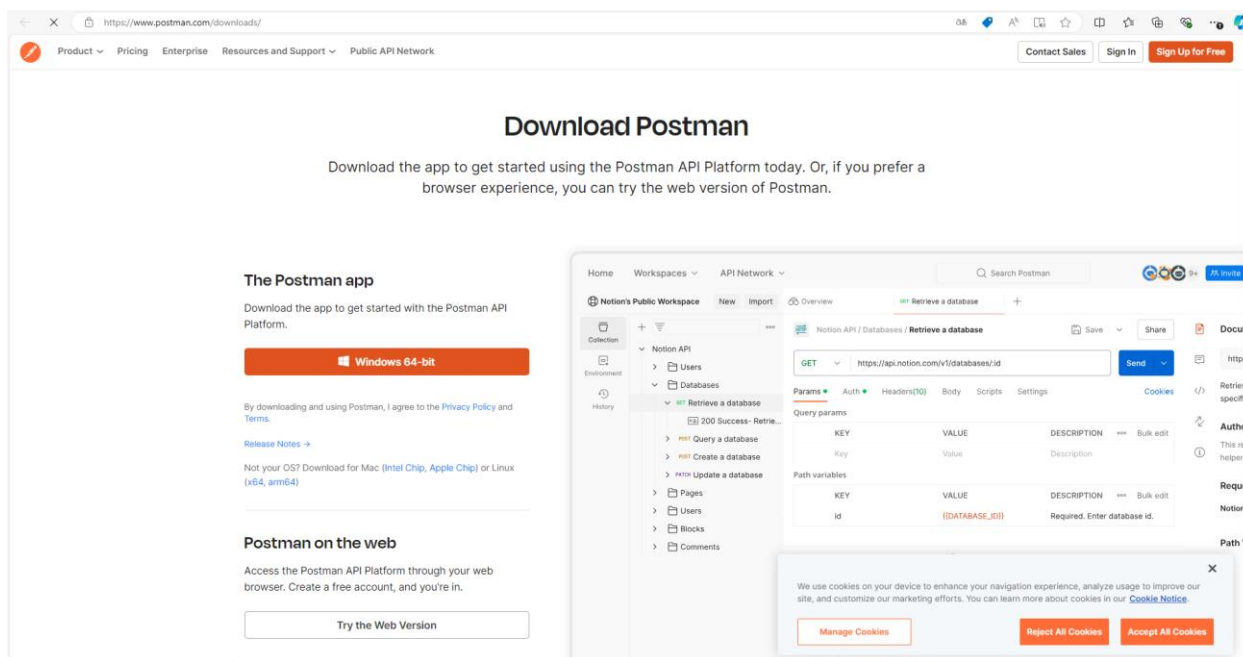


Установка Postman

1. Скачивание Postman:

- Перейдите на официальный сайт <https://www.postman.com/downloads/> и скачайте версию для вашей операционной системы.

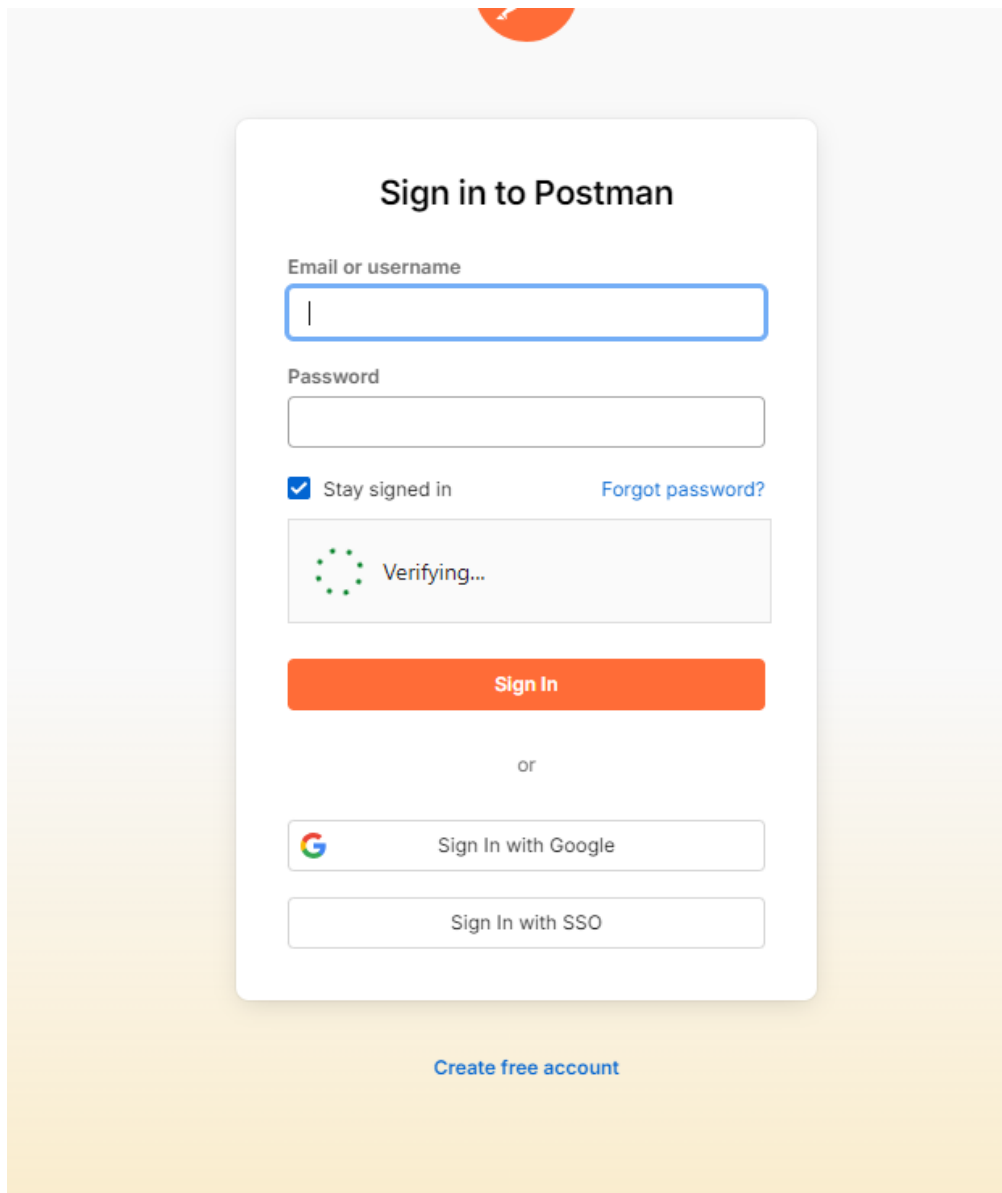
- Установите приложение, следуя инструкциям установщика.



[Изображение 1: Главная страница Postman с кнопкой загрузки]

2. Создание аккаунта:

- После установки вы можете создать аккаунт или войти в существующий. Это позволит вам сохранять ваши коллекции и настройки в облаке.



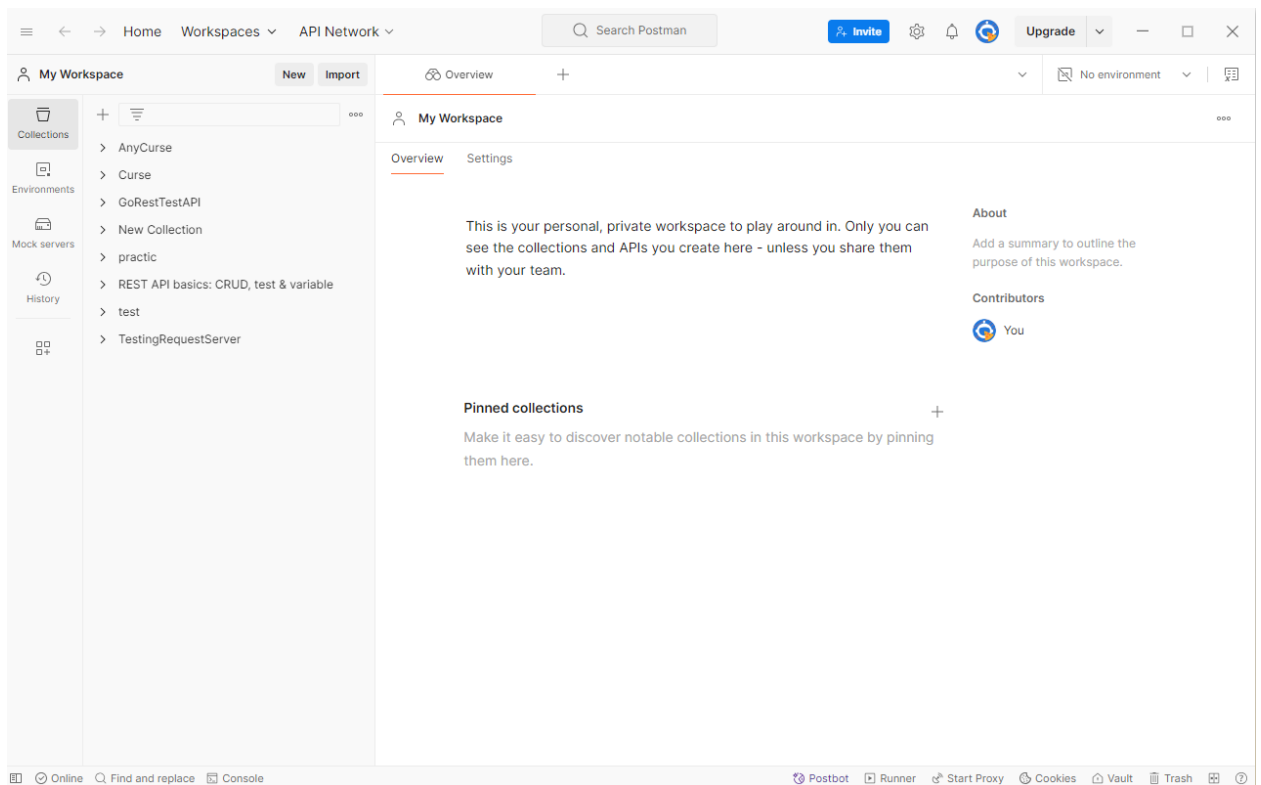
[Изображение 2: Экран входа в Postman]

Основные функции Postman

1. Интерфейс Postman

Postman имеет интуитивно понятный интерфейс, который состоит из следующих основных компонентов:

- Навигационная панель: Здесь вы можете управлять коллекциями, окружениями и настройками.
- Рабочая область: Здесь вы будете создавать и отправлять запросы.
- История запросов: Позволяет просматривать ранее отправленные запросы.

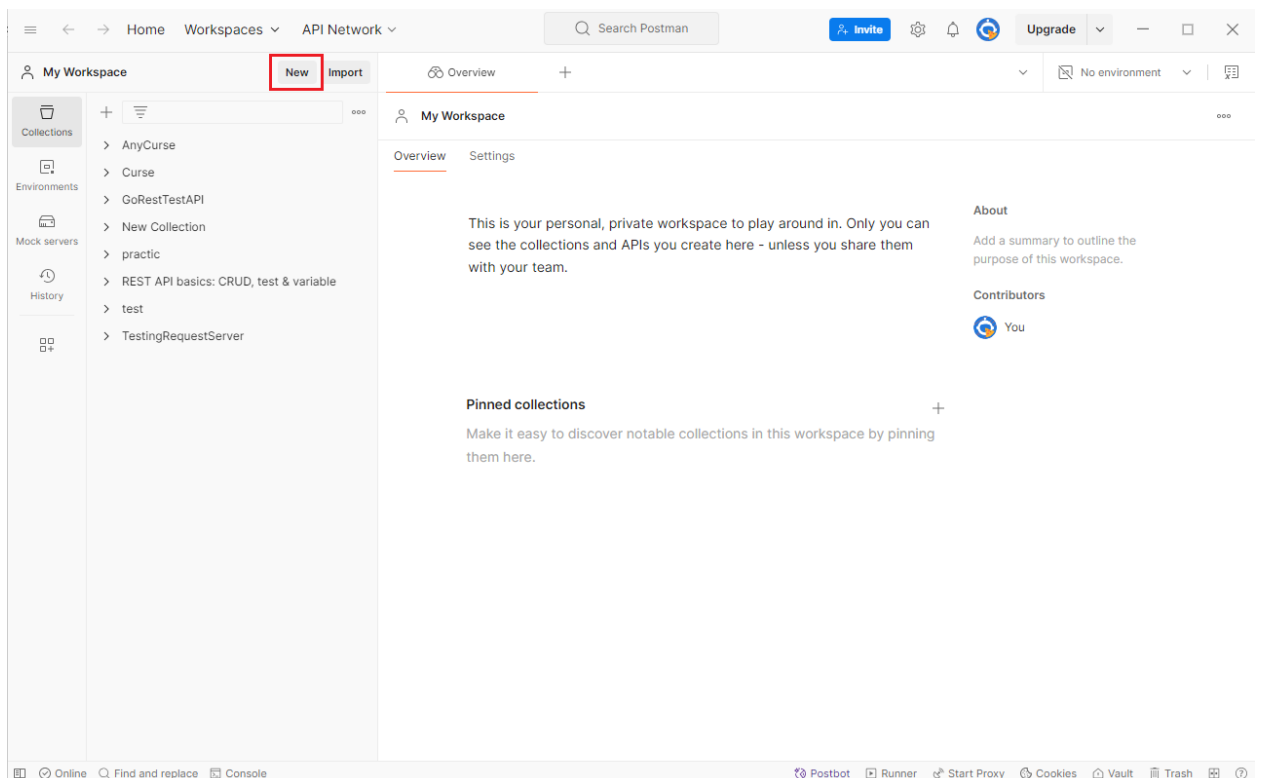


[Изображение 3: Основной интерфейс Postman с выделенными компонентами]

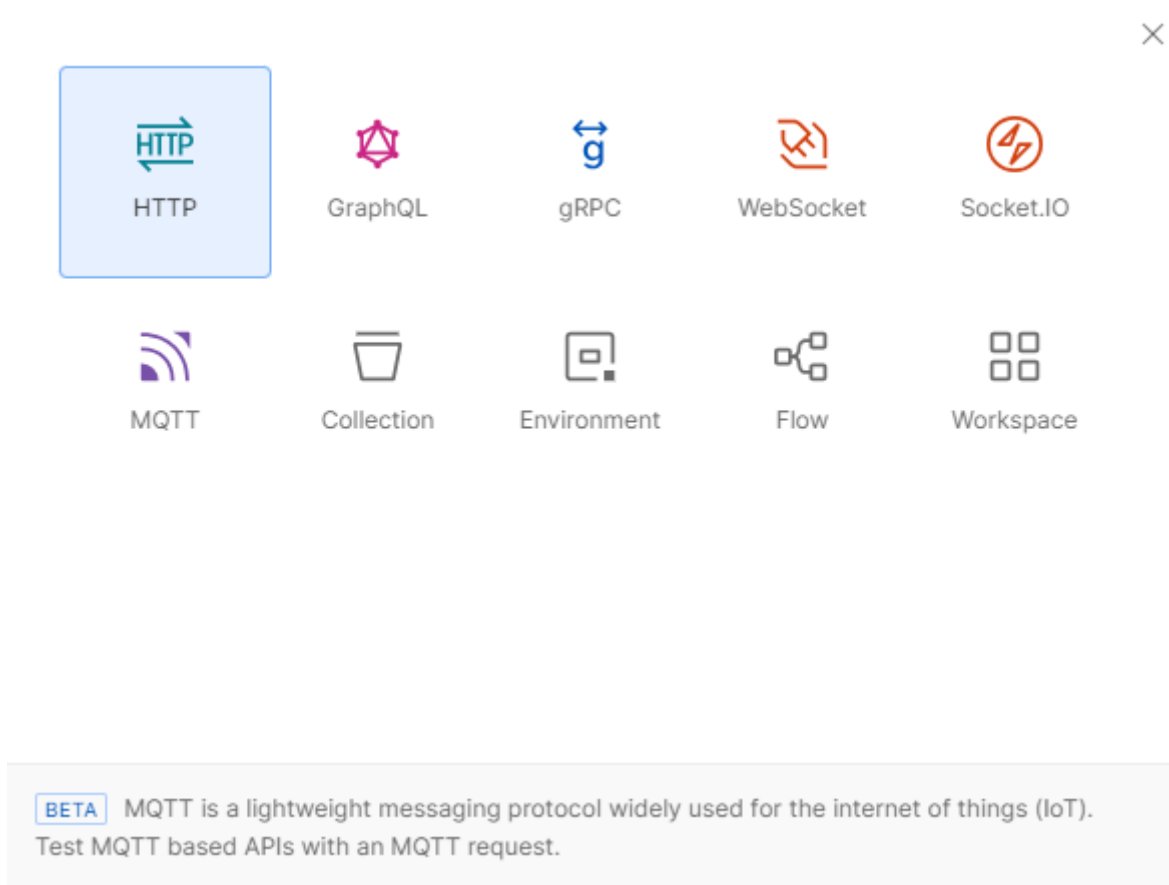
2. Создание запроса

1. Создание нового запроса:

- Нажмите на кнопку "New" в верхнем левом углу и выберите "Request".

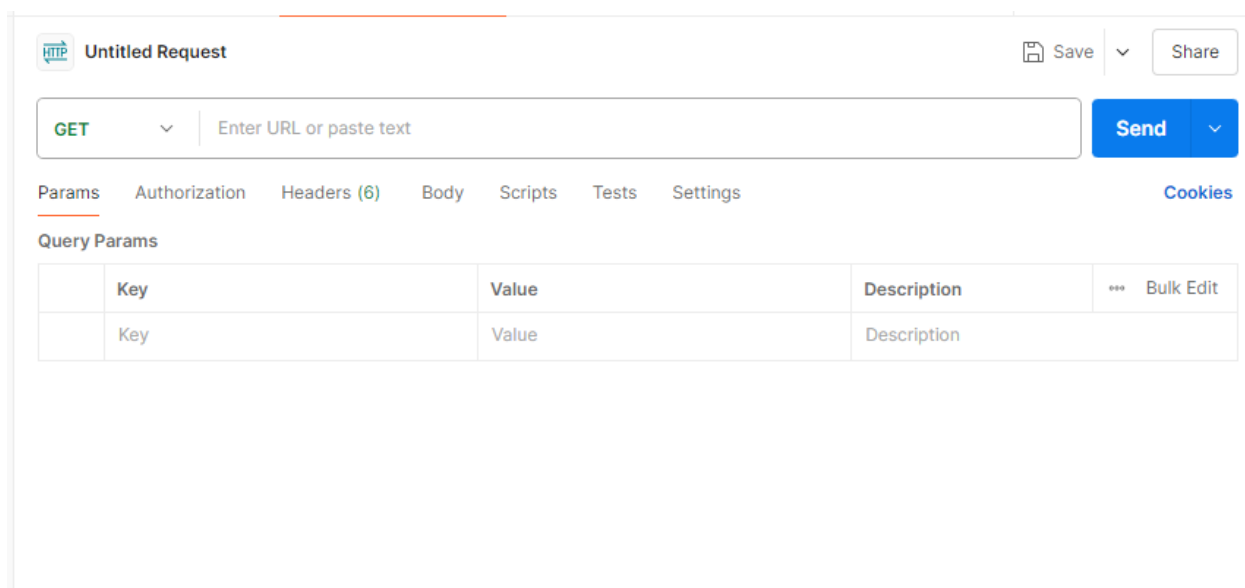


[Изображение 4: Кнопка "New" для создания нового запроса]



2. Настройка запроса:

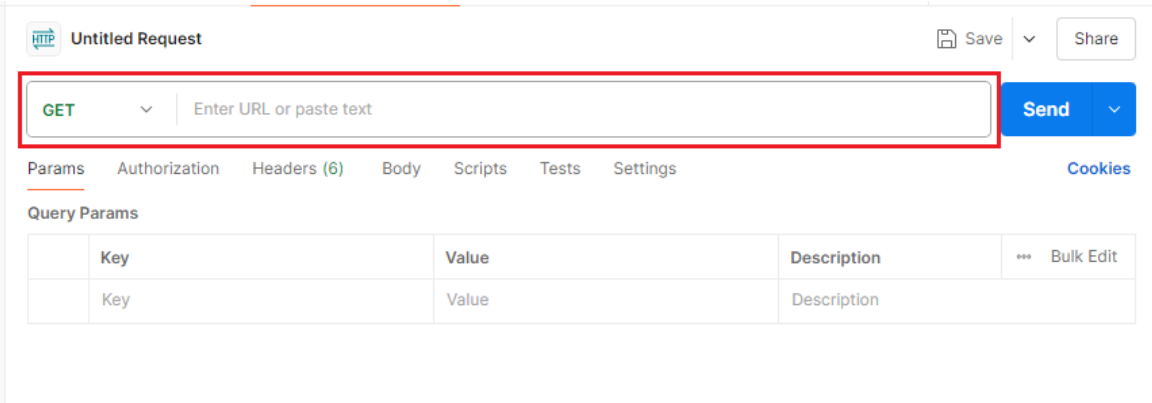
- Введите название запроса и выберите коллекцию, в которой он будет сохранен.
- Выберите метод HTTP (GET, POST, PUT, DELETE и т.д.) из выпадающего списка.



[Изображение 5: Окно настройки нового запроса]

3. Ввод URL:

- Введите URL-адрес API, к которому хотите отправить запрос.

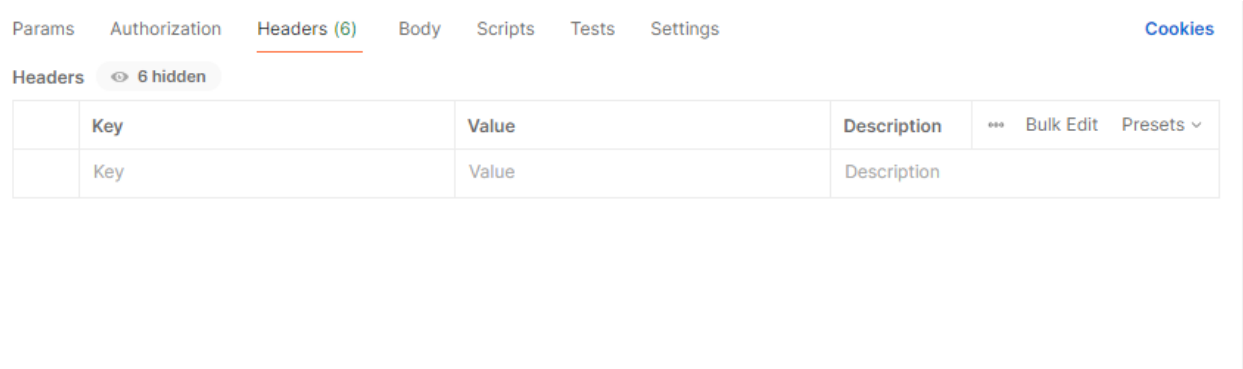


[Изображение 6: Поле для ввода URL-адреса]

3. Отправка запроса

1. Добавление параметров и заголовков:

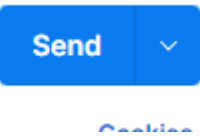
- Вы можете добавлять параметры запроса и заголовки, нажав на соответствующие вкладки под полем URL.



[Изображение 7: Вкладки для параметров и заголовков]

2. Отправка запроса:

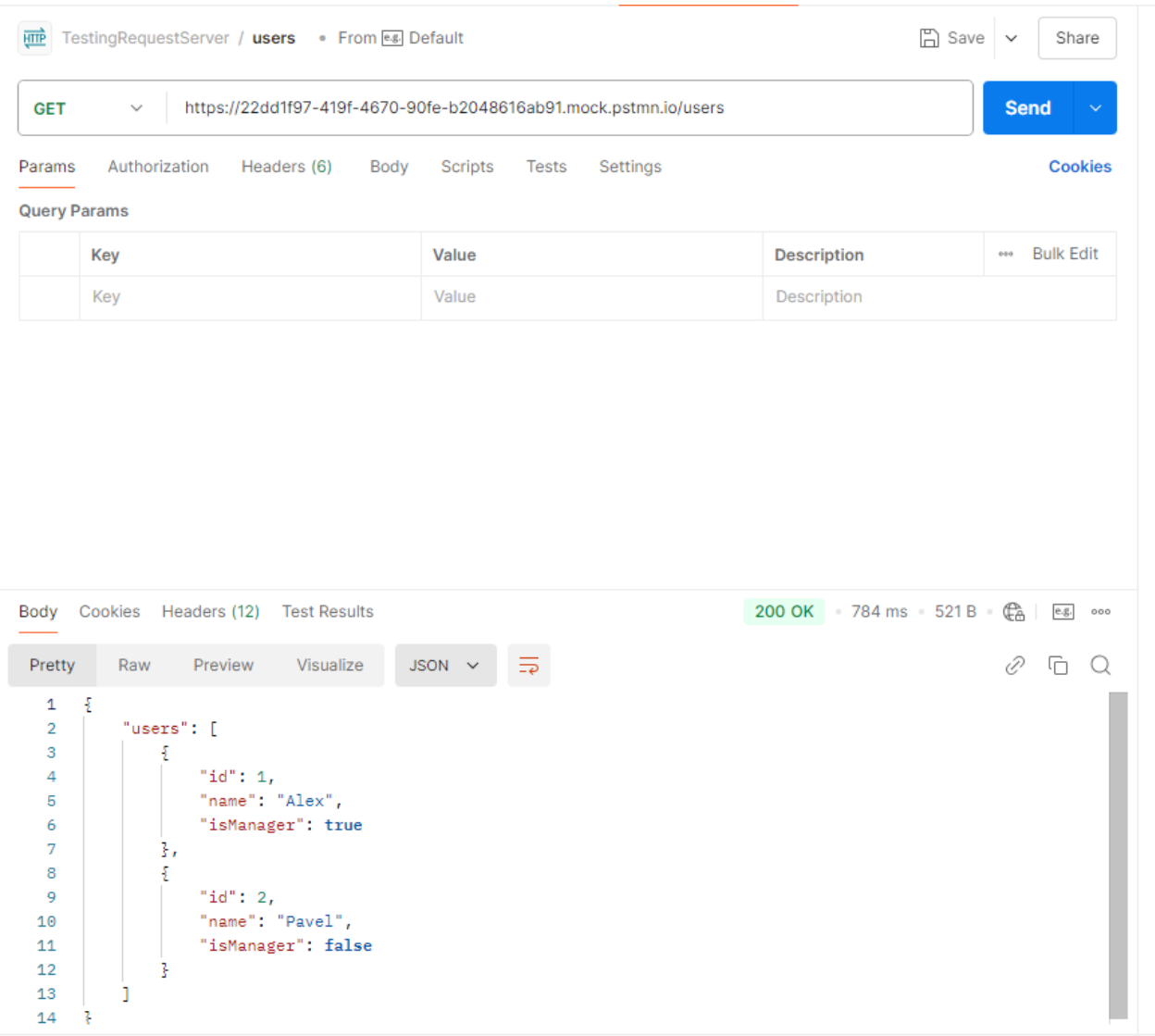
- Нажмите кнопку "Send", чтобы отправить запрос.



[Изображение 8: Кнопка "Send" для отправки запроса]

3. Просмотр ответа:

- После отправки запроса вы увидите ответ API в нижней части экрана. Здесь отображается статус ответа, заголовки и тело ответа.

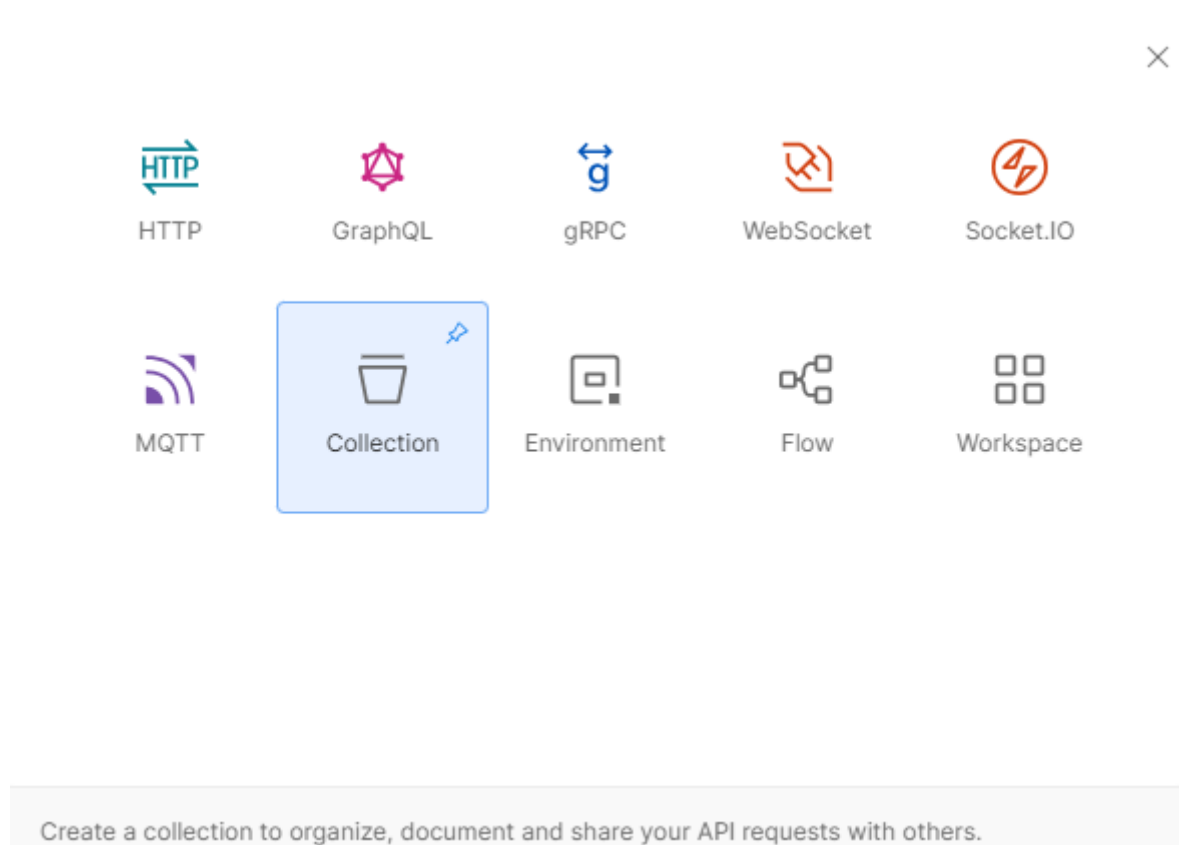


[Изображение 9: Ответ API с отображением статуса и тела ответа]

4. Работа с коллекциями

1. Создание коллекции:

- Коллекции позволяют организовывать запросы. Нажмите на "New" и выберите "Collection".



[Изображение 10: Окно создания новой коллекции]

2. Добавление запросов в коллекцию:


- При создании нового запроса вы можете выбрать, в какую коллекцию его сохранить.


SAVE REQUEST


Request name


[Add description](#)


Save to Select a collection/folder


 Search for collection or folder


 AnyCurse


 Curse


 GoRestTestAPI

 New Collection

 practic

 REST API basics: CRUD, test & variable

 test

 TestingRequestServer

New Collection

Save

Cancel

[Изображение 11: Выбор коллекции для сохранения запроса]

3. Экспорт и импорт коллекций:

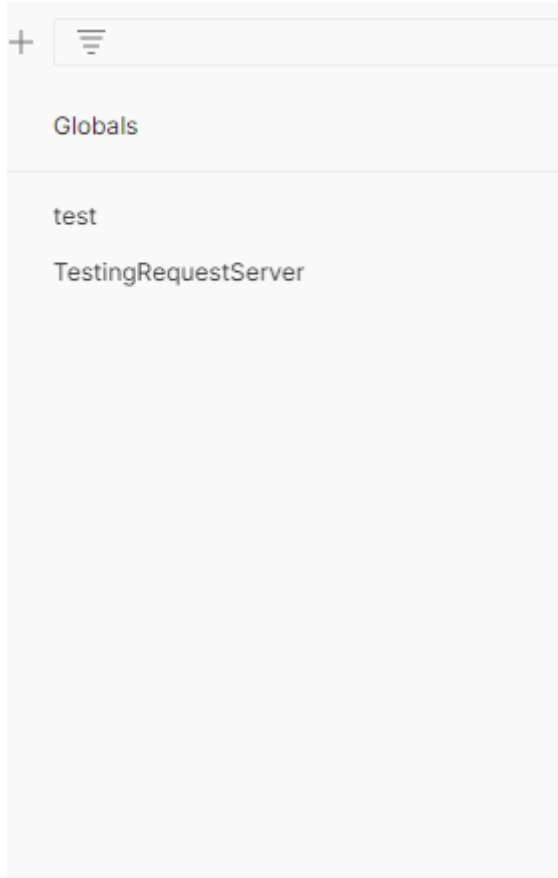
- Вы можете экспортировать коллекции для совместного использования с другими пользователями или импортировать коллекции, созданные другими.

5. Использование переменных

Postman поддерживает переменные, которые можно использовать для динамического управления значениями в запросах.

1. Создание переменной:

- Перейдите в " Environments" и создайте новое окружение, добавив переменные.



[Изображение 13: Окно управления окружениями]

2. Использование переменной в запросе:

- Вы можете использовать переменные в URL и заголовках, обрамляя их двойными фигурными скобками, например: `{{variableName}}`.



[Изображение 14: Пример использования переменной в запросе]

6. Настройка мокирования

1. Сохраните запрос:

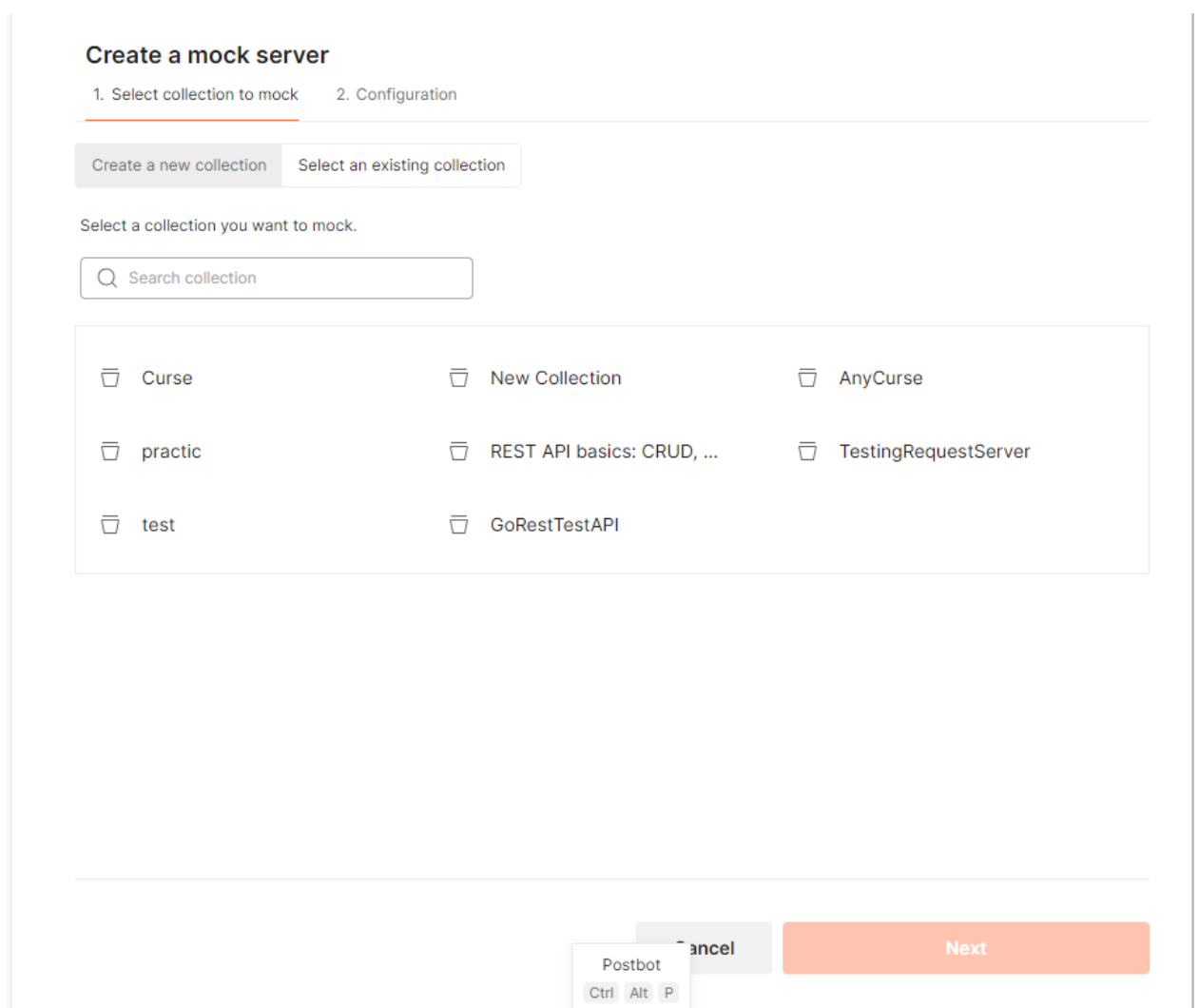
- После настройки запроса нажмите "Save".

2. Создание мока:

- Перейдите на вкладку "Mock" в верхней части экрана.
- Нажмите на кнопку "Create a Mock Server".

3. Настройка мок-сервера:

- Выберите коллекцию, для которой вы хотите создать мок.
- Убедитесь, что выбран нужный запрос.
- Установите параметры для мокирования, такие как имя и описание.



[Изображение 16: Настройка параметров для Mock Server]

4. Создание мока:

- Нажмите "Create Mock Server". Postman создаст мок-сервер и предоставит вам URL для доступа к нему.

🔍 Search your calls...

🔄 Refresh Logs



No mock server calls yet

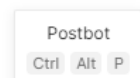
To call the mock server, follow these steps:

1. Send a request to the following mock server URL, followed by the request path

<https://a83936fc-082f-4428-9d6c-aa0945b925af.mock.pstmn.io> 📄

2. Add examples responses to each request that the mock server will return.

[Learn what examples are and how to use them](#) ➤

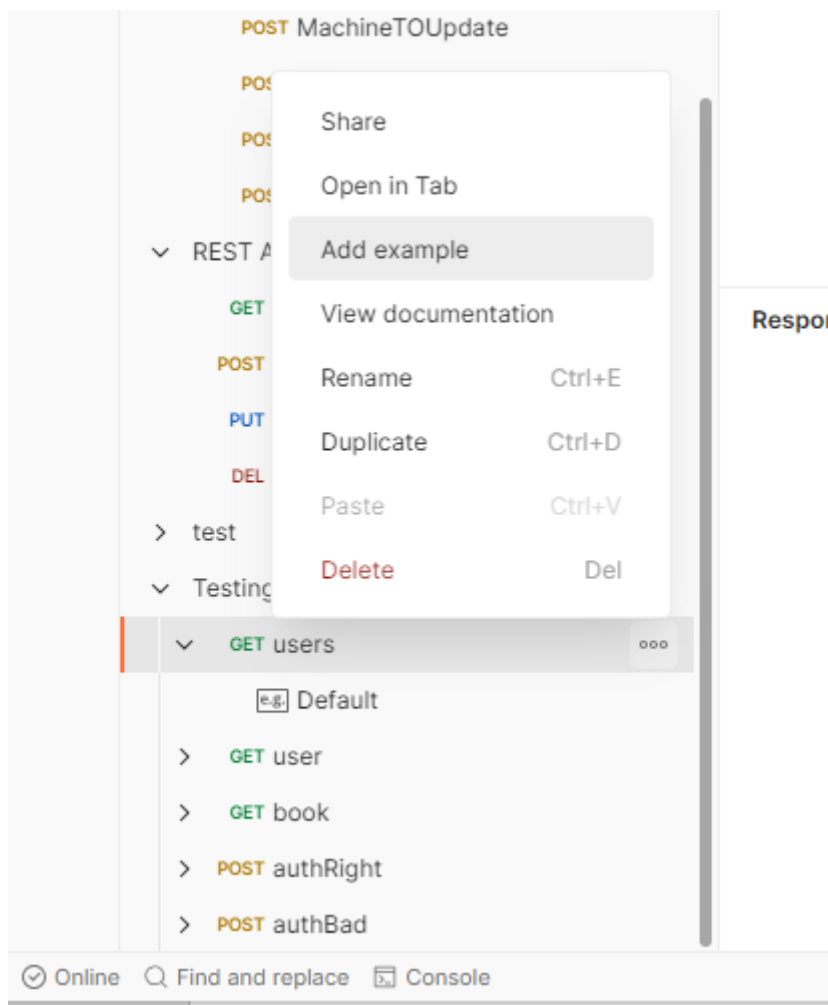


[Изображение 17: URL мок-сервера после его создания]

7. Настройка ответов

1. Добавление ответов:

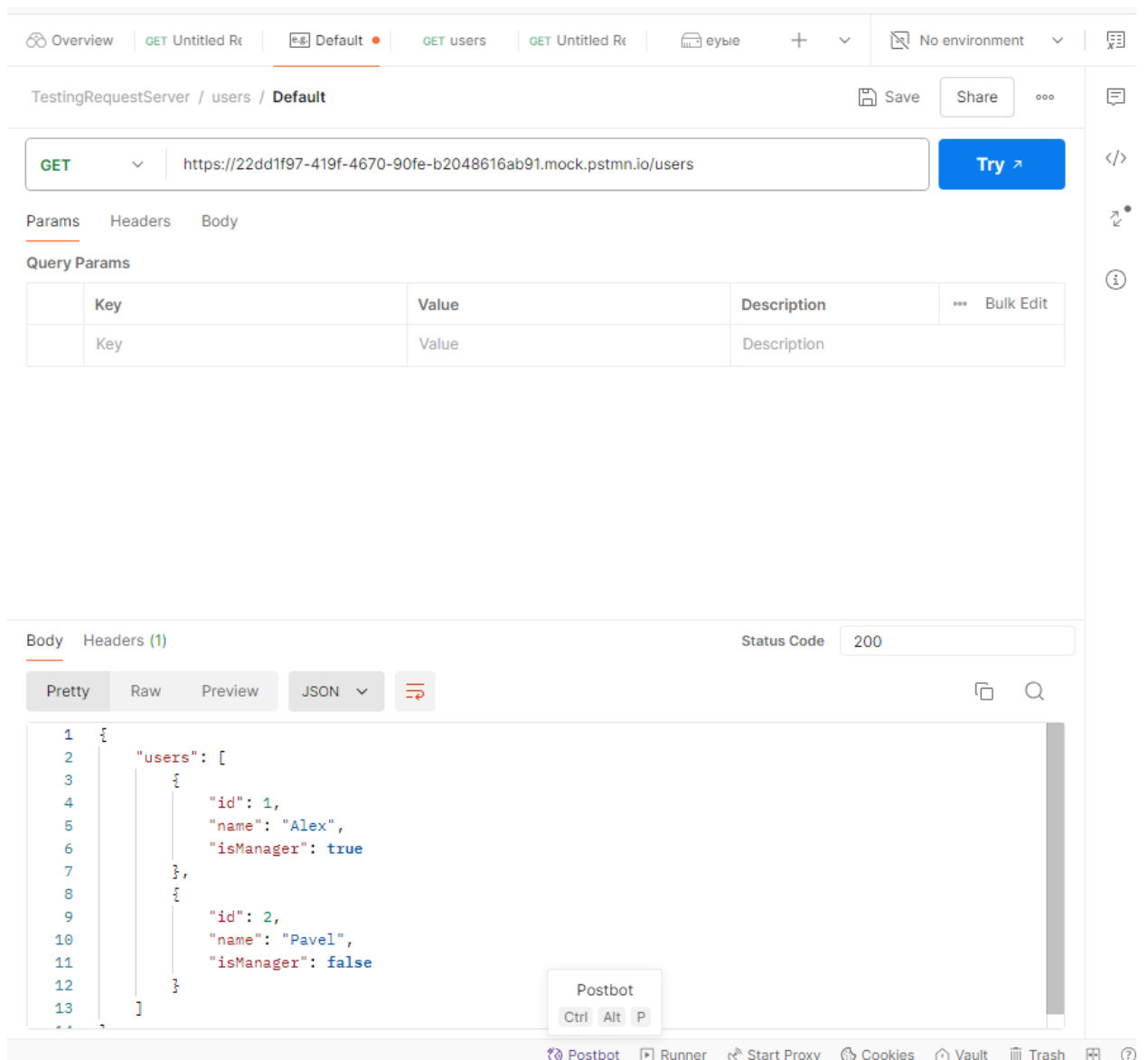
- Внутри вашего запроса вы можете настроить ответы, которые будет возвращать мок-сервер.
- Перейдите на вкладку "Examples" и нажмите "Add Example".



[Изображение 18: Добавление примера ответа]

2. Настройка примера ответа:

- Введите статус ответа (например, 200), заголовки и тело ответа (например, JSON-объект).
- Сохраните пример.

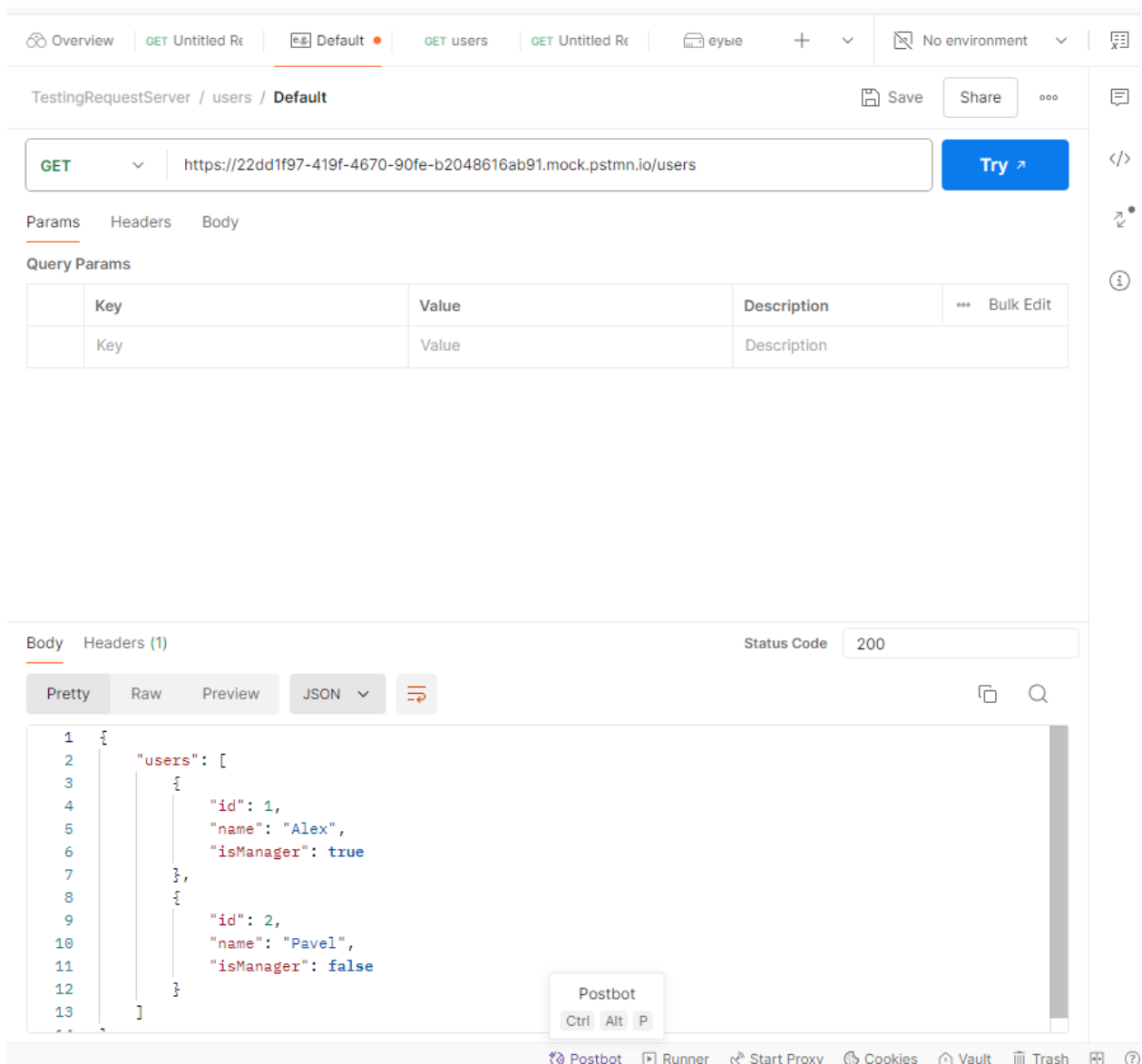


[Изображение 19: Настройка примера ответа с JSON-объектом]

5. Тестирование мок-сервера

1. **Отправка запроса к мок-серверу**:

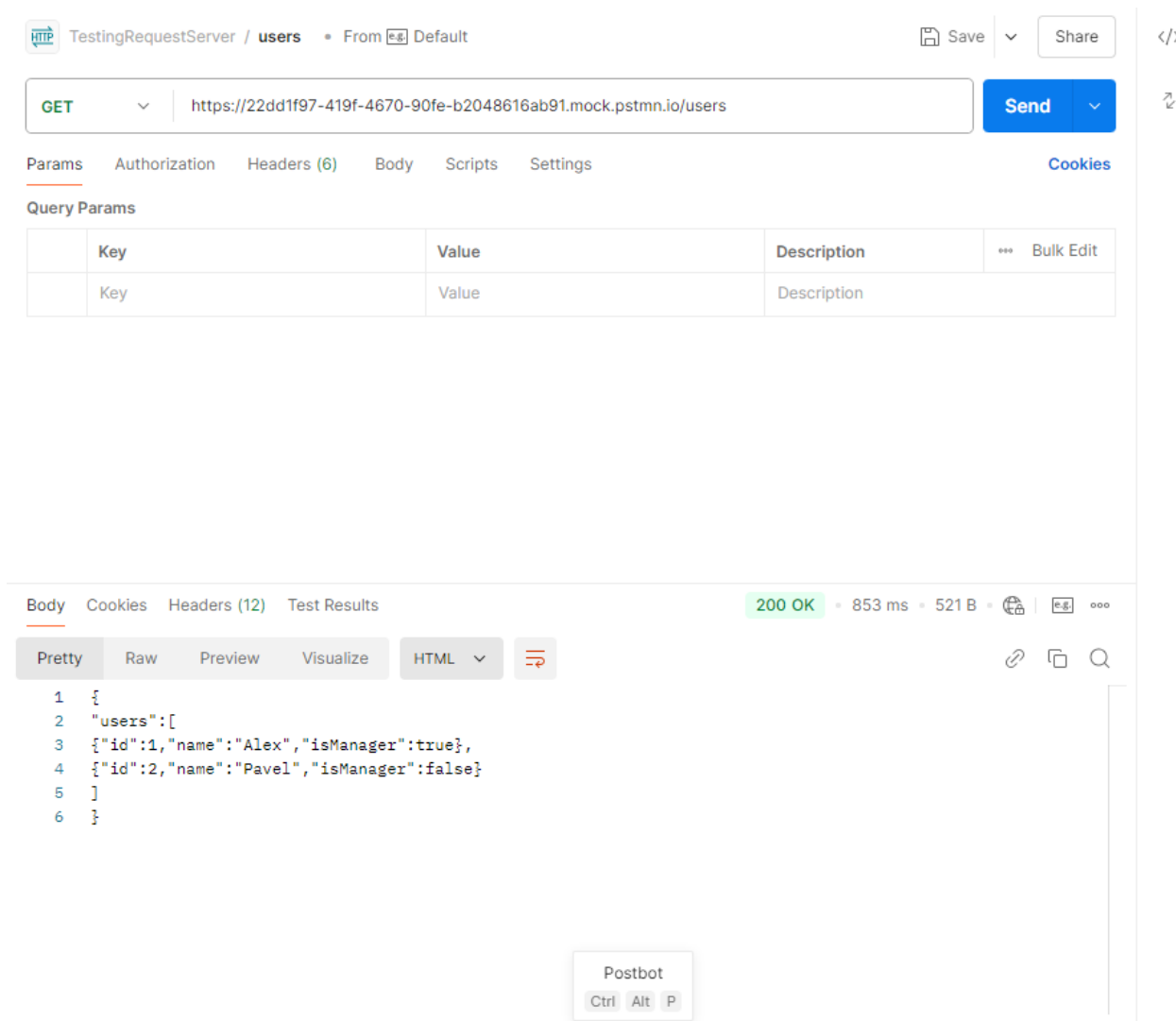
- Используйте URL, предоставленный мок-сервером, чтобы отправить запрос.
- Убедитесь, что вы получаете ожидаемый ответ.



[Изображение 20: Отправка запроса к мок-серверу]

2. Проверка ответа:

- Убедитесь, что ответ соответствует тому, что вы настроили в примере.



[Изображение 21: Ответ от мок-сервера]

RXJS Vue

1. Создайте компонент UserList.vue в папке src/components:

```
<template>
  <div>
    <h1>Список пользователей</h1>
    <ul>
      <li v-for="user in users" :key="user.id">
        {{ user.name }} - {{ user.email }}
      </li>
    </ul>
    <div v-if="error">{{ error }}</div>
    <div v-if="loading">Загрузка...</div>
  </div>
</template>

<script lang="ts">
import { defineComponent, ref, onMounted } from 'vue';
import { from } from 'rxjs';
import { ajax } from 'rxjs/ajax';

export default defineComponent({
  setup() {
    const users = ref<User[]>([]);
    const error = ref<string>('');
    const loading = ref<boolean>(true);

    onMounted(() => {
      loading.value = true;
      from(ajax<User[]>('https://api.example.com/users')).subscribe(
        (data) => {
          users.value = data;
          loading.value = false;
        },
        (err) => {
          error.value = err.message;
          loading.value = false;
        }
      );
    });
  }
});
```

```

import { catchError, map } from 'rxjs/operators';

class User {
  id: number;
  name: string;
  email: string;
}

export default defineComponent({
  name: 'UserList',
  setup() {
    const users = ref<User[]>([]);
    const loading = ref<boolean>(true);
    const error = ref<string | null>(null);

    const fetchUsers = () => {
      return from(
        ajax.getJSON<User[]>('https://jsonplaceholder.typicode.com/users')
      ).pipe(
        map((response) => {
          loading.value = false;
          return response;
        }),
        catchError((err) => {
          loading.value = false;
          error.value = 'Ошибка при загрузке данных';
          return [];
        })
      );
    };

    onMounted(() => {
      fetchUsers().subscribe({
        next: (data) => {
          users.value = data;
        },
        error: (err) => {
          console.error('Error fetching users:', err);
        },
      });
    });

    return {
      users,
      loading,
      error,
    };
  },
});
</script>

<style scoped>
ul {
  list-style-type: none;

```



```
padding: 0;
}

li {
  margin: 5px 0;
}
</style>
```

Объяснение кода

1. Импорт необходимых модулей:

- Мы импортируем `from` из `rxjs`, `ajax` из `rxjs/ajax`, а также необходимые функции из `Vue`.

2. Класс `User`:

- Определяем класс `User`, который описывает структуру данных пользователя.

3. Создание реактивных переменных:

- `users` — массив пользователей.
- `loading` — флаг, указывающий на загрузку данных.
- `error` — строка для хранения сообщений об ошибках.

4. Функция `fetchUsers`:

- Используем `ajax.getJSON` для получения данных из API.
- Применяем операторы `map` и `catchError` для обработки успешного ответа и ошибок.

5. `onMounted`:

- При монтировании компонента вызываем `fetchUsers` и подписываемся на результат, обновляя состояние переменных `users`, `loading` и `error`.

2. Обновление `App.vue`

Теперь необходимо подключить наш компонент `UserList` в `App.vue`.

Задание:

1. Создать mock сервер библиотеки, реализовать в mock сервере запрос на получение всех книг, в качестве ответа на запрос отправлять коллекцию json объектов `book`. Объект `book`

обладает ключами author, name, year ключ author содержит Фамилию и инициалы автора, name – название книги и year-год выхода.

Пример:

```
{"author":Имя.Ф.О, "name":"Книга1", "year":1994},
```

2. Создать vue компонент который обратиться к mock серверу и отобразит корректные данные, для получения данных использовать rx js