->getOneCityById(id);

//преобразование в вектор

auto cities = result->fetch<oatpp::Vector<Object<CityDbDto>>>();

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

throw oatpp::web::protocol::http::HttpError(oatpp::web::protocol::http::Status::CODE\_404, "Заметка не найдена", {});

//берем единственную заметку

auto city = cities[0];

return cities[0];

}

public:

CityController(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(citiesAbout) {

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

info->summary = "City abillities";

info->addSecurityRequirement("Auth required");

}

//о функциях заметок

ENDPOINT("GET", "/city/about", citiesAbout, AUTHORIZATION(std::shared\_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>, authObject)) {

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

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

return ResponseFactory::createResponse(Status::CODE\_200, header + list);

}

//получить список всех заметок

ENDPOINT\_INFO(citiesAll) {

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

info->summary = "Get all cities";

info->addSecurityRequirement("Auth required");

}

ENDPOINT("GET", "/city/all", citiesAll,

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->getAllCity();

//преобразование в вектор

auto cities = result->fetch<oatpp::Vector<Object<CityDbDto>>>();

return ResponseFactory::createResponse(Status::CODE\_200, cities, objectMapper);

}

//получить заметку по id

ENDPOINT\_INFO(citiesOne) {

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

info->summary = "Get the city by id";

info->addSecurityRequirement("Auth required");

}

ENDPOINT("GET", "/city/{id}", citiesOne, PATH(Int16, id),

AUTHORIZATION(std::shared\_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>, authObject)) {

auto userID = checkAuth(authObject);

auto city = getCity(id, userID->id);

return ResponseFactory::createResponse(Status::CODE\_200, city, objectMapper);

}

//добавление заметки

ENDPOINT\_INFO(citiesAdd) {

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

info->summary = "Add City";

info->addSecurityRequirement("Auth required");

}

ENDPOINT("POST", "/city", citiesAdd, BODY\_DTO(Object<CityDto>, cityDto,

), 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->addCity(cityDto);

//возвращаем в формате json (201 - объект добавлен)

return ResponseFactory::createResponse(Status::CODE\_201, cityDto, objectMapper);

}

//обновление заметки

ENDPOINT\_INFO(citiesUpdate) {

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

info->summary = "Update city by id";

info->addSecurityRequirement("Auth required");

}

ENDPOINT("PUT", "/city/{id}", citiesUpdate, BODY\_DTO(Object<CityDto>, cityDto), PATH(Int16, id),

AUTHORIZATION(std::shared\_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>, authObject)) {

auto userID = checkAuth(authObject);

auto city = getCity(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->updateCity(cityDto, id);

return ResponseFactory::createResponse(Status::CODE\_200, cityDto, objectMapper);

}

//удаление заметки

ENDPOINT\_INFO(citiesDelete) {

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

info->summary = "Delete city by id";

info->addSecurityRequirement("Auth required");

}

ENDPOINT("DELETE", "/city/{id}", citiesDelete, PATH(Int16, id),

AUTHORIZATION(std::shared\_ptr<oatpp::web::server::handler::DefaultBasicAuthorizationObject>, authObject)) {

auto userID = checkAuth(authObject);

//если такого нет, то 404

auto city = getCity(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->deleteCity(id);

//возвращаем в формате json (204 - пустой ответ)

return ResponseFactory::createResponse(Status::CODE\_204, "");

}

};

#include OATPP\_CODEGEN\_END(ApiController)