**1. Создайте файл data.js** и вставьте в него объекты users и orders, используя синтаксис ES6 для экспорта:

const users = {

"264ac85a-770c-4698-81e5-a207424158cb": {

familyName: "Иванов",

middleName: "Иван",

email: "ivan.ivanov@example.com",

registrationDate: "2023-01-15"

},

"b408f257-507d-4bda-9890-e4ce2e779b02": {

familyName: "Петров",

middleName: "Петр",

email: "petr.petrov@example.com",

registrationDate: "2023-02-22"

},

"56390375-d5f0-49a6-bbc7-3688376b6f69": {

familyName: "Сидоров",

middleName: "Сидор",

email: "sidor.sidorov@example.com",

registrationDate: "2023-03-10"

}

};

const orders = {

"a1b2c3d4-e5f6-7g8h-9i0j-k1l2m3n4o5p6": [

{

userId: "264ac85a-770c-4698-81e5-a207424158cb",

product: "Ноутбук",

quantity: 1,

orderDate: "2023-03-01"

},

{

userId: "264ac85a-770c-4698-81e5-a207424158cb",

product: "Мышь",

quantity: 2,

orderDate: "2023-03-02"

}

],

"g1h2i3j4-k5l6-m7n8-o9p0-q1r2s3t4u5v6": [

{

userId: "b408f257-507d-4bda-9890-e4ce2e779b02",

product: "Клавиатура",

quantity: 1,

orderDate: "2023-02-25"

}

],

"m1n2o3p4-q5r6-s7t8-u9v0-w1x2y3z4a5b6": [

{

userId: "56390375-d5f0-49a6-bbc7-3688376b6f69",

product: "Монитор",

quantity: 1,

orderDate: "2023-04-05"

},

{

userId: "56390375-d5f0-49a6-bbc7-3688376b6f69",

product: "Сетевой кабель",

quantity: 3,

orderDate: "2023-04-06"

}

]

};

export { users, orders };

**2. Откройте файл server.js и импортируйте** в него объекты:

import { users, orders } from './data.js';

**3. Чтобы использовать id**, переданный в запросе, как ключ для получения данных из объекта users, **вам нужно изменить вашу обработку запроса**, чтобы вместо выполнения запроса к удаленному сервису использовать данные из локального объекта.

app.get('/users/:id', async (req, res) => {

try {

console.log("Обработка запроса на стороне API");

if (req.params) {

console.log("Параметры запроса: ", req.params);

}

const userId = req.params.id; // Получаем id из параметров запроса

const userData = users[userId]; // Ищем данные пользователя по id в объекте users

if (userData) {

// Если пользователь найден, отправляем данные в ответе

res.json(userData);

} else {

// Если пользователь не найден, отправляем 404

res.status(404).send("Пользователь не найден");

}

} catch (error) {

console.log("Возникла ошибка!");

if (error.cause) {

console.log("Текст ошибки: ", error.cause);

}

res.status(500).send("Произошла ошибка на сервере");

}

});

**Объяснение изменений:**

Получение ID: Извлекаем id из параметров запроса с помощью req.params.id.

Поиск пользователя: Ищем данные пользователя в объекте users с помощью users[userId].

Проверка на существование: Если пользователь найден, отправляем его данные в ответе. Если пользователя с указанным id нет, отправляем статус 404 с сообщением "Пользователь не найден".

Обработка ошибок: В случае ошибки мы просто обрабатываем ее как 500, так как ошибка может возникнуть не только из-за отсутствия пользователя.

В реальной ситуации данные пользователей обычно хранятся в базе данных, например, MongoDB.

В типичном приложении API код осуществляет соединение с базой данных и выполняет запрос, чтобы получить данные о пользователе по его id. Например, вместо обращения к локальному объекту users, в реальной реализации код выглядел бы так:

const userData = await UserModel.findById(userId); // Запрос в базу данных

**4.** По аналогии **измените код, который отвечает** за обработку запросов к **'/orders/:id'** API. Вместо объекта users используйте объект **orders**. **Убедитесь, что вы используйте авторизацию на этом API.**

**ИНСТРУКЦИЯ ПО ТЕСТИРОВАНИЮ КОДА**

Для тестирования кода, который обрабатывает запросы к API на локальном сервере, воспользуйтесь Postman.

**1. Запустите сервер**

Откройте терминал или командную строку. Перейдите в директорию проекта, где находится файл server.js. Запустите сервер с помощью команды:

node server.js

Убедитесь, что сервер запущен и слушает HTTP-запросы на http://localhost:PORT, где PORT — это порт, на котором работает ваш сервер (в данном случае 3000).

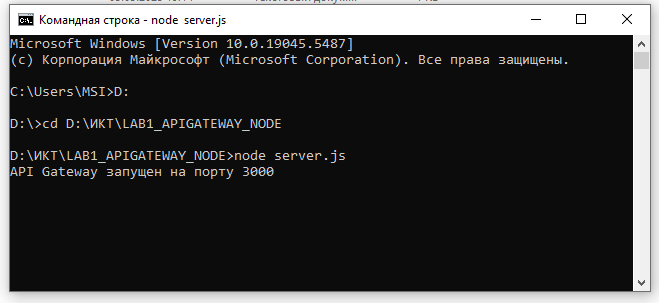


Рисунок 1 – Node.js сервер запущен на порту 3000

**2 Установите и откройте Postman**

Если у вас еще нет Postman, скачайте и установите его с официального сайта (https://www.postman.com/). Откройте Postman.

**4 Создайте новую коллекцию**

В левой панели Postman нажмите на кнопку New (Новый). Выберите Collection (Коллекция). Дайте название коллекции (например, "БГАС Лабораторные") и нажмите Create (Создать).

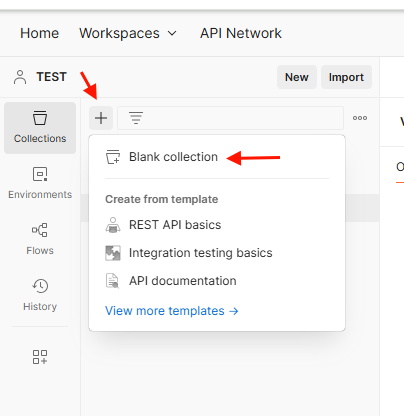


Рисунок 2 – Создание новой коллекции в Postman

**5 Создайте новый запрос в коллекции**

Выберите вашу новую коллекцию. Нажмите на кнопку Add Request (Добавить запрос). Назовите запрос (например, "Get User by ID"). Убедитесь, что тип запроса установлен на GET. В поле URL введите адрес вашего сервера с указанием идентификатора пользователя. Например:

http://localhost:3000/users/56390375-d5f0-49a6-bbc7-3688376b6f69

Сохраните изменения в вашем запросе.

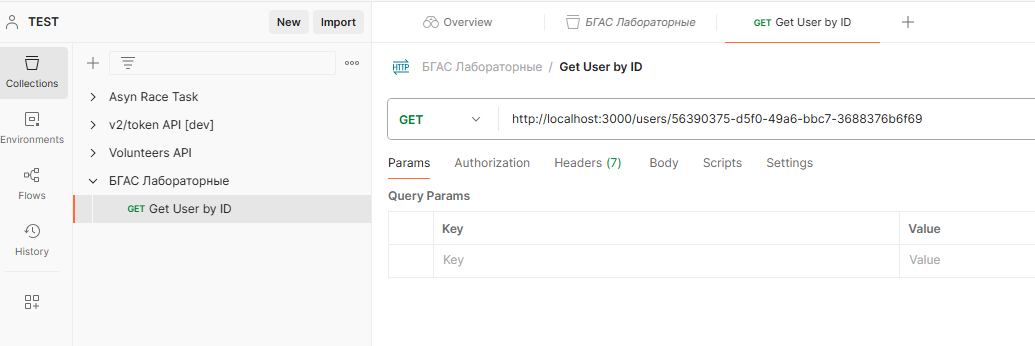


Рисунок 3 – Результат создания нового GET запроса на http://localhost:3000/users/ API с именем "Get User by ID" в коллекции «БГАС Лабораторные» и использование 56390375-d5f0-49a6-bbc7-3688376b6f69 в качестве id

**6 Отправьте запрос**

Нажмите кнопку Send (Отправить). Посмотрите на ответ в нижней части Postman.

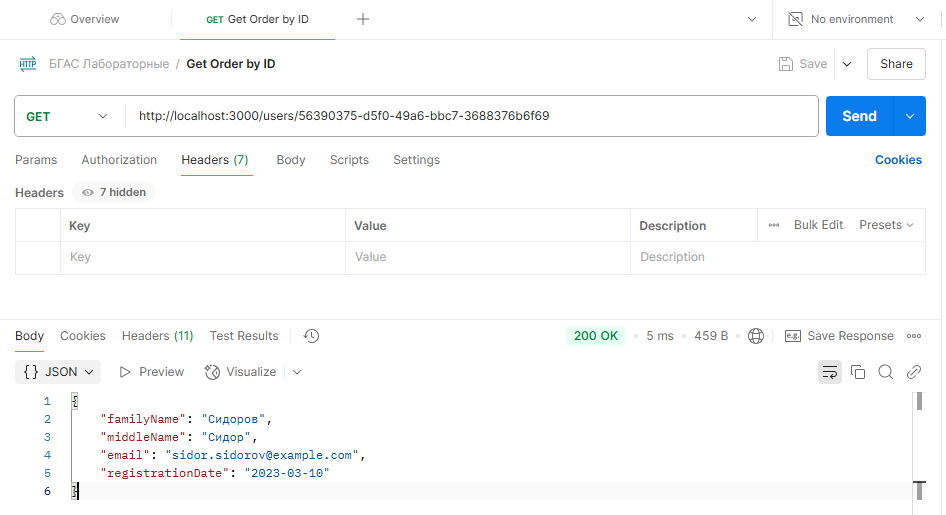


Рисунок 4 – Результат запроса на /users API

|  |  |
| --- | --- |
| **Результат – Бесплатные иконки: файлы и папки** | **Проверьте результат!** |

Если идентификатор пользователя существует в объекте users, вы должны увидеть данные о пользователе в формате JSON.

Если идентификатор пользователя не существует, вы должны получить ответ с кодом 404 и сообщением "Пользователь не найден".

Если произошла ошибка, вы получите сообщение с кодом 500 и текстом "Произошла ошибка на сервере".

**7. По аналогии создайте отдельный запрос к API orders**. Обратите внимание что в этом случае, в самом запросе надо будет так же настроить авторизацию (использовать авторотационный токен, который вы создали ранее, на предыдущих лабораторных).

Тип авторизации вы должны определить САМОСТОЯТЕЛЬНО!

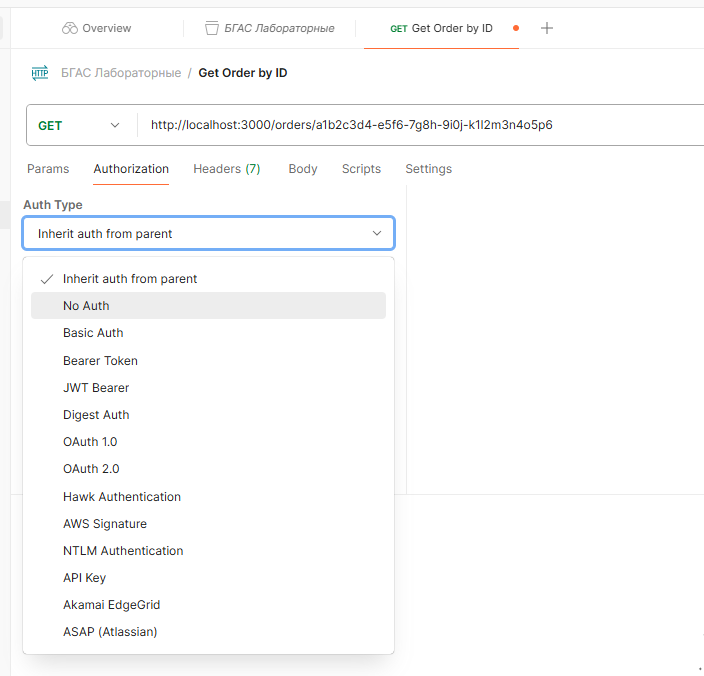


Рисунок 5 – Вкладка «Авторизация» в запросе «Get Order by ID»

8. Показать результаты преподавателю. Если ваше приложение работает корректно, то в таком случае вам нужно будет сделать commit, push в ваш личный репозиторий. Далее, изменения, которые вы сделали в своей ветке необходимо объединить с веткой develop через Pull Request (запрос на слияние).

**Важно! Перед тем как делать commit вам нужно очистить учетные данные git на компьютере!**

**cmdkey /delete:LegacyGeneric:target=git:https://github.com**

**Данная команда используется в операционной системе Windows для удаления сохранённых учётных данных (credentials) из менеджера учётных данных для специфического ресурса, в данном случае — для доступа к репозиторию на GitHub по протоколу HTTPS.**

**Напоминание!** Придерживаться git-branch стратегии важно, так как она обеспечивает структурированность и организованность в процессе разработки. Четкое разделение веток позволяет изолировать изменения, что минимизирует риски возникновения конфликтов и ошибок в стабильной версии кода. Это также упрощает совместную работу, так как каждый разработчик может параллельно работать над своими задачами. Кроме того, благодаря стратегии управления версиями становится легче отслеживать и контролировать изменения, а также быстро откатывать их в случае необходимости. Наличие отдельных веток для различных этапов разработки улучшает интеграцию с процессами непрерывной интеграции и доставки (CI/CD), что в конечном итоге способствует повышению качества продукта.