**MA - Travel Itinerary Generator**

**User Microservice** - responsible for **user management, authentication, and role-based access control**. It provides functionalities for **user registration, profile management, role assignment, and account status management**.

### 

| **METHODS** | |
| --- | --- |
| registerUser(User user) | **User Registration**   * Assigns **ADMIN** role to the first registered user, others get **USER**. * Removes **preferences** for admin users. * Encrypts the password before saving. |
| getUserPreferences(Long userId) | **Fetching User Preferences**   * Retrieves **preferences** for a given user ID. |
| getUserByEmail(String email) | **Fetching User by Email**   * Finds a **user by email** (useful for login/authentication) |
| getUsersPaged(int page, int size) | **Paginated User Retrieval**   * Retrieves **non-admin users** in a paginated manner. |
| getUserById(Long id) | **Fetching User by ID**   * Retrieves a **specific user** by ID. |
| disableUser(Long id) | **Disabling a User**   * **Soft disables** a user by setting **active = false** |
| enableUser(Long id) | **Enabling a User**   * Re-enables a **disabled** user. |
| updateUser(Long userId, User updatedUser) | **Updating User Information**   * Updates **name, email, and preferences** while keeping other fields unchanged. |
| getRoleByUserId(Long userId) | **Getting User Role**   * Fetches the **role** of a user by their ID. |
| promoteToAdmin(Long userId) | **Promoting User to Admin**   * Changes the user's role to **ADMIN**. |

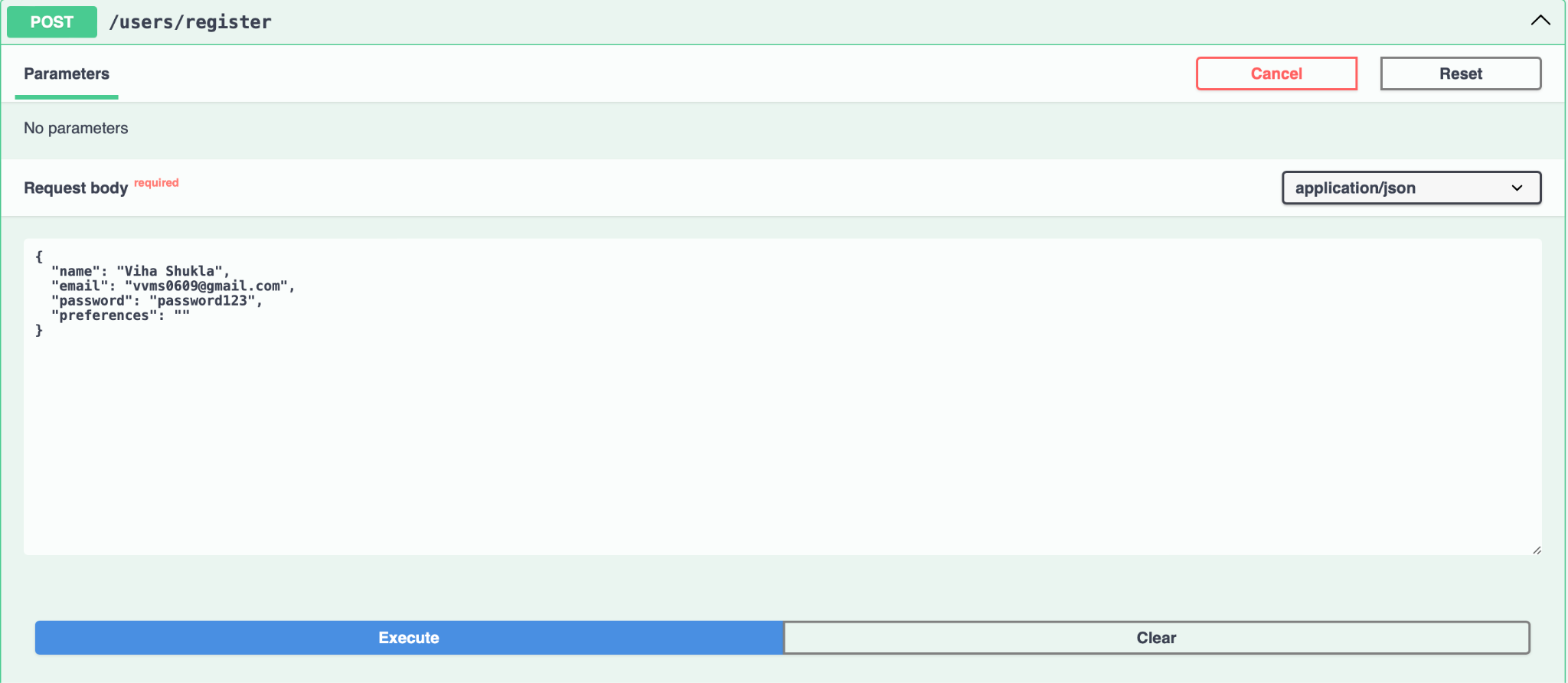
**UserController**

**HTTP REQUESTS**

| **HTTP Method** | **Endpoint** | **Request Parameters/Body** | **Functionality** | **Response** |
| --- | --- | --- | --- | --- |
| **POST** | /users/register | **Body:** UserJSON | Registers a new user, assigns role, encrypts password | Returns saved user details (200 OK) |
| **GET** | /users/profile | **Query Param:**userId | Retrieves user details by ID | Returns user details (200 OK) or 404 Not Found |
| **PUT** | /users/profile | **Query Param:**userId  **Body:** User JSON | Updates user details (name, email, preferences) | Returns updated user (200 OK) |
| **GET** | /users/email/{email} | **Path Variable:**email | Fetches user details by email | Returns UserResponse DTO (200 OK) or throws exception |
| **GET** | /users/{id}/preferences | **Path Variable:** id | Retrieves user preferences by user ID | Returns preferences (200 OK) or 404 Not Found |

**Output** -> using **Swagger - ui**

1. **Create the user** ( first user is by default admin with no preferences)

****

**Response**

{

"userId": 1,

"name": "Viha Shukla"

"email": "vvms0609@gmail.com"

"password": 3205105Eg1C9mu553=27KnYSWaVutalwo205dfnxPISCGDpNXGbj.aRnYel

"active": true,

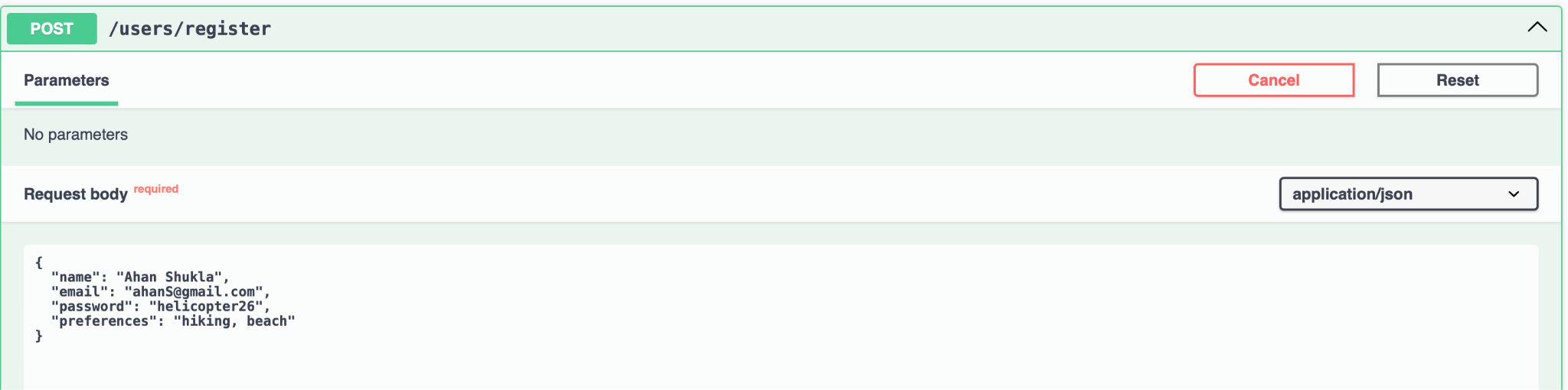
"preferences": null,

"role": "ADMIN"

}

****

1. **Creating an user**

****

**Response**

{

"userId": 2,

"name": "Ahan Shukla",

"email": "ahanS@gmail.com",

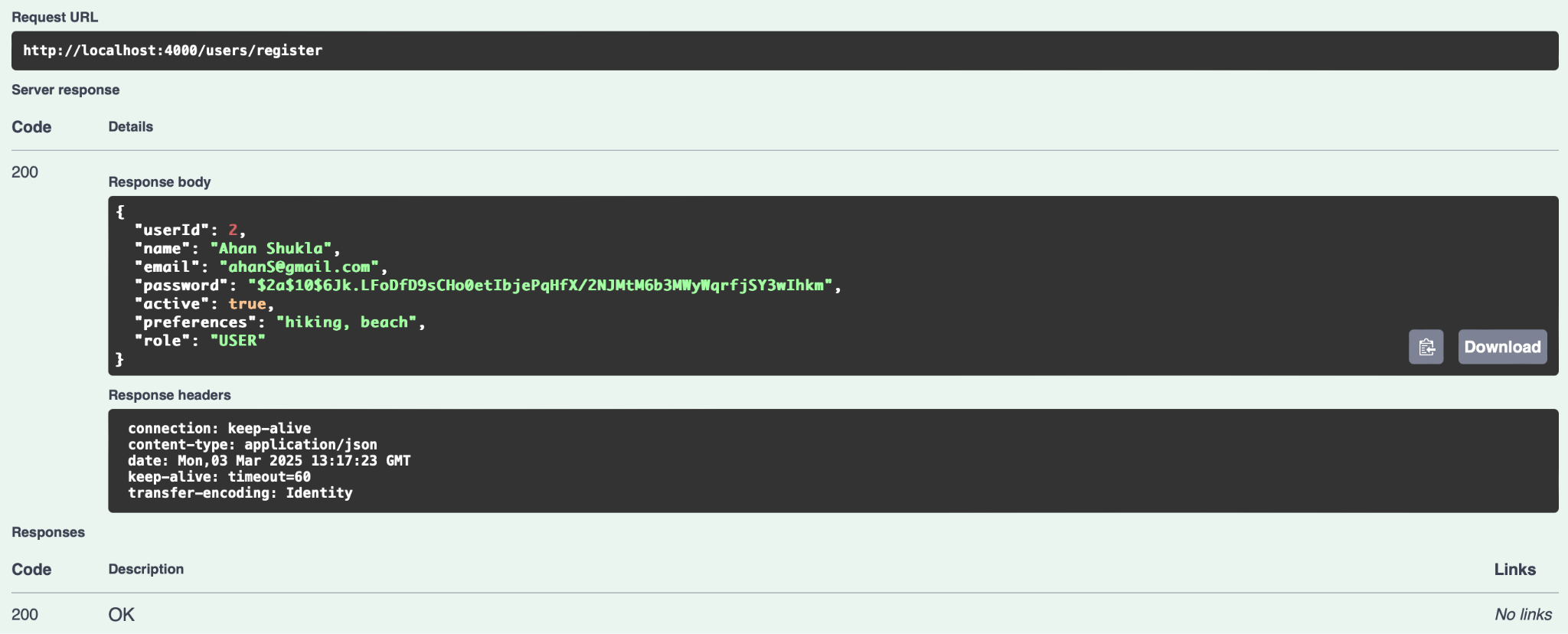
"password": "$2a$10$6Jk.LFoDfD9sCHo0etIbjePqHfX/2NJMtM6b3MWyWqrfjSY3wIhkm",

"active": true,

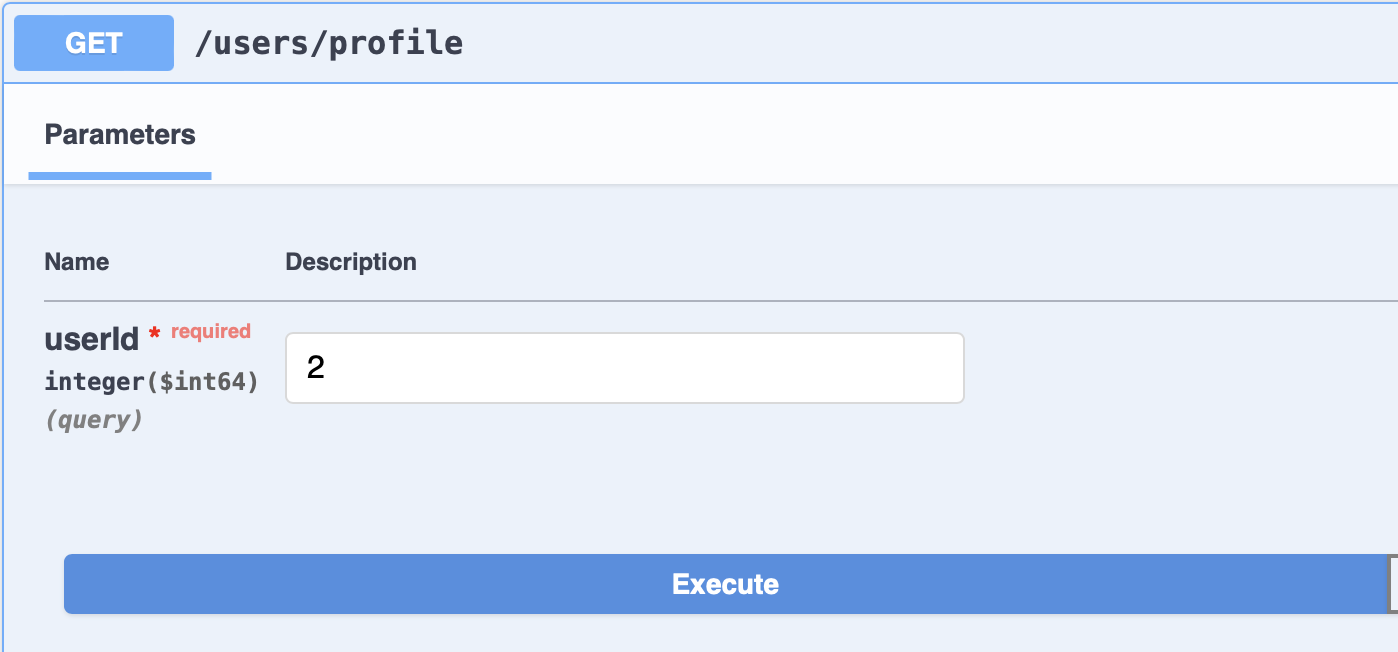
"preferences": "hiking, beach",

"role": "USER"

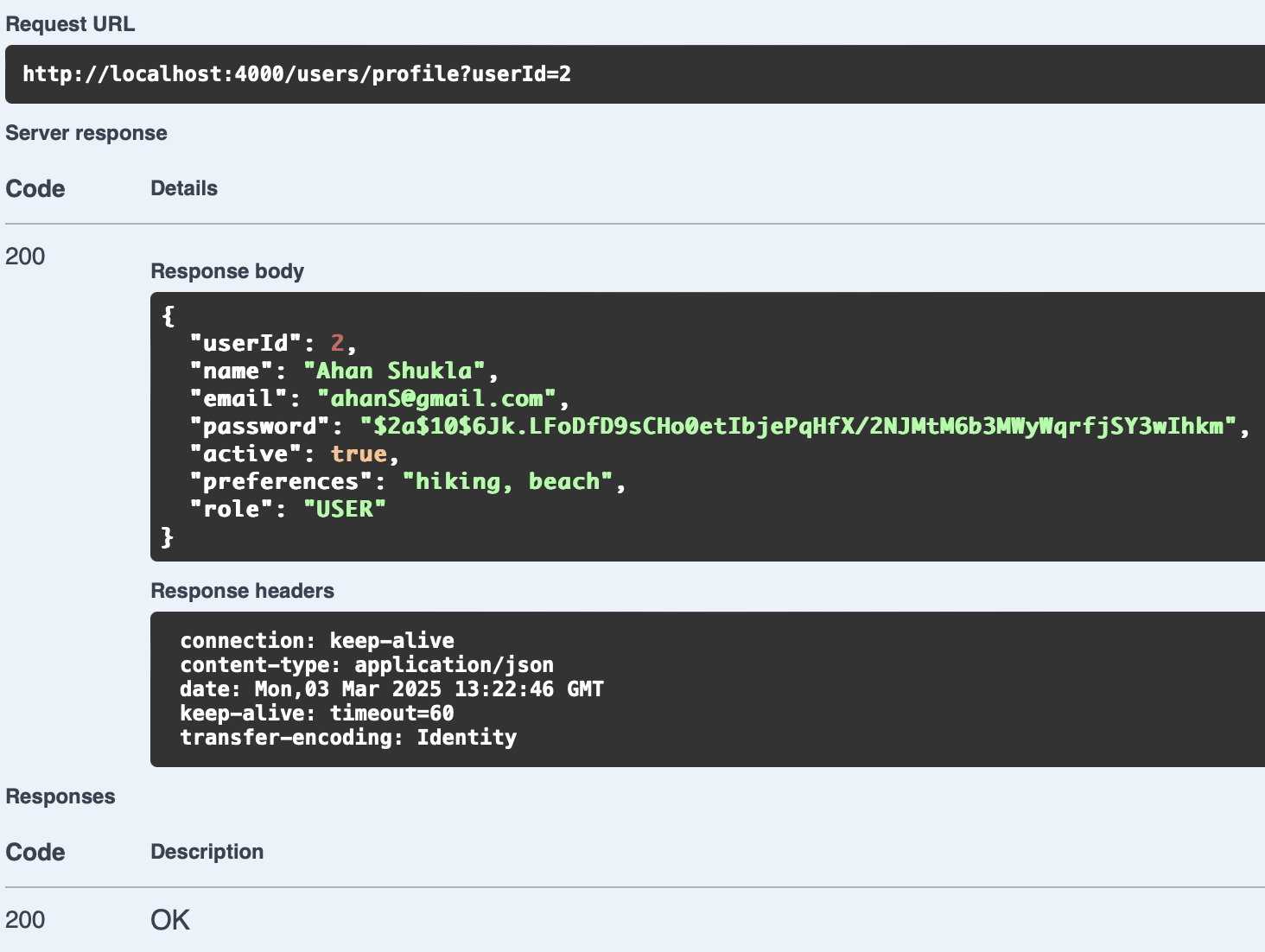
}

****

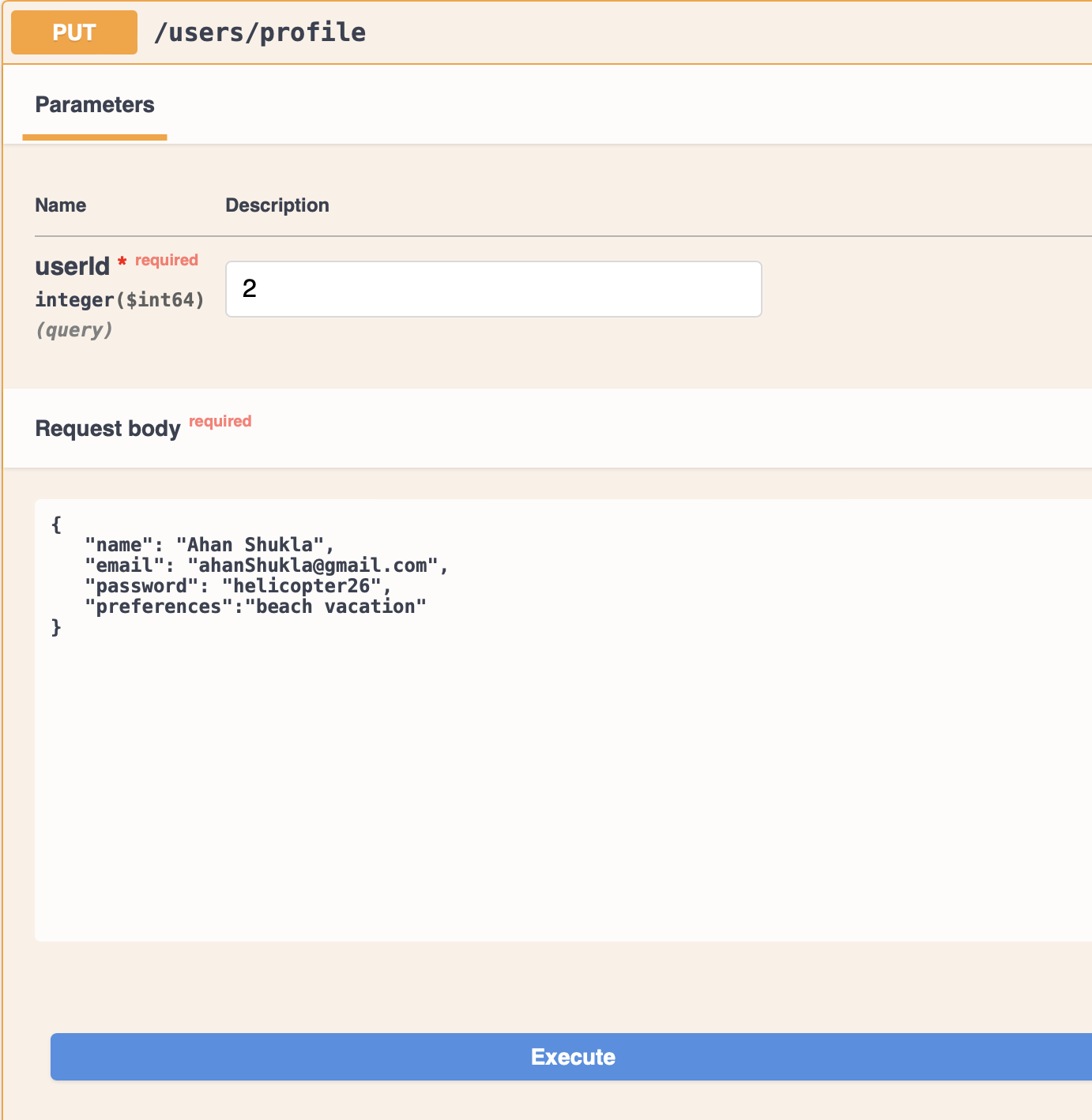
1. **Get user profile**



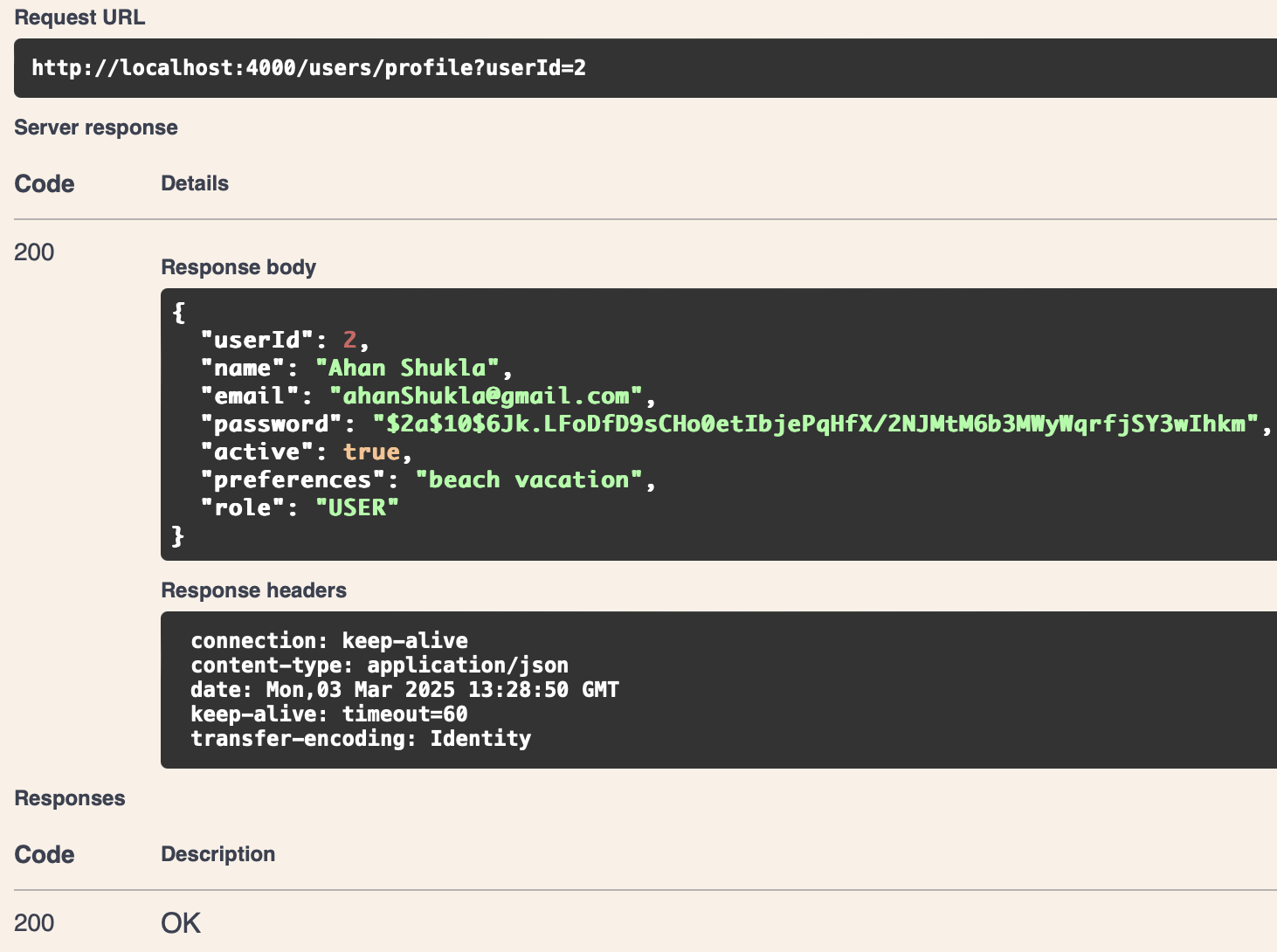
**Response**



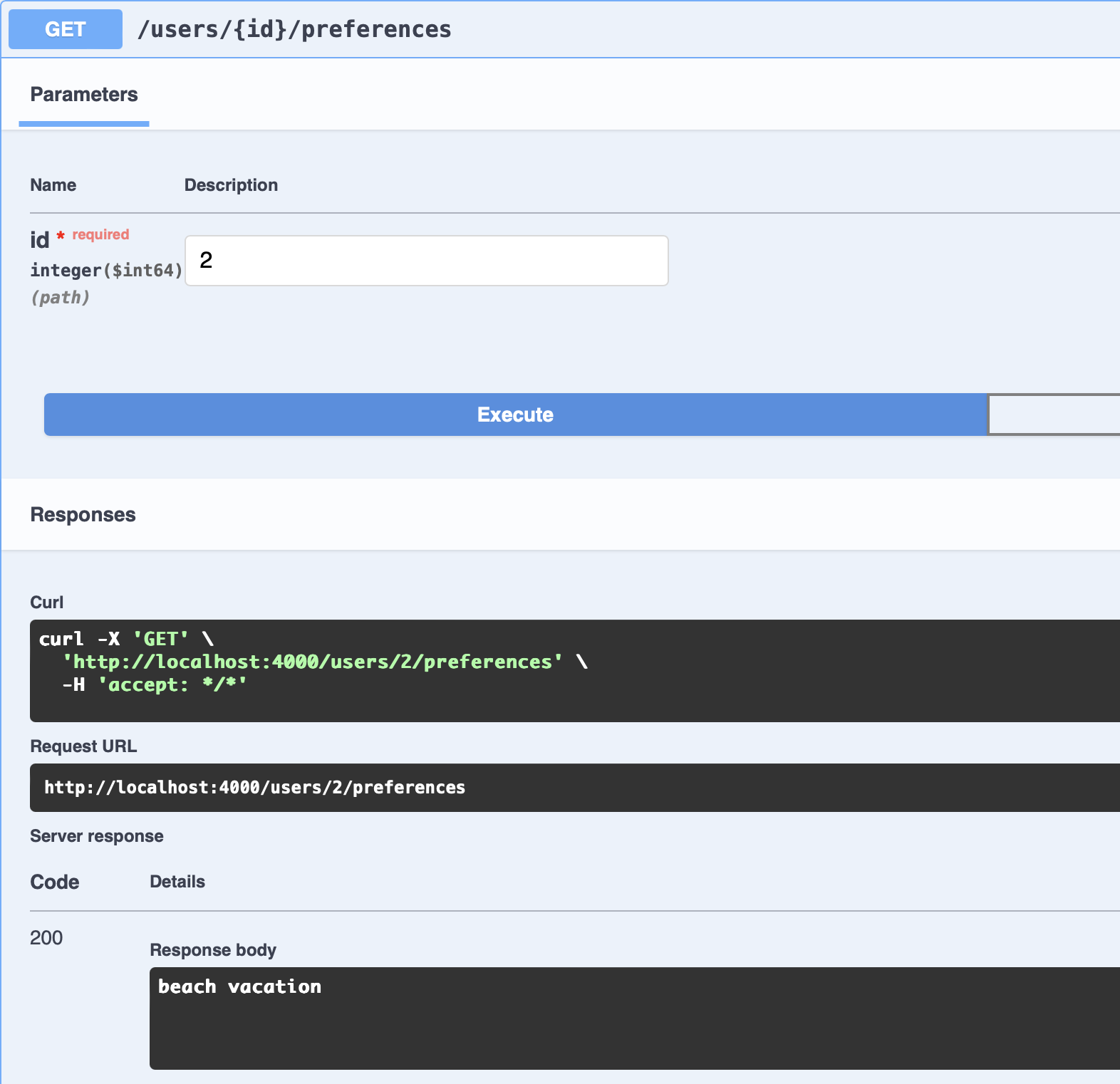
1. **Update User profile -> can change name, preferences and email**

****

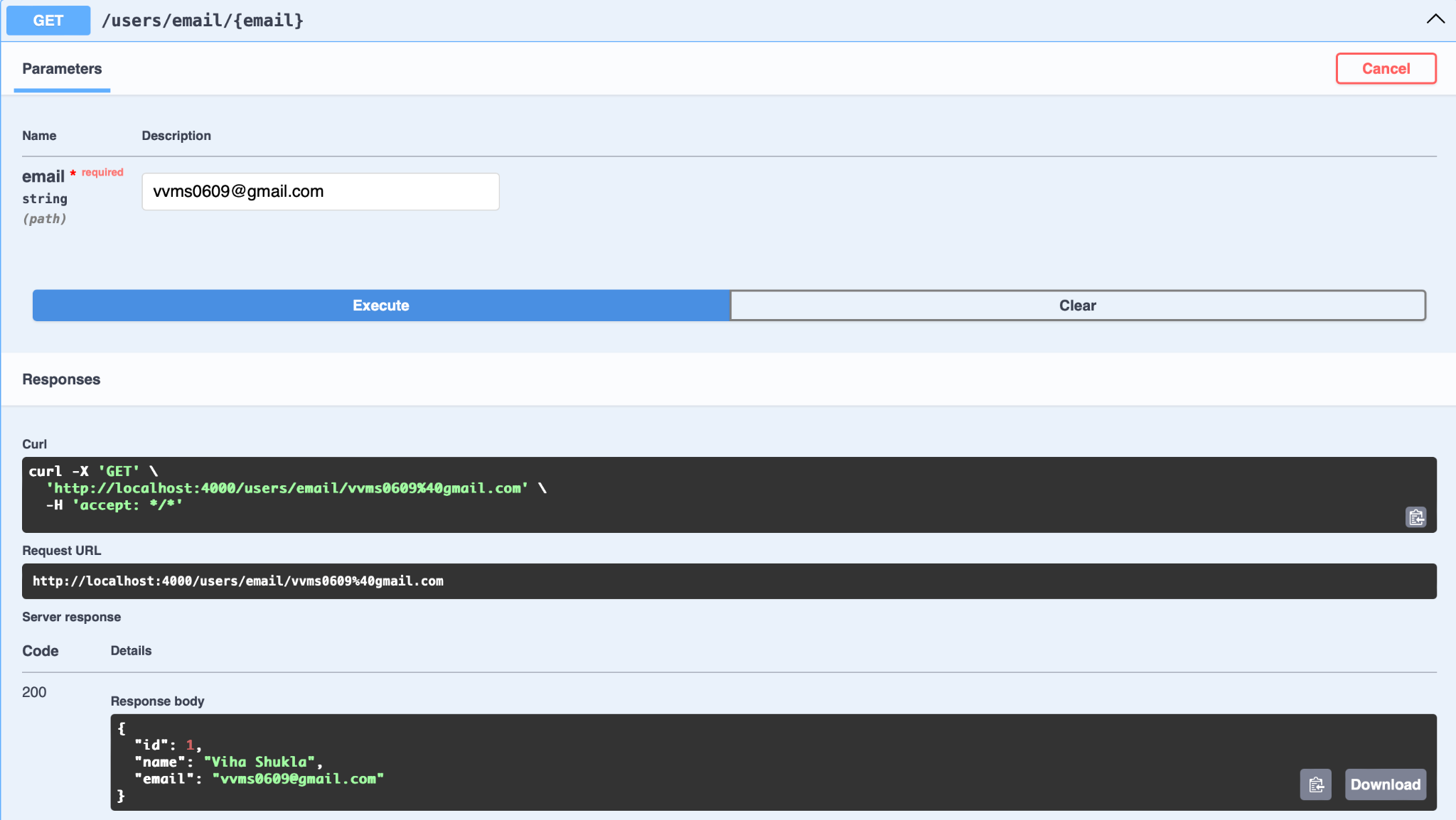
**Response**

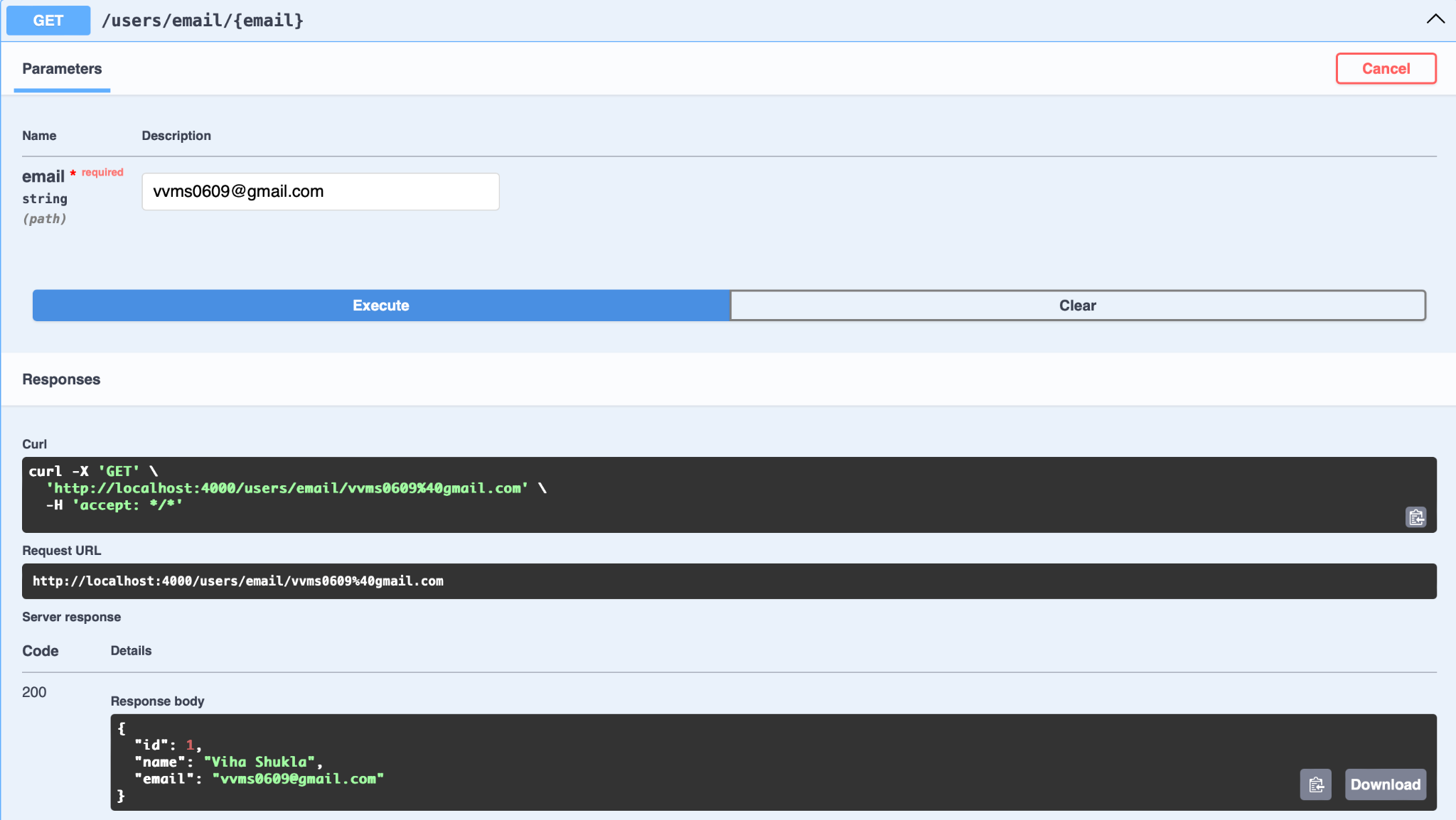
****

1. **Get User preferences by id**

****

1. **Get user by email**

****

****

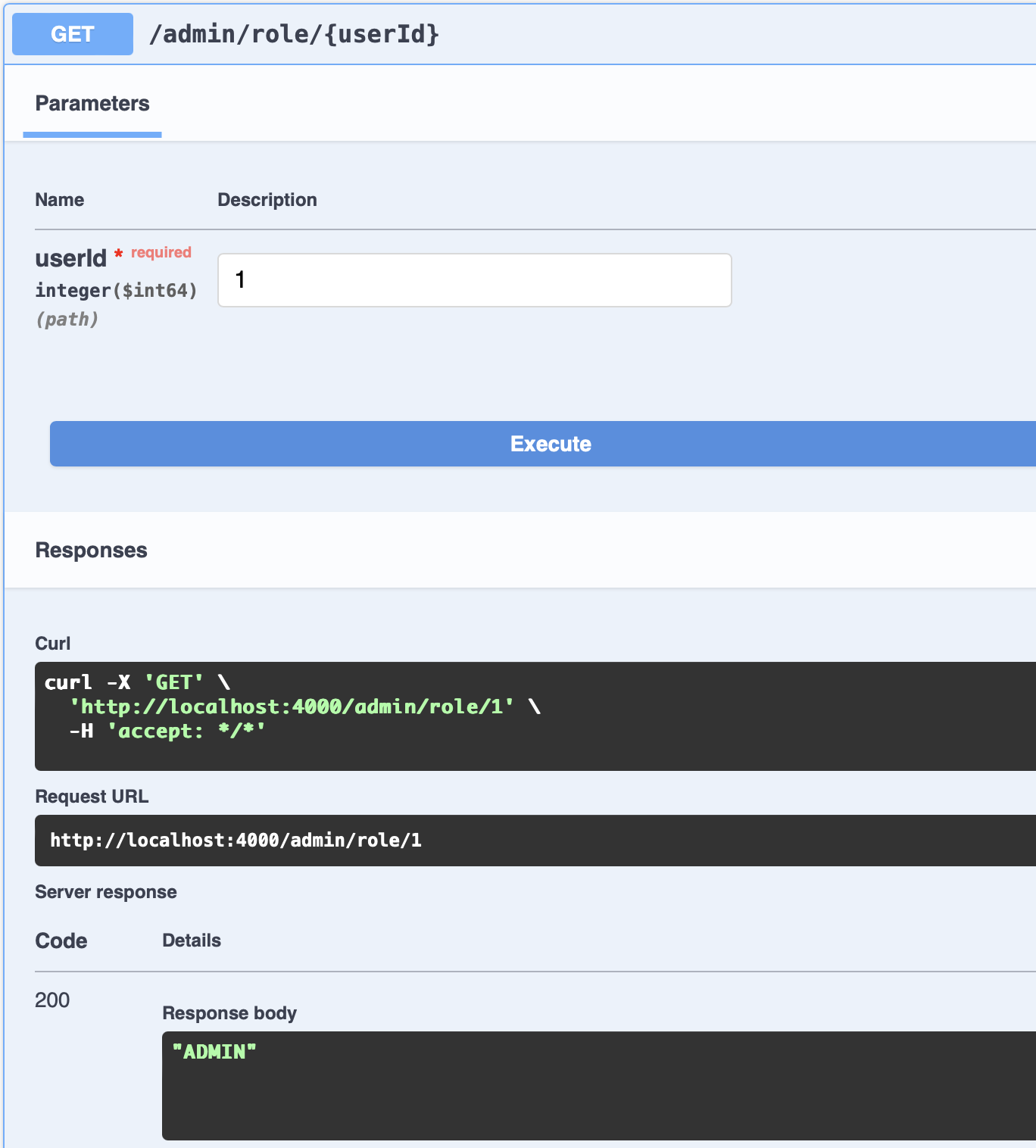
**AdminController**

**HTTP REQUESTS**

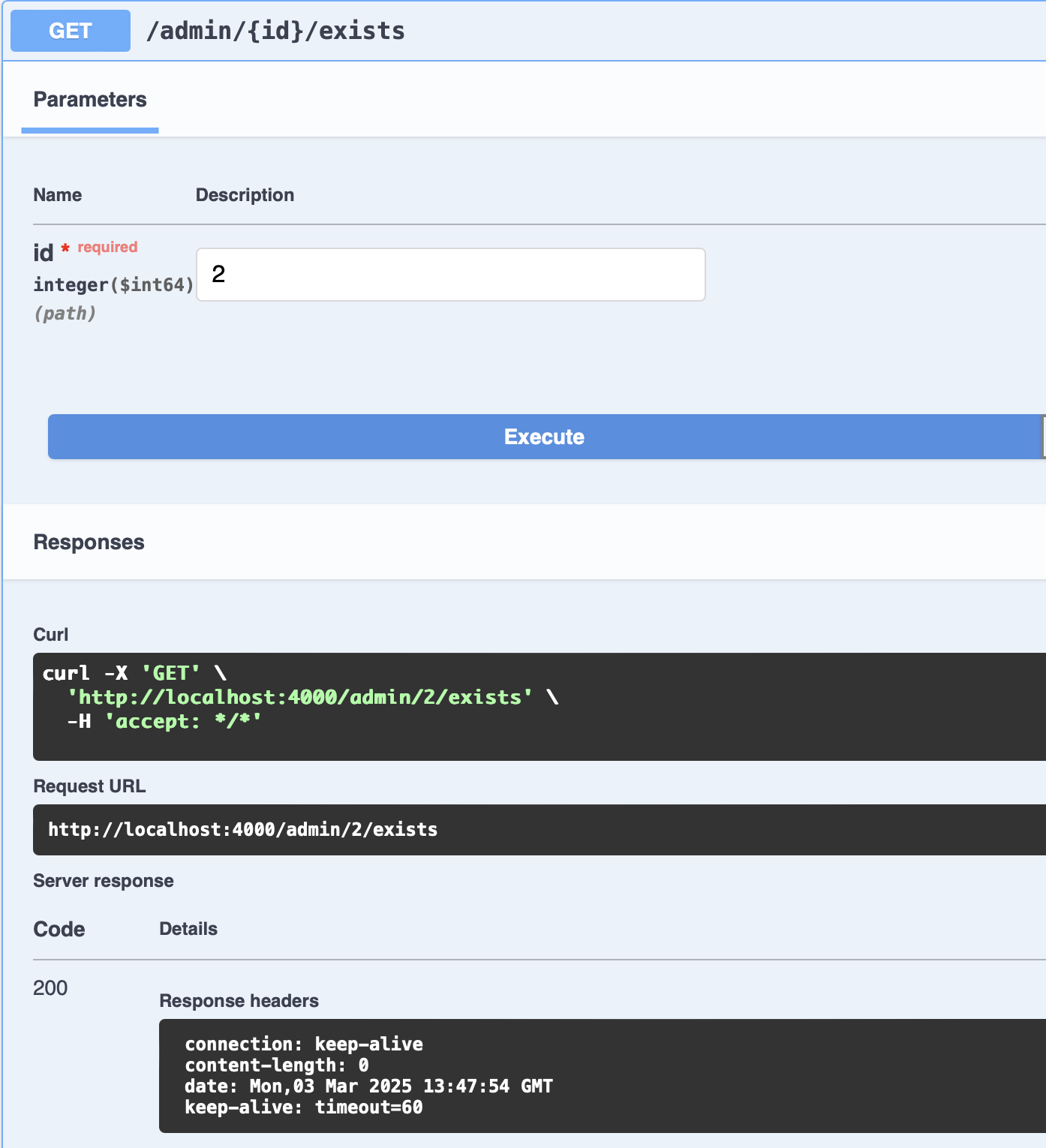
| **HTTP Method** | **Endpoint** | **Request Parameters/Body** | **Functionality** | **Response** |
| --- | --- | --- | --- | --- |
| **GET** | /admin/paged | Get a paginated list of users (excluding admins). | page (int), size (int) | 200 OK (Page of Users) |
| **GET** | /admin/{id}/preferences | Get a user's travel preferences. | id (User ID, Long) | 200 OK (Preferences) / 404 Not Found |
| **GET** | /admin/{id}/exists | Check if a user exists. | id (User ID, Long) | 200 OK (User exists) / 404 Not Found |
| **GET** | /admin/role/{userId} | Get a user's role. | userId (Long) | 200 OK (Role) / 404 Not Found |
| **PUT** | /admin/promote/{userId} | Promote a user to admin. | userId (Long) | 200 OK ("User promoted to ADMIN.") |
| **PUT** | /admin/enable/{userId} | Enable a disabled user account. | userId (Long) | 200 OK ("User enabled.") |
| **PUT** | /admin/disable/{userId} | Disable a user account. | userId (Long) | 200 OK ("User disabled.") |

**Output →** using Swagger-ui

1. **Get role by entering id**

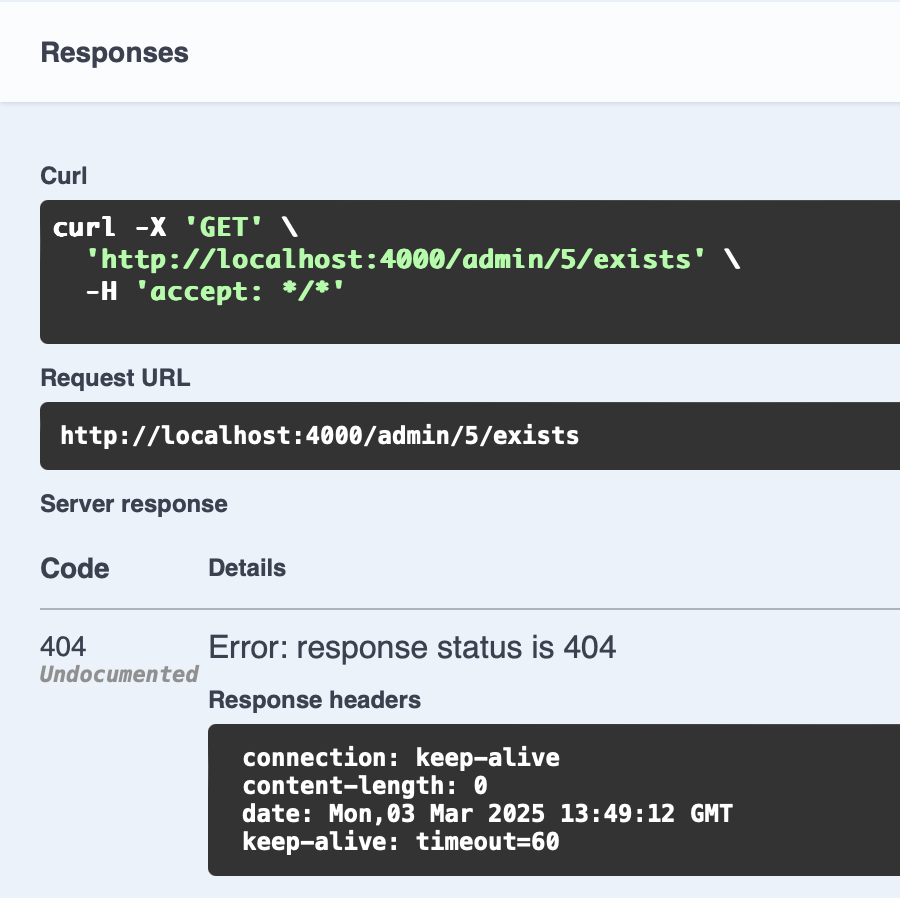


1. **Check is user exists by id**

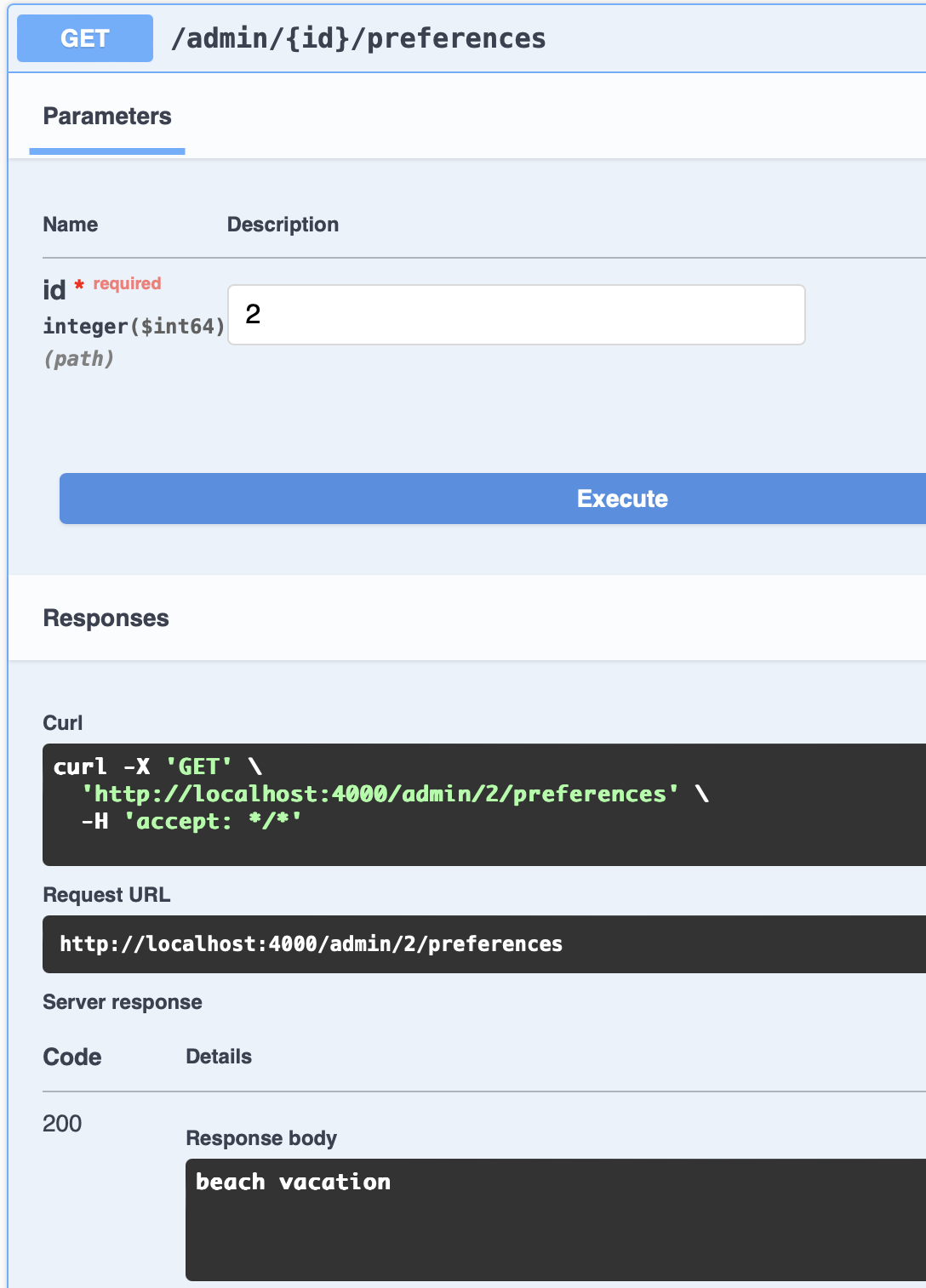




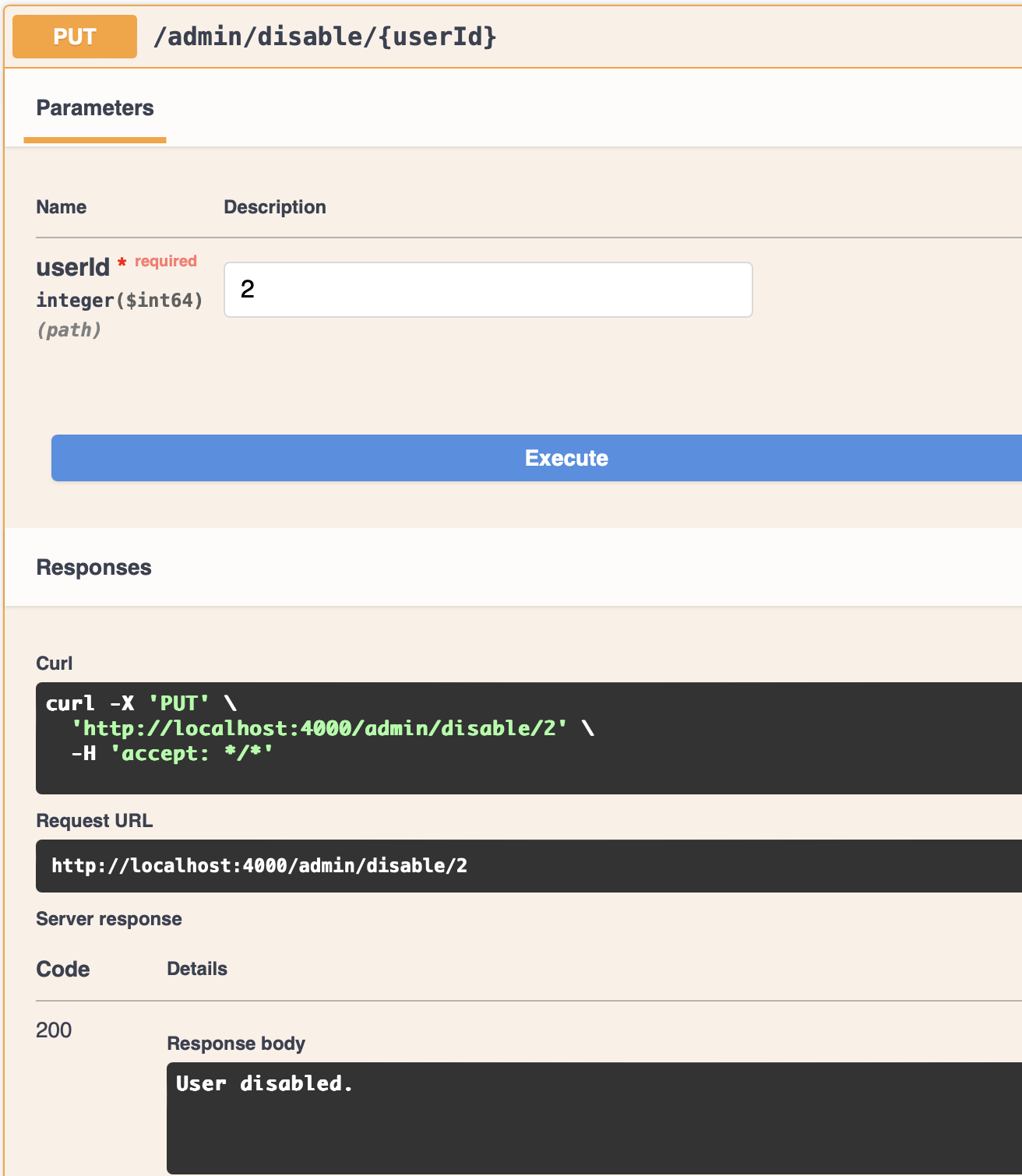
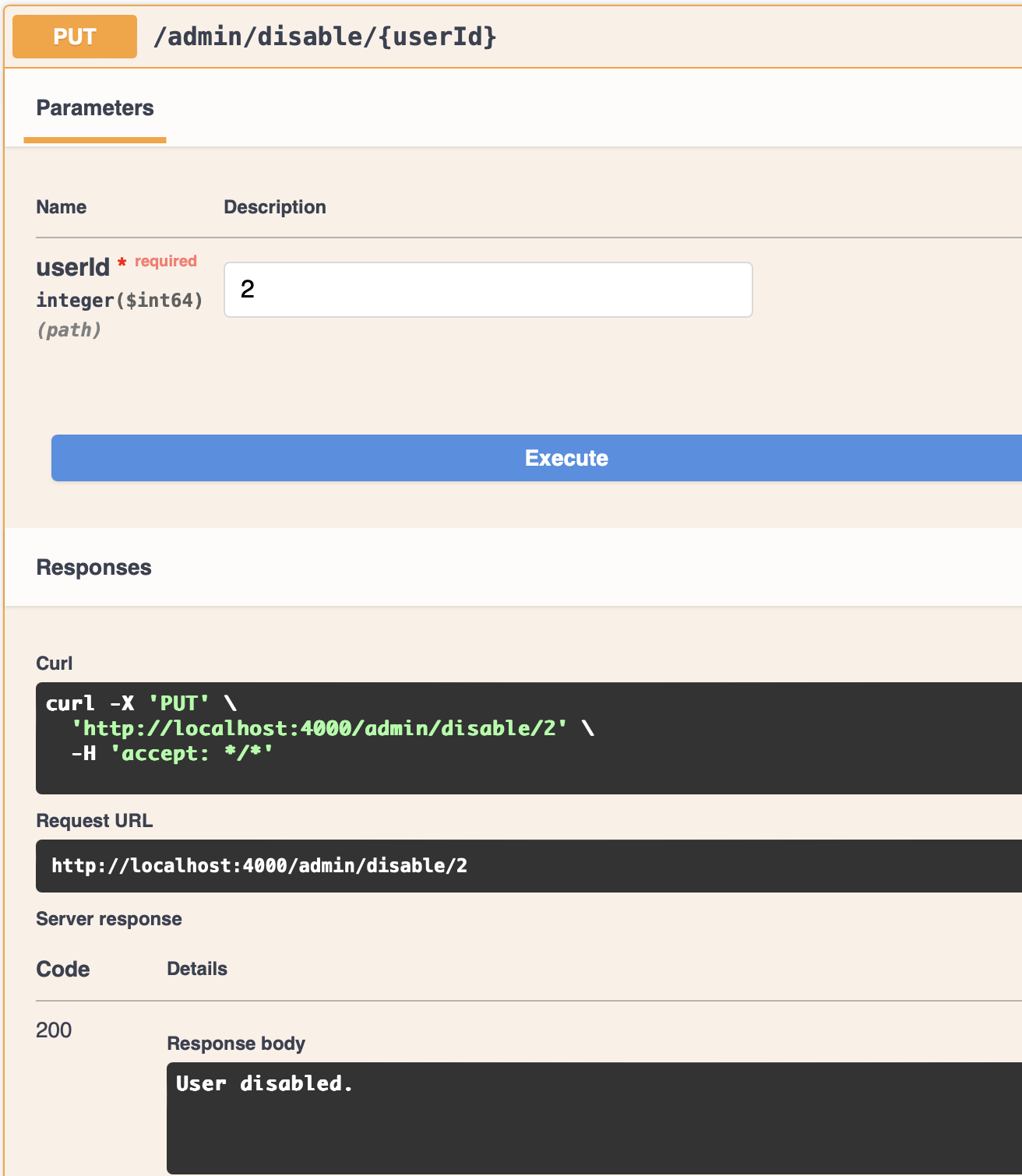
For user who doesn’t exists -> shows 404 response



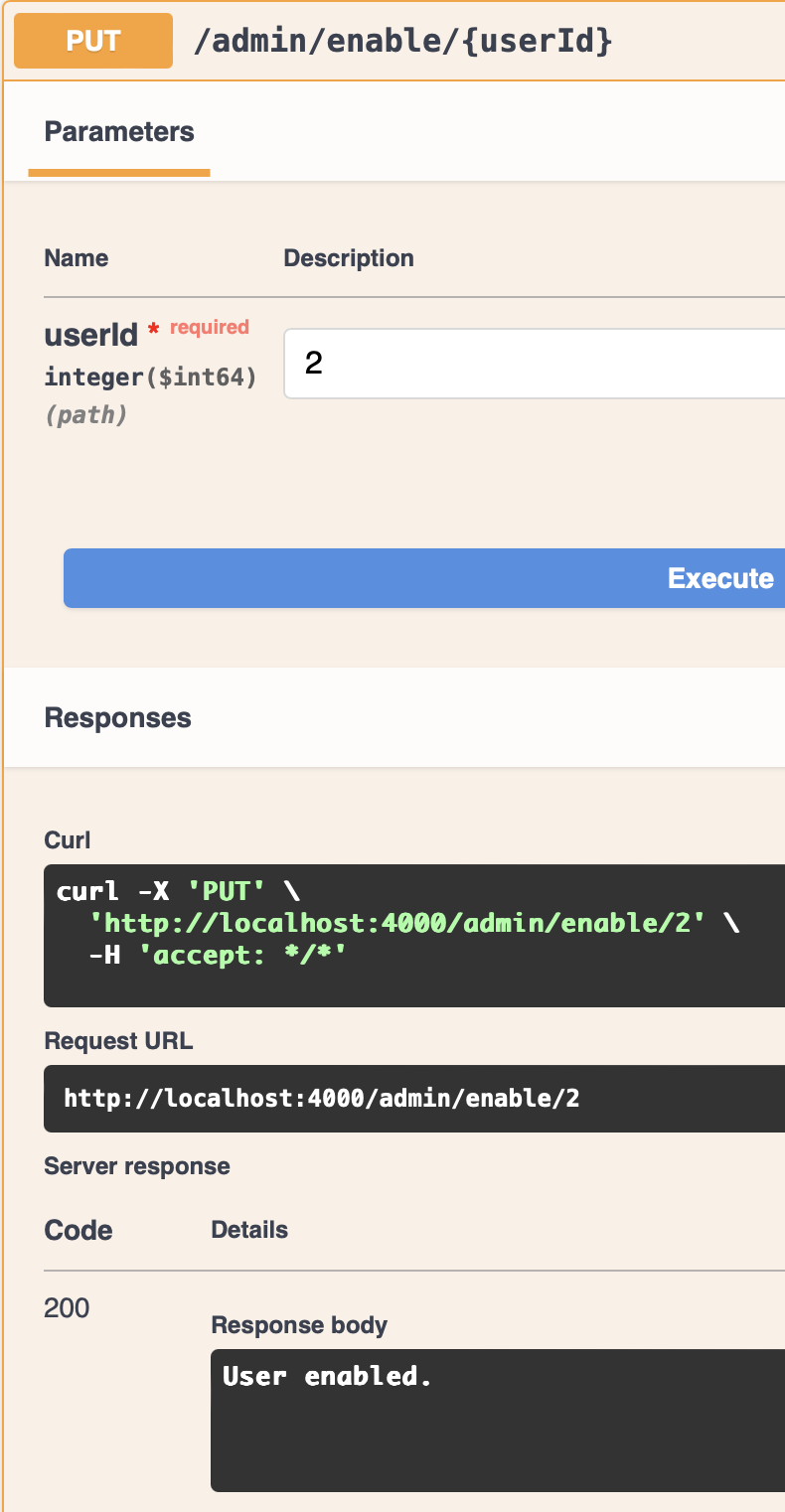
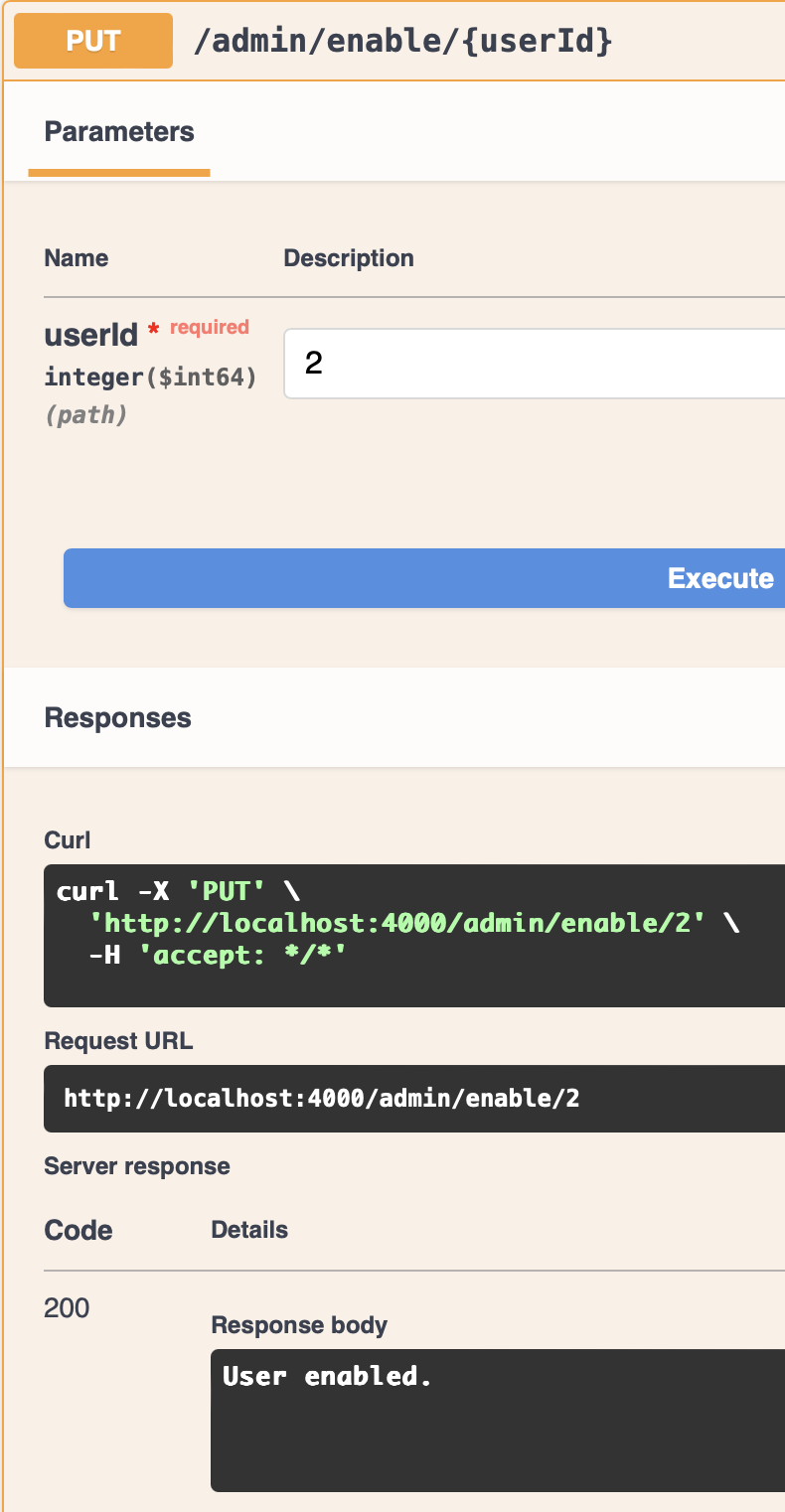
1. **Get user preferences by id**



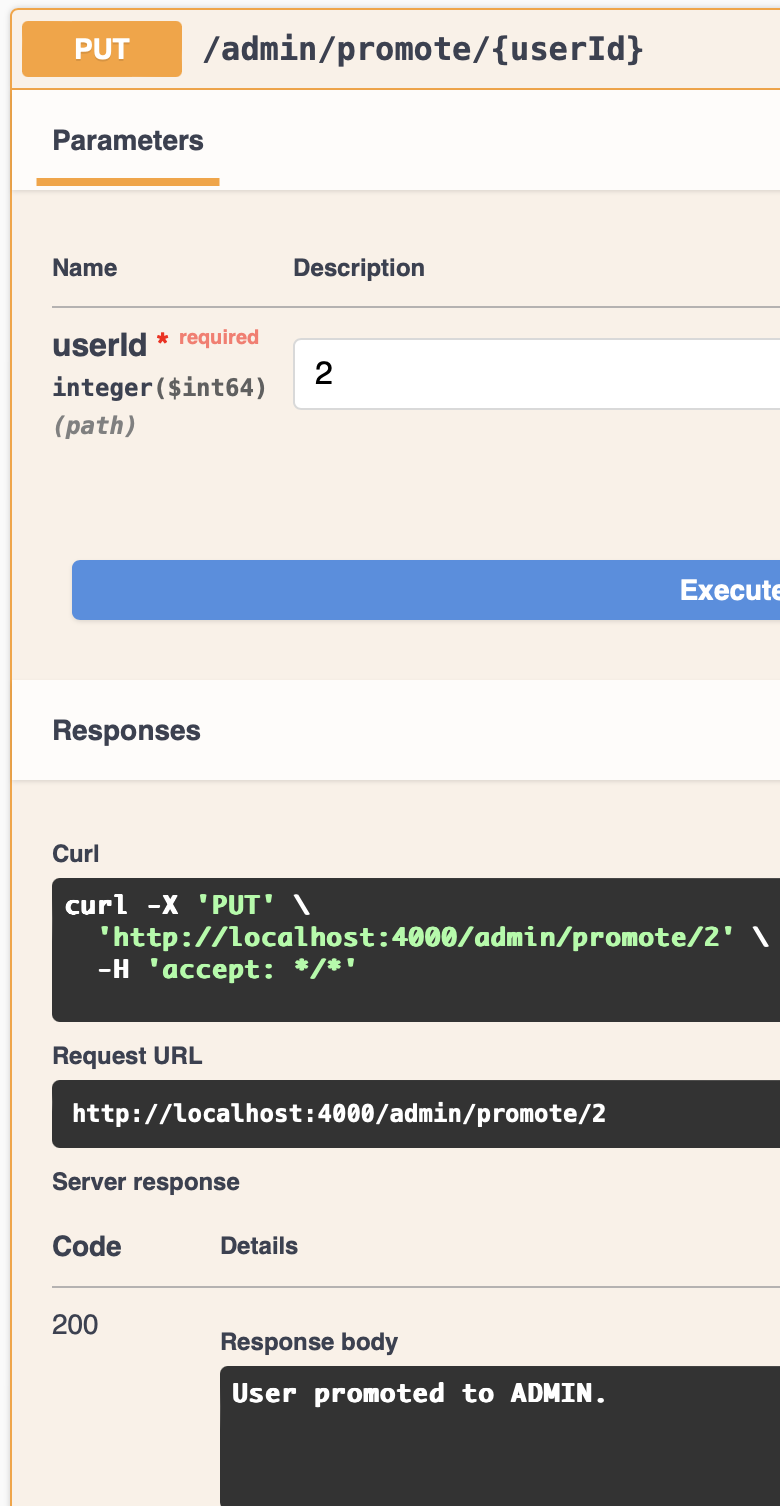
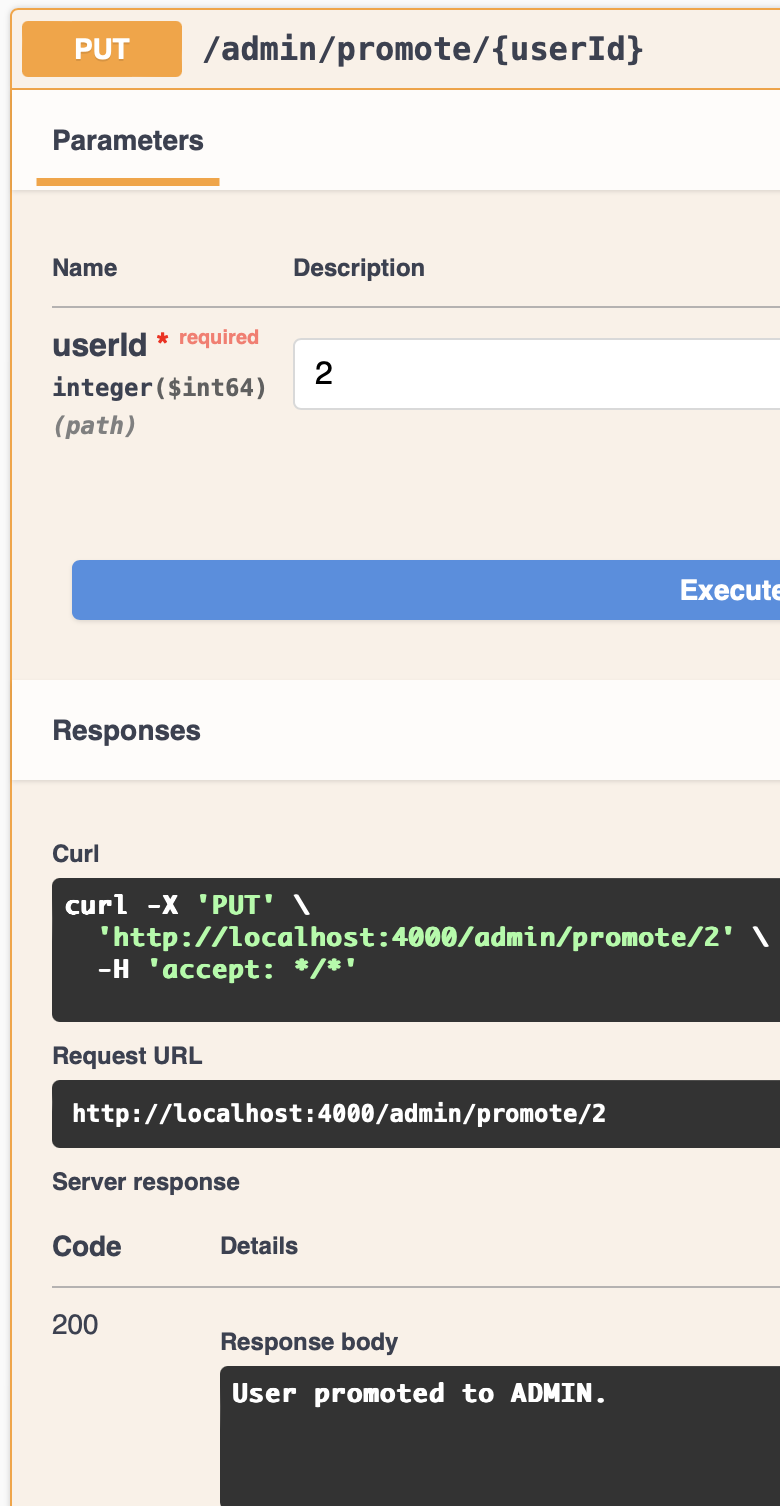
1. **Disable a user’s account**



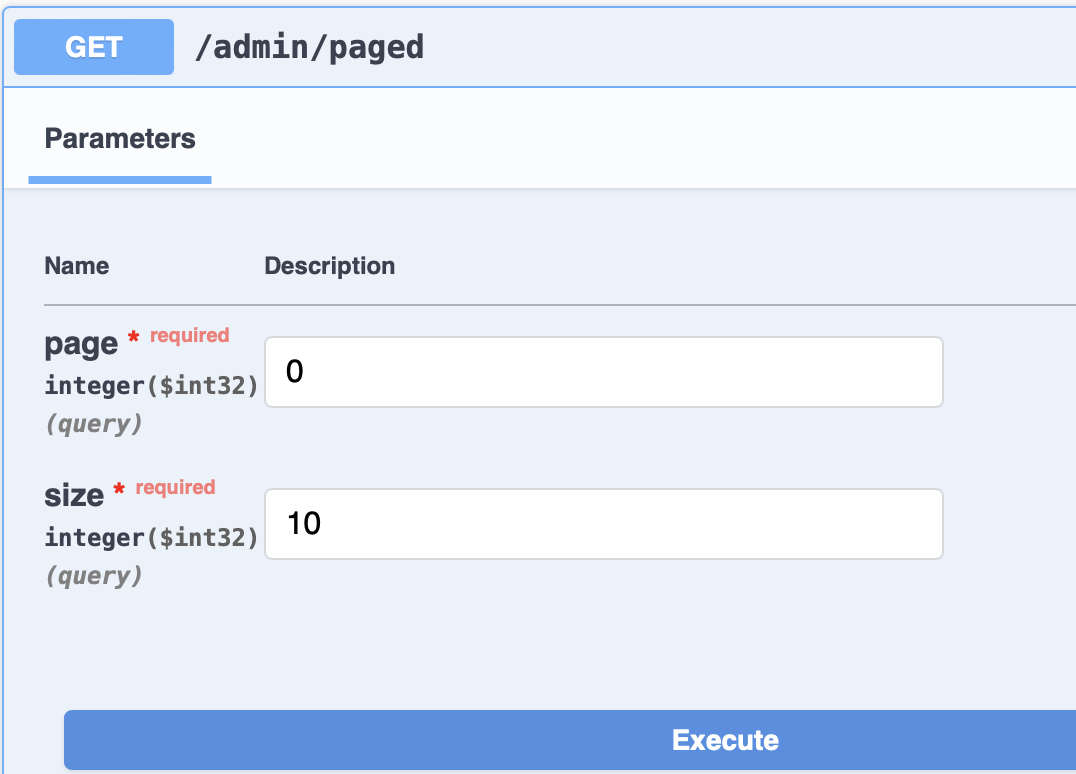
1. **Enable user’s account**

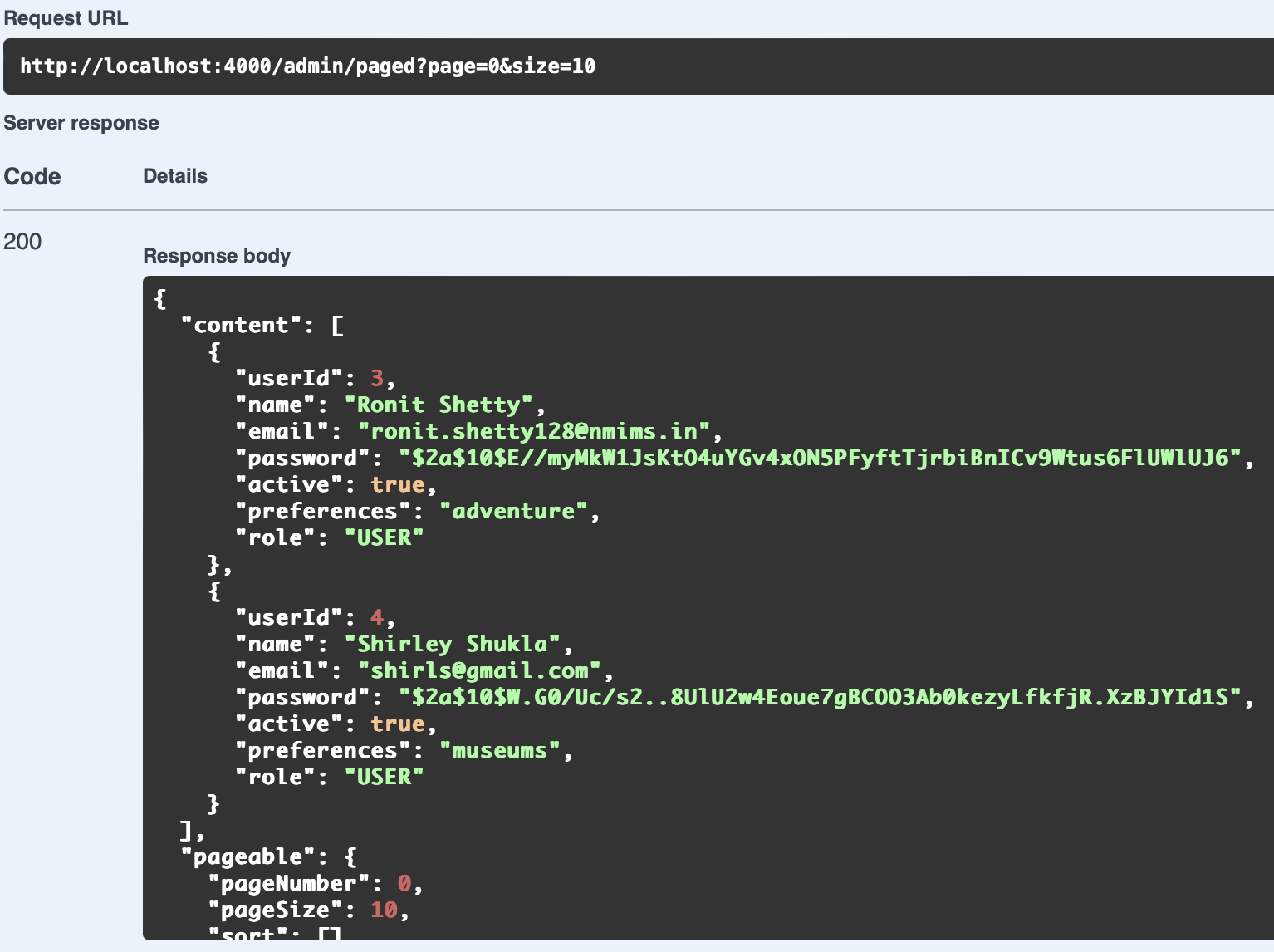


1. **Promoting user to admin**

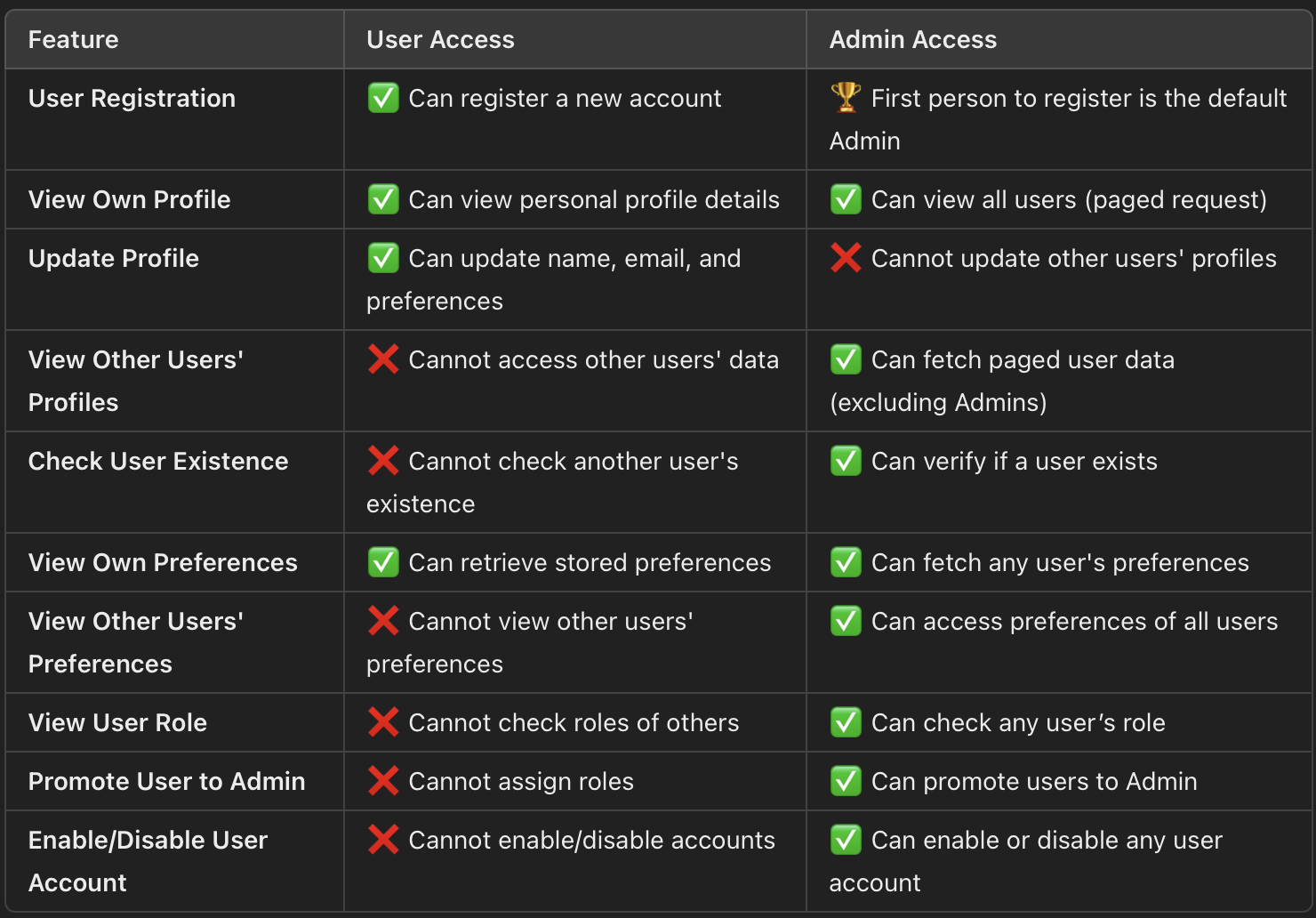


1. **Get a paginated list of users (excluding admins)**

****



**User Vs Admin Access**



**Itinerary Microservice -** handles itinerary creation, retrieval, deletion, and management while enforcing role-based access control. It interacts with the User Microservice to validate users and their roles.

| **METHODS** | | | |
| --- | --- | --- | --- |
| **Method Name** | **Purpose** | **How It Works** | **Access Control** |
| **createItinerary(Long userId, Itinerary itinerary)** | Creates a new itinerary for a user | - Calls User Microservice to get user details.  - Checks if the user is a "USER".  - Saves the itinerary if the check passes. | Admins cannot create itineraries  Users can create itineraries |
| **getItineraryById(Long id, Long userId)** | Fetches an itinerary by its ID | - Retrieves the itinerary from the database.  - Calls User Microservice to get requester details.  - Allows access only if the requester is an Admin or the itinerary owner. | Users cannot access itineraries of other users  Admins can access all itineraries |
| **getItinerariesByUserId(Long userId)** | Fetches all itineraries of a user | - Calls the repository to retrieve itineraries for the given user. | Users can see only their own itineraries  Admins can access all users' itineraries |
| **checkItineraryExists(Long id)** | Checks if an itinerary exists | - Calls existsById(id) in the repository and returns true/false. | Accessible to all users/admins |
| **getUserIdsByDestination(String destination)** | Retrieves user IDs of those who have itineraries for a specific destination | - Finds itineraries for the destination.  - Extracts unique userIds. | Admins can use this to see which users have been to a location |
| **findItinerariesByUserIdAndDestination(Long userId, String destination)** | Fetches itineraries for a specific user and destination | - Calls the repository to filter results by userId and destination. | Users can only see their own itineraries  Admins can view any user's itineraries |
| **deleteItinerary(Long id, Long userId)** | Deletes an itinerary if the requester is the owner or an Admin | - Retrieves itinerary details.  - Calls User Microservice to verify the requester's role.  - Deletes the itinerary if authorized. | Users cannot delete other users' itineraries  Admins can delete any itinerary |
| **getAllItineraries(Long adminId)** | Fetches all itineraries (Admin only) | - Calls User Microservice to verify if the requester is an Admin.  - Fetches all itineraries from the repository. | Users cannot access all itineraries  Admins can access all itineraries |

**HTTP REQUESTS**

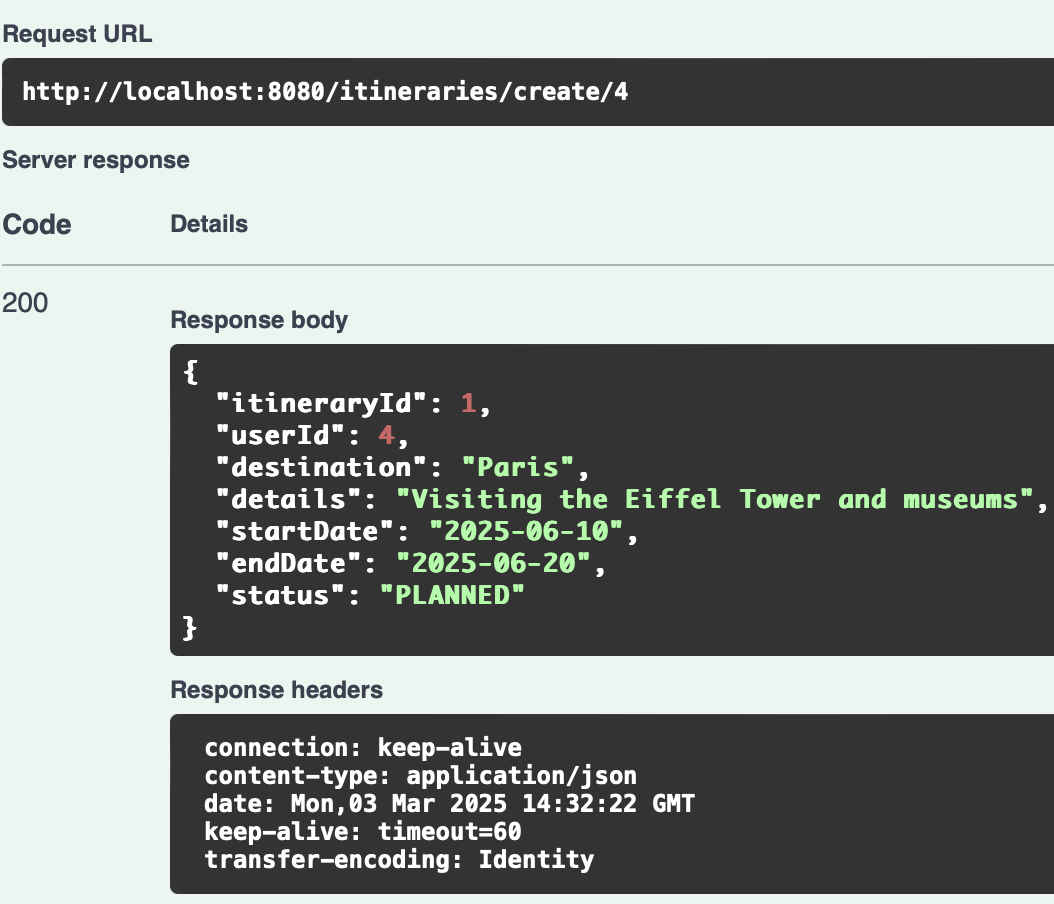
| **HTTP Method** | **Endpoint** | **Request Parameters/Body** | **Functionality** | **Response** |
| --- | --- | --- | --- | --- |
| **POST** | /itineraries/create/{userId} | userId (Path Variable), Itinerary (Request Body) | Create a new itinerary (Only for Users). | 200 OK (Created Itinerary) / 403 Forbidden (If not a User) |
| **GET** | /itineraries/{id} | id (Path Variable), userId (Query Param) | Retrieve an itinerary by ID (User can access only their own, Admins can access all). | 200 OK (Itinerary) / 403 Forbidden / 404 Not Found |
| **GET** | /itineraries/user/{userId} | userId (Path Variable) | Get all itineraries for a specific user. | 200 OK (List of Itineraries) |
| **GET** | /itineraries/user/{userId}/{destination} | userId (Path Variable), destination (Path Variable) | Retrieve itineraries by user and destination. | 200 OK (List of Itineraries) |
| **GET** | /itineraries/{id}/exists | id (Path Variable) | Check if a specific itinerary exists. | 200 OK (true or false) |
| **GET** | /itineraries/users/{destination} | destination(Path Variable) | Retrieve user IDs with itineraries for a given destination. | 200 OK (List of User IDs) |
| **DELETE** | /itineraries/{id} | id (Path Variable), userId (Query Param) | Delete an itinerary (Only Admins or Owners can delete). | 200 OK (Success Message) / 403 Forbidden / 404 Not Found |
| **GET** | /itineraries/admin/{adminId} | adminId (Path Variable) | Retrieve all itineraries (Only Admins can access). | 200 OK (List of Itineraries) / 403 Forbidden |

**Output**

1. **Creating an itinerary for a user**



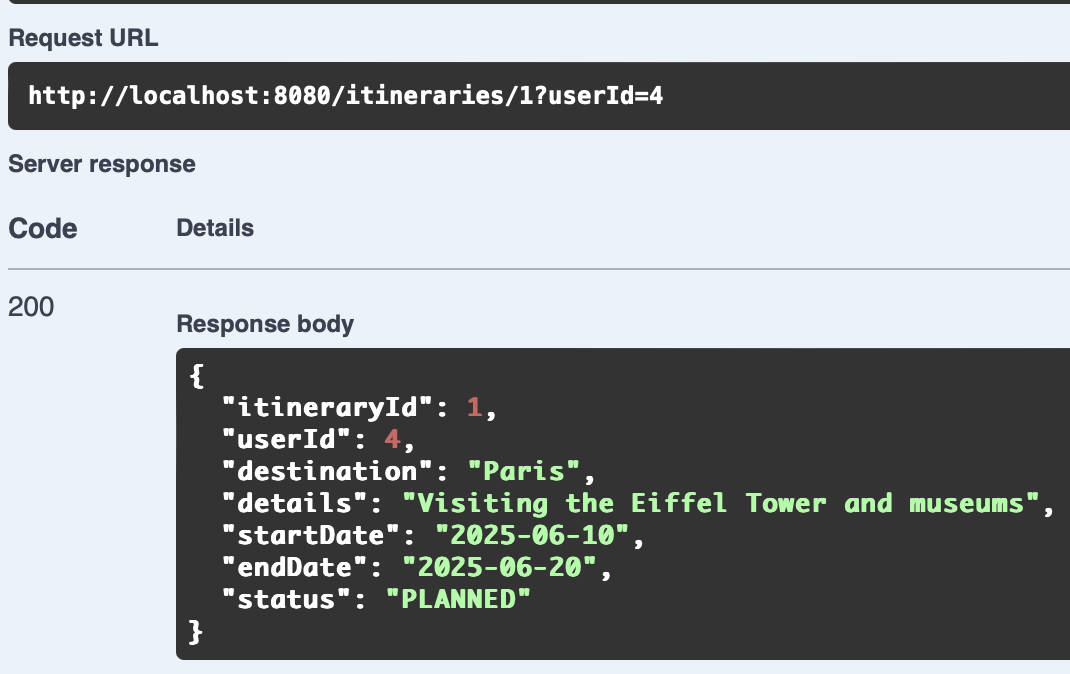
**Response**



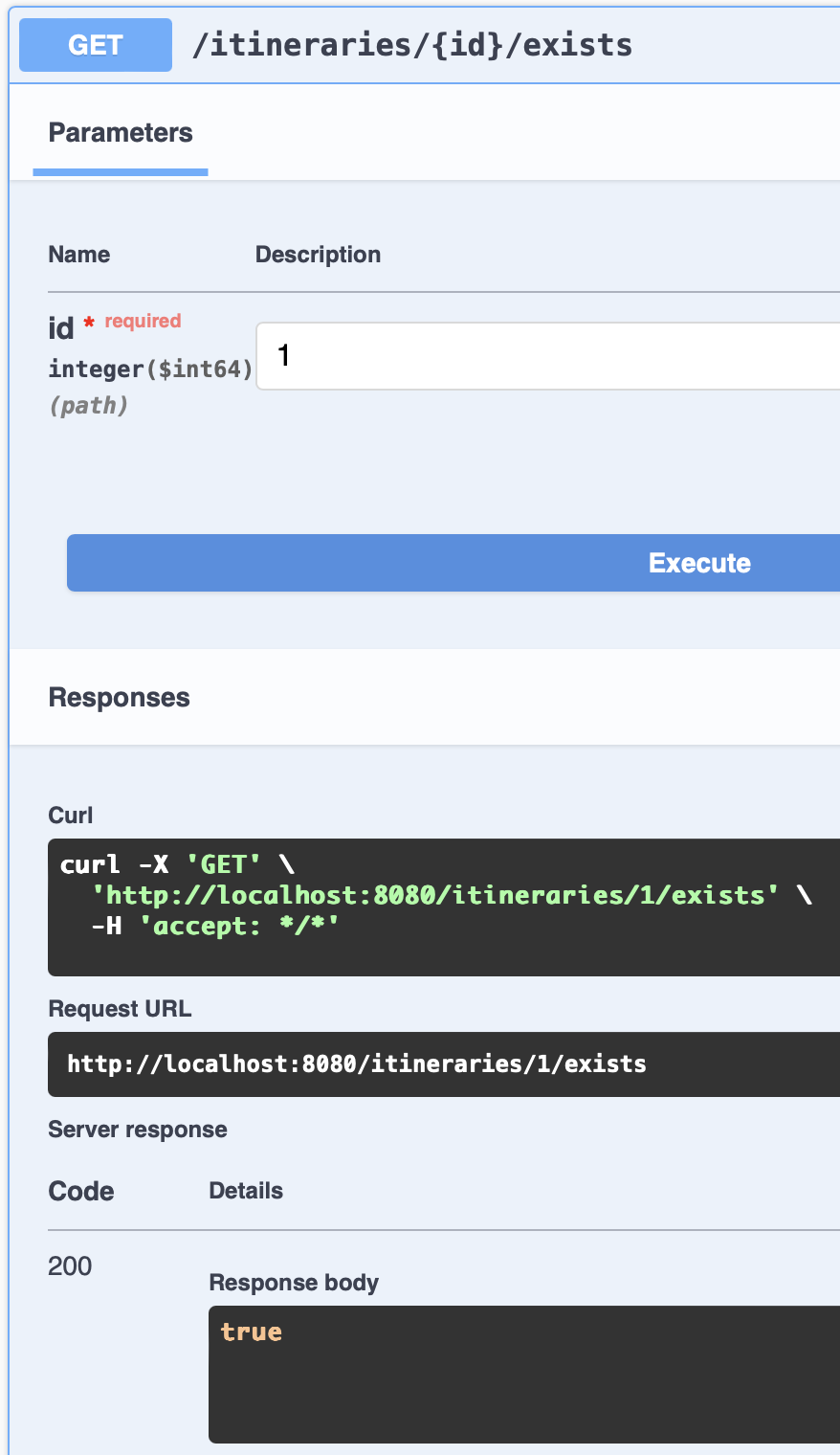
1. **Get itinerary by id and inputting userId (owner or admin)**



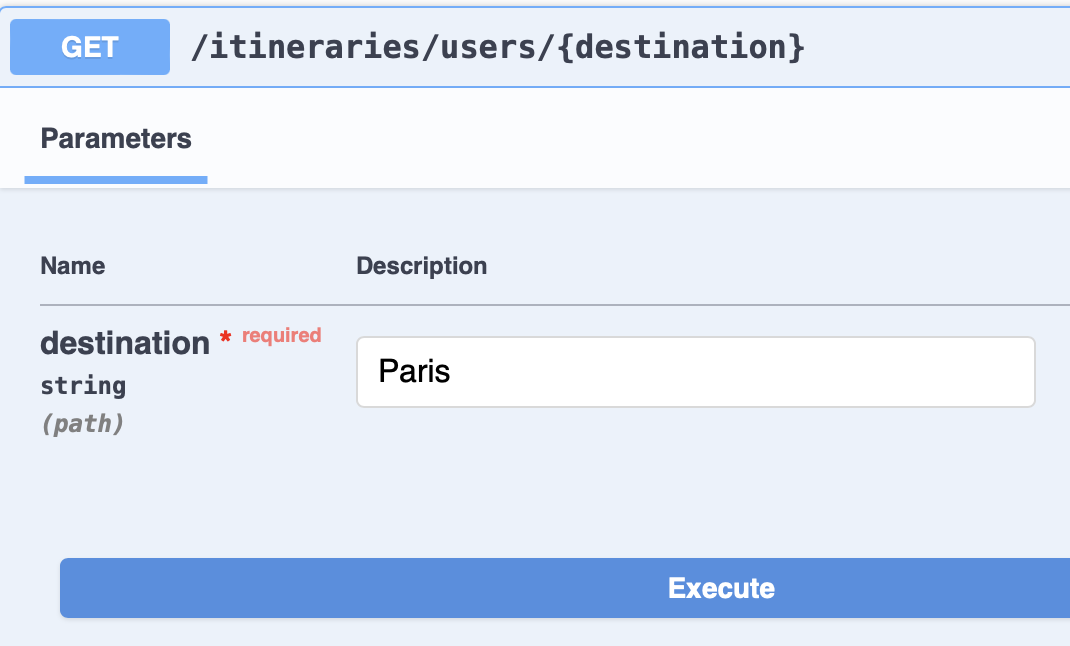
**Response**

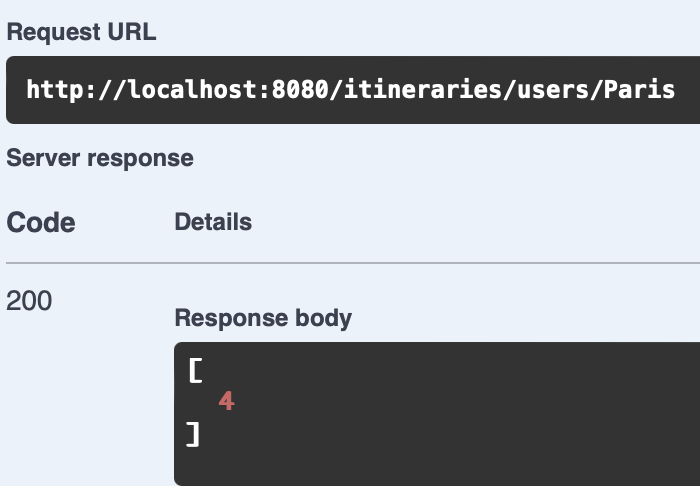


1. **Check if itinerary exists by Id**

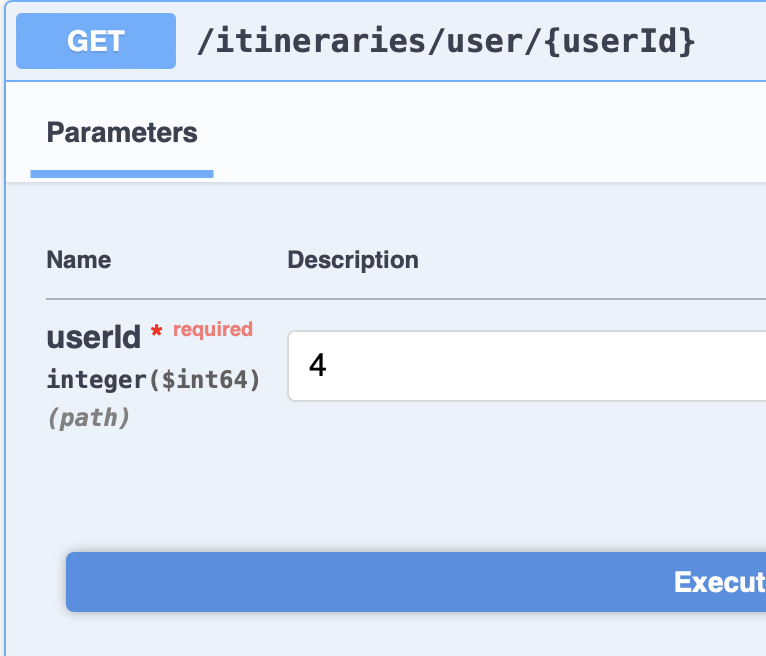


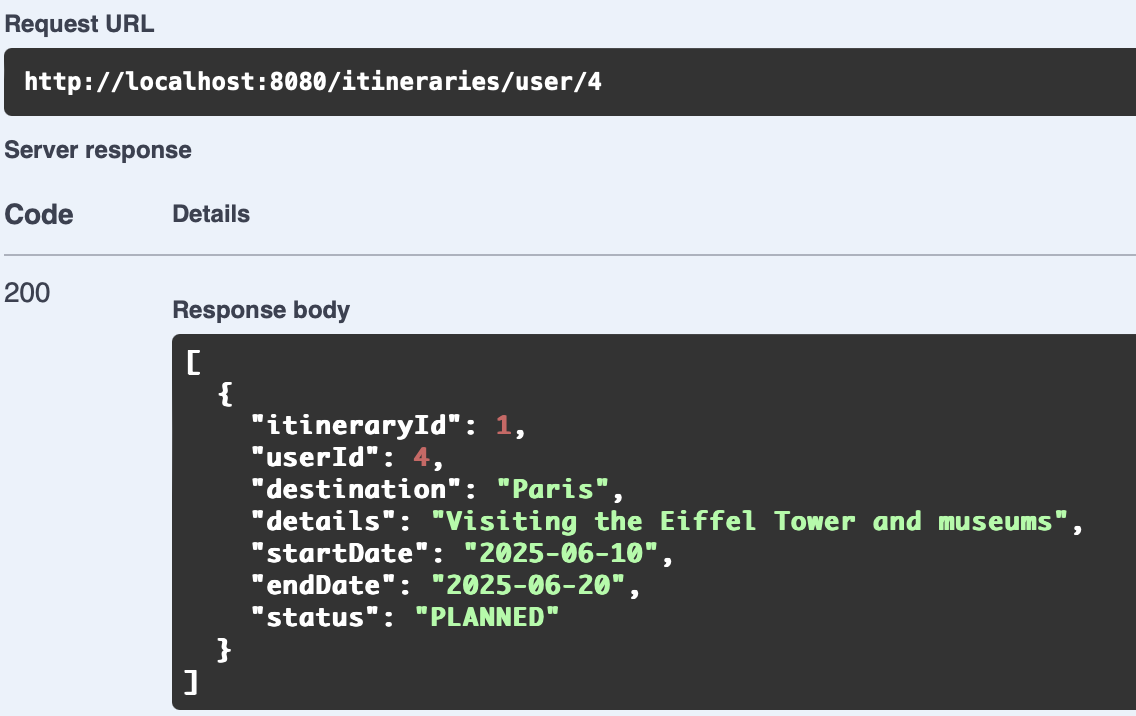
1. **Get user ids by destination**

****

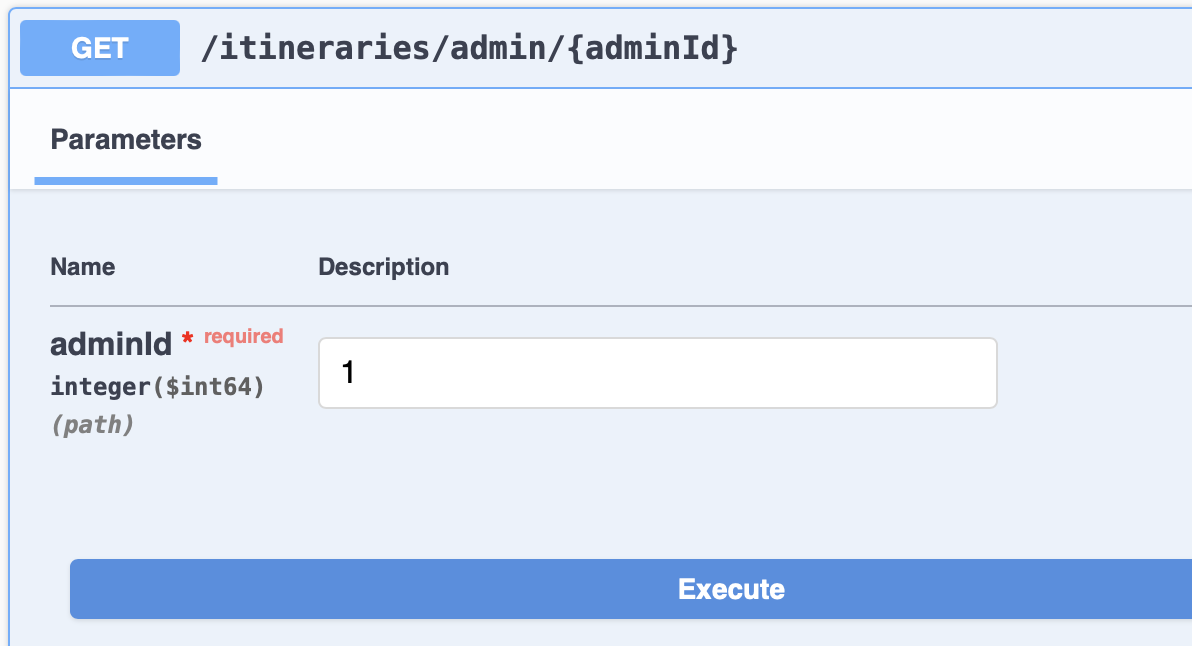
****

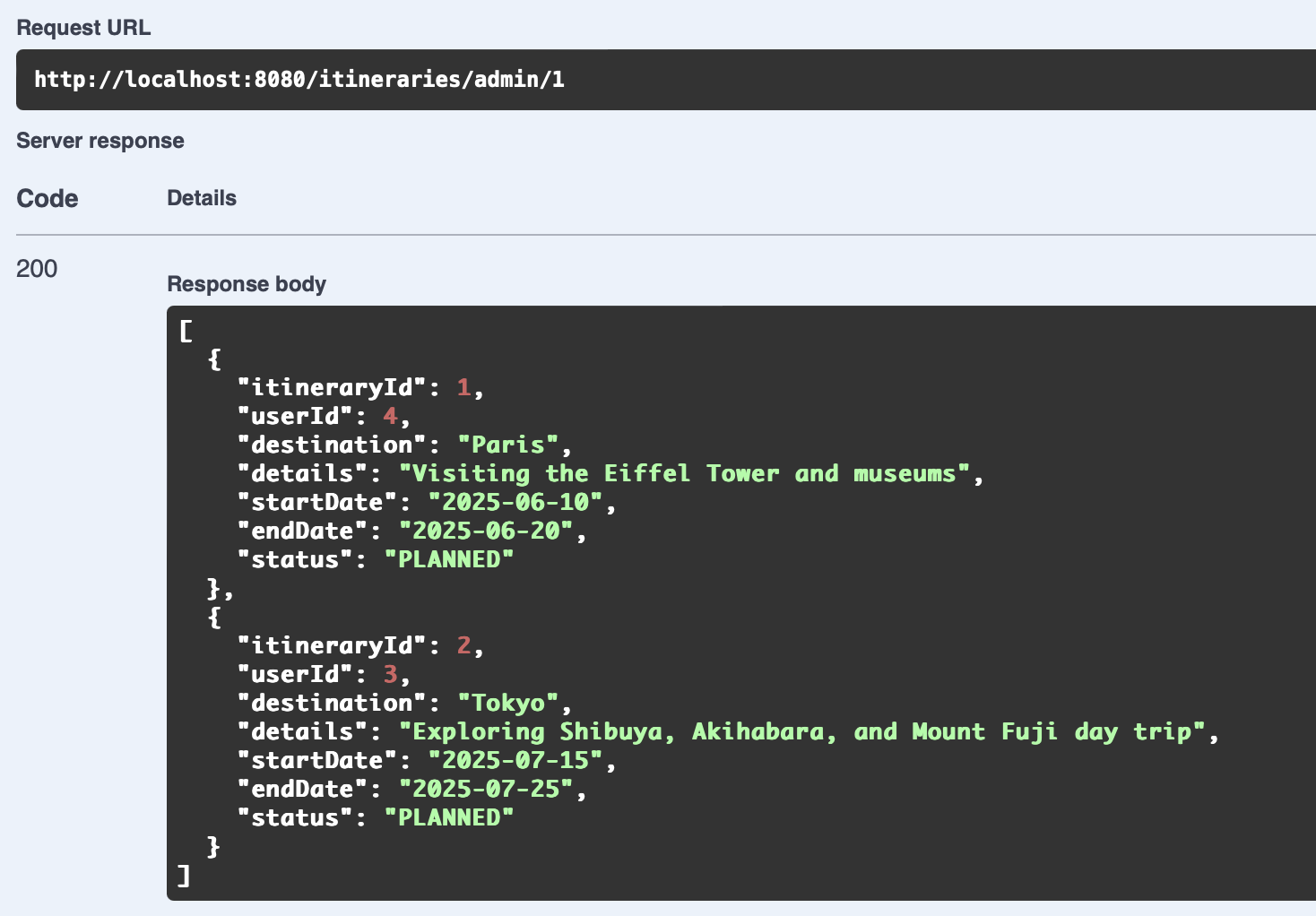
1. **Get itinerary by user id**

****

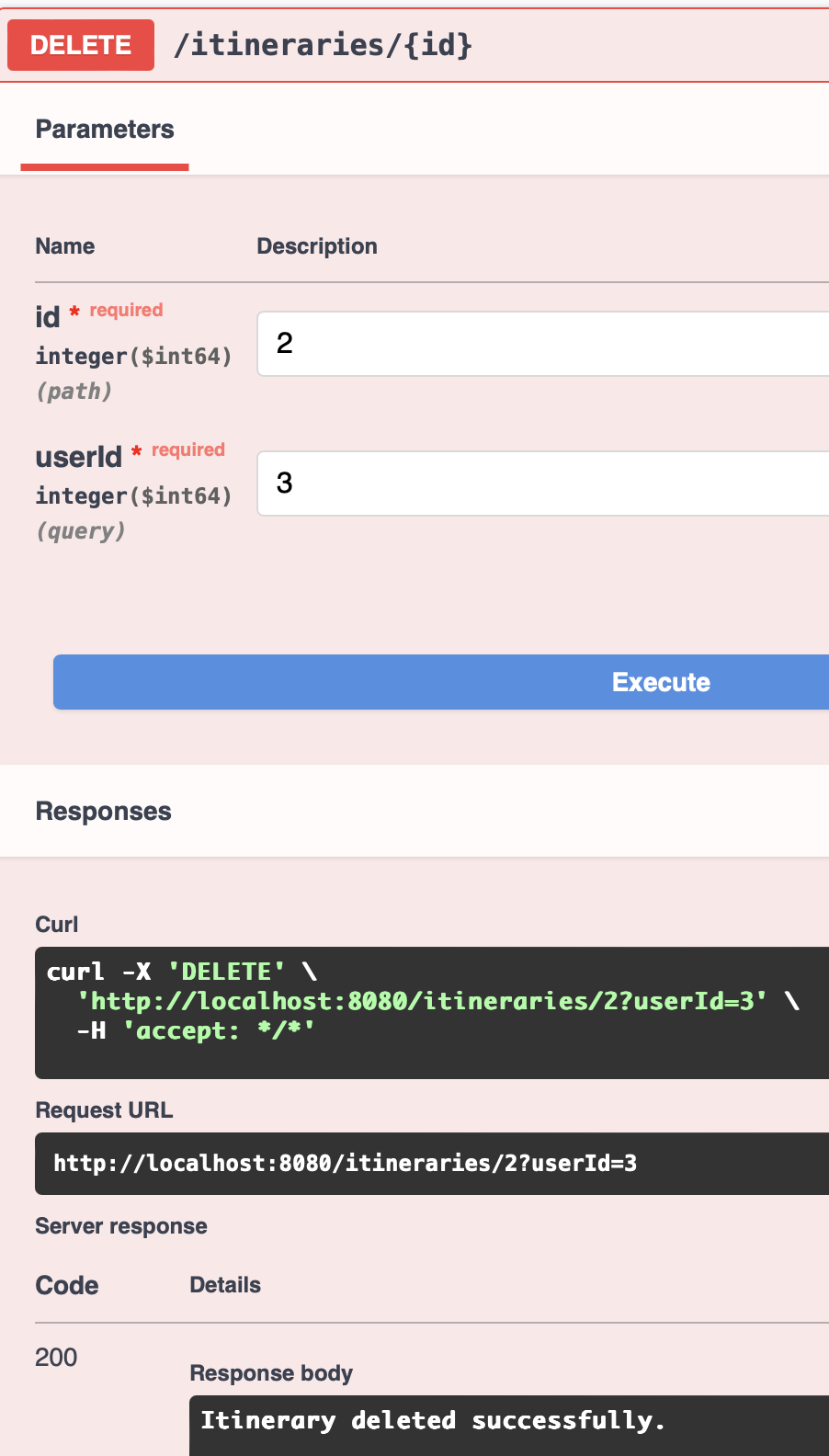
****

1. **Get all itineraries by using adminId**

****

****

1. **Delete an itinerary**

****

**Activity Microservice -** manages travel activities, allowing admins to create, retrieve, and delete activities while enforcing role-based access control. It interacts with the User Microservice to validate admin privileges.

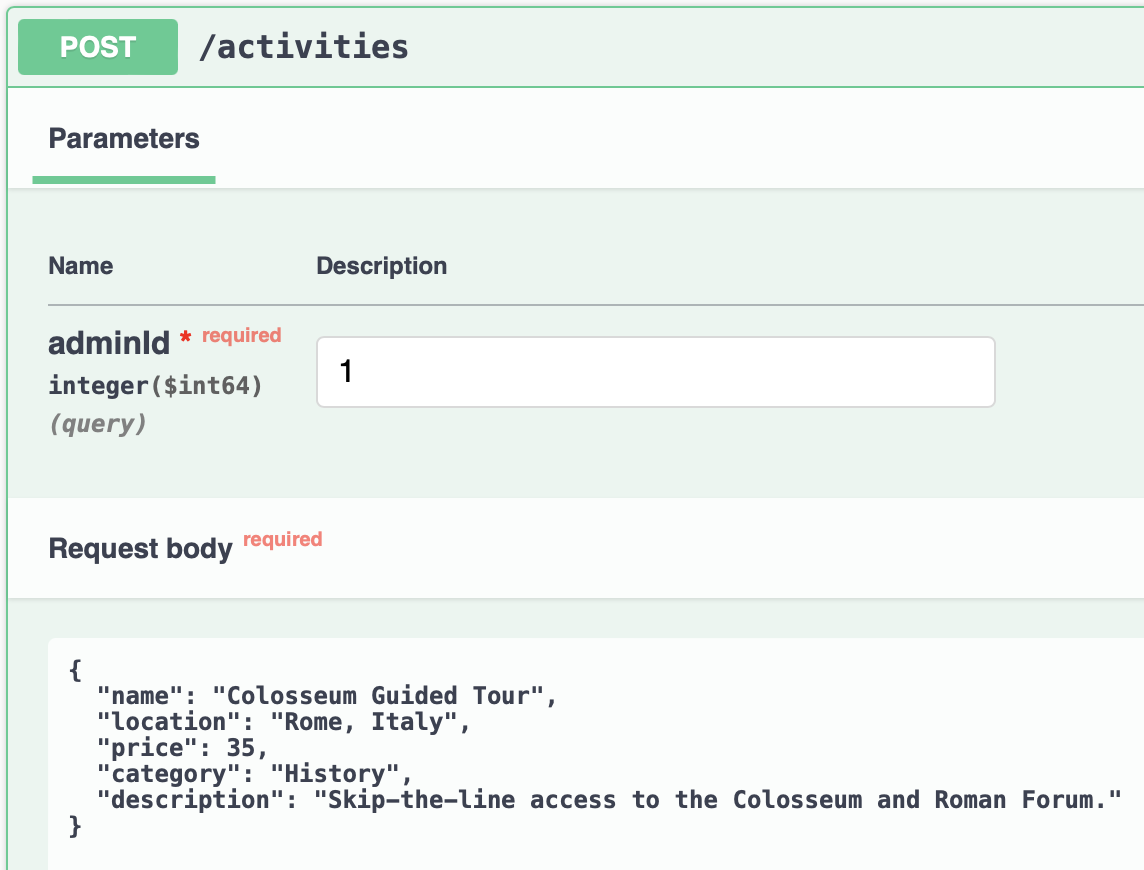
| **METHODS** | | | |
| --- | --- | --- | --- |
| **Method Name** | **Purpose** | **How It Works** | **Access Control** |
| **createActivity(Long adminId, Activity activity)** | Creates a new activity. | - Calls User Microservice to validate the admin.  - Checks if the user has an "ADMIN" role.  - Saves the activity in the database if validation passes. | Users cannot create activities.  Only Admins can create activities. |
| **getAllActivities()** | Retrieves all available activities. | - Fetches all activity records from the database. | Accessible to all users. |
| **checkActivityExists(Long id)** | Checks if an activity exists. | - Verifies if an activity with the given ID is present in the database. | Accessible to all users. |
| **getActivityById(Long id)** | Retrieves details of a specific activity. | - Searches for an activity by ID.  - Throws an error if not found. | Accessible to all users. |
| **getActivitiesByLocation(String location)** | Fetches activities available in a specific location. | - Queries the database for activities matching the given location. | Accessible to all users. |
| **getActivitiesByCategory(String category)** | Retrieves activities based on their category (e.g., Adventure, Culture). | - Searches for activities based on the specified category. | Accessible to all users. |
| **deleteActivity(Long id, Long adminId)** | Deletes an activity from the system. | - Calls User Microservice to validate the admin.  - Checks if the user has an "ADMIN" role.  - Deletes the activity if validation passes. | Users cannot delete activities.  Only Admins can delete activities. |

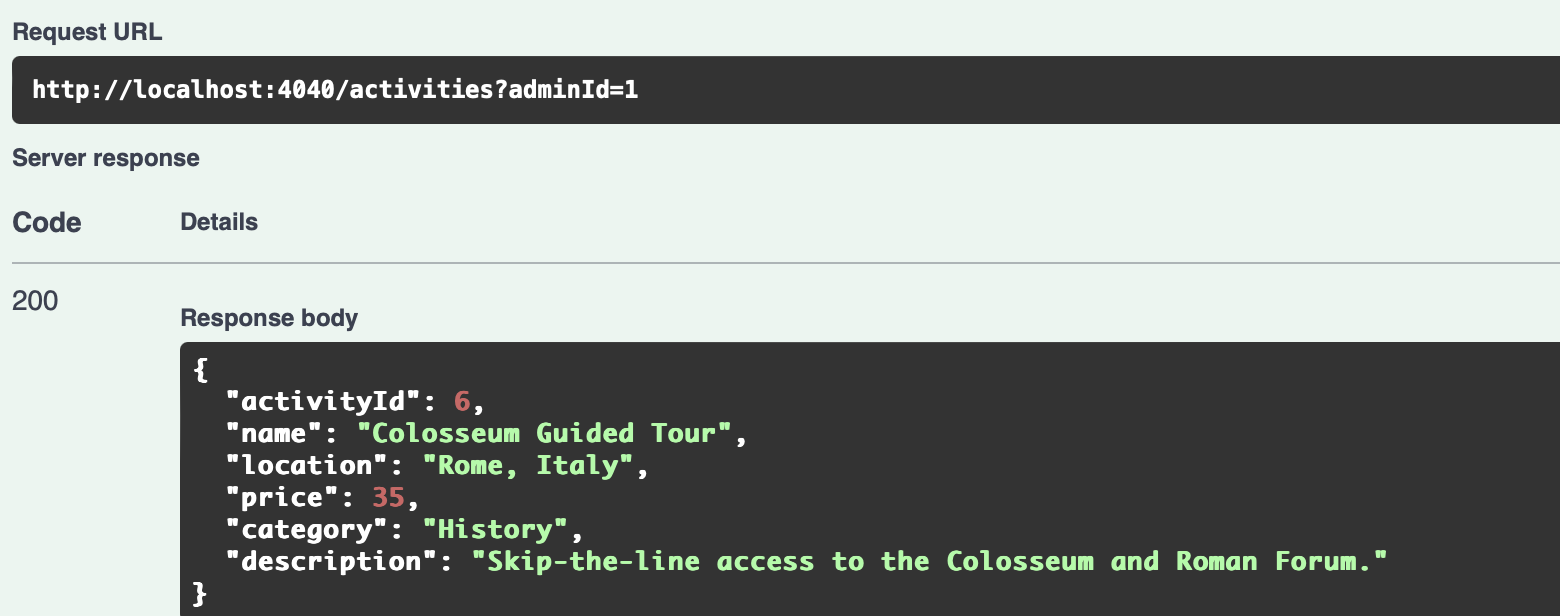
**HTTP REQUESTS**

| **HTTP Method** | **Endpoint** | **Request Parameters/Body** | **Functionality** | **Response** |
| --- | --- | --- | --- | --- |
| **POST** | **/activities** | adminId (Long) (Query Param), Activity(Request Body) | Create a new activity (Admin Only). | 200 OK (Created Activity) / 403 Forbidden (If not Admin) |
| **GET** | /activities/{id}/exists | id (Long) | Check if an activity exists. | 200 OK (true or false) |
| **GET** | /activities | None | Retrieve all activities. | 200 OK (List of Activities) |
| **GET** | /activities/{id} | id (Long) | Get activity details by ID. | 200 OK (Activity) / 404 Not Found |
| **GET** | /activities/location/{location} | location (String) | Retrieve activities by location. | 200 OK (List of Activities) |
| **GET** | /activities/category/{category} | category (String) | Retrieve activities by category. | 200 OK (List of Activities) |
| **DELETE** | /activities/{id} | id (Long) (Path Variable), adminId (Long) (Query Param) | Delete an activity (Admin Only). | 200 OK / 403 Forbidden(If not Admin) / 404 Not Found |

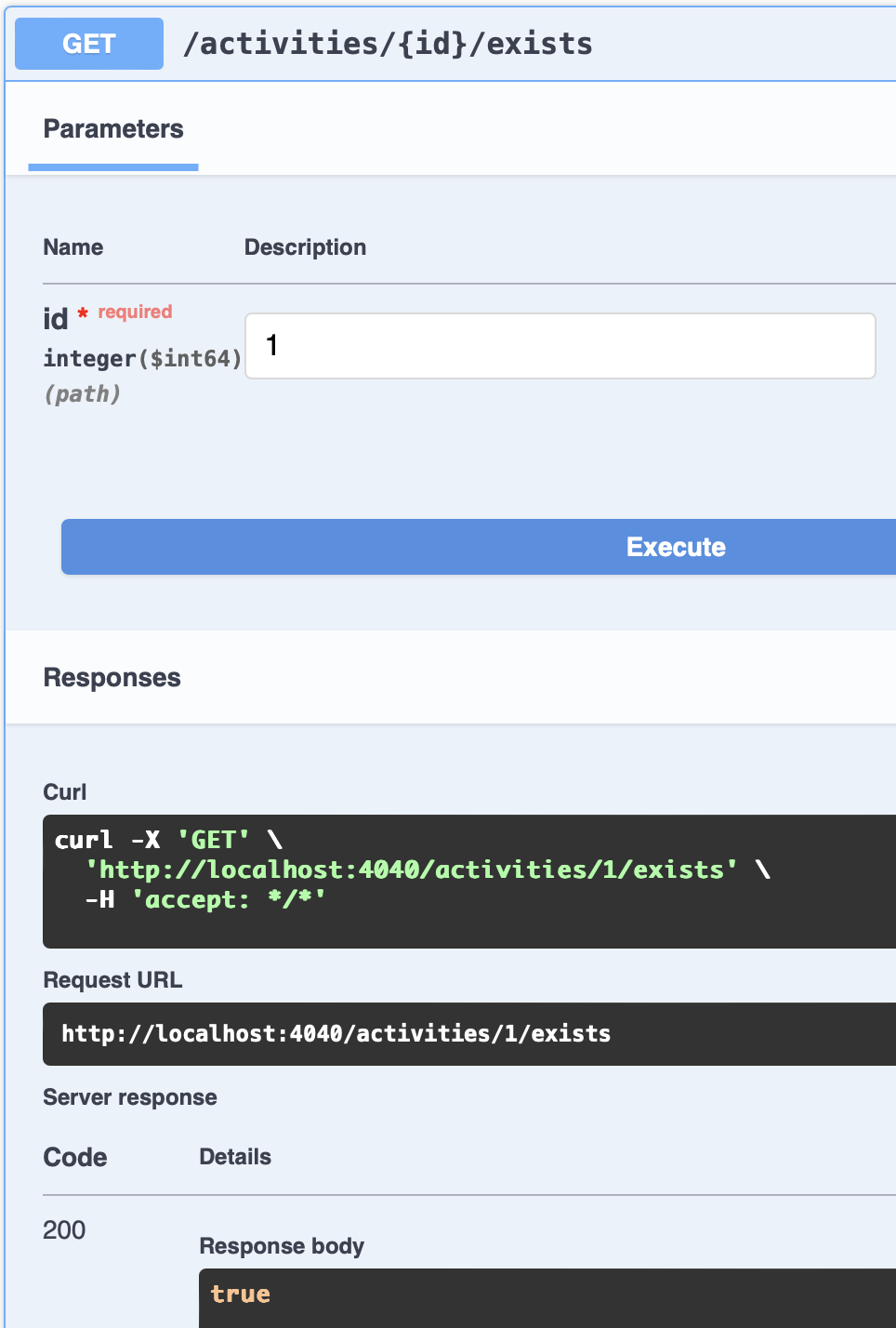
**Output**

1. **Create an activity**

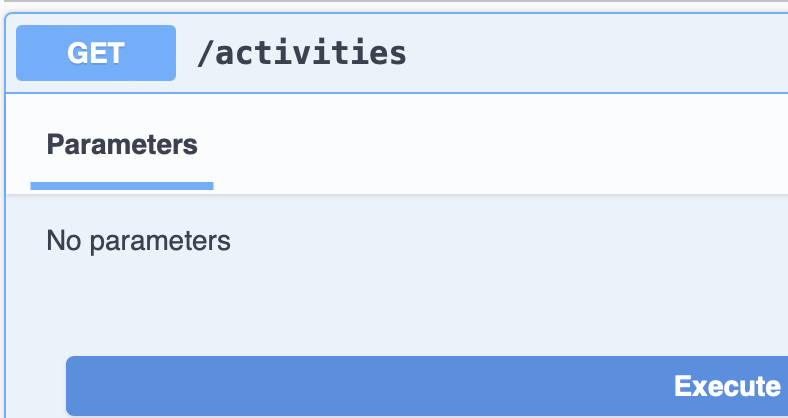
****

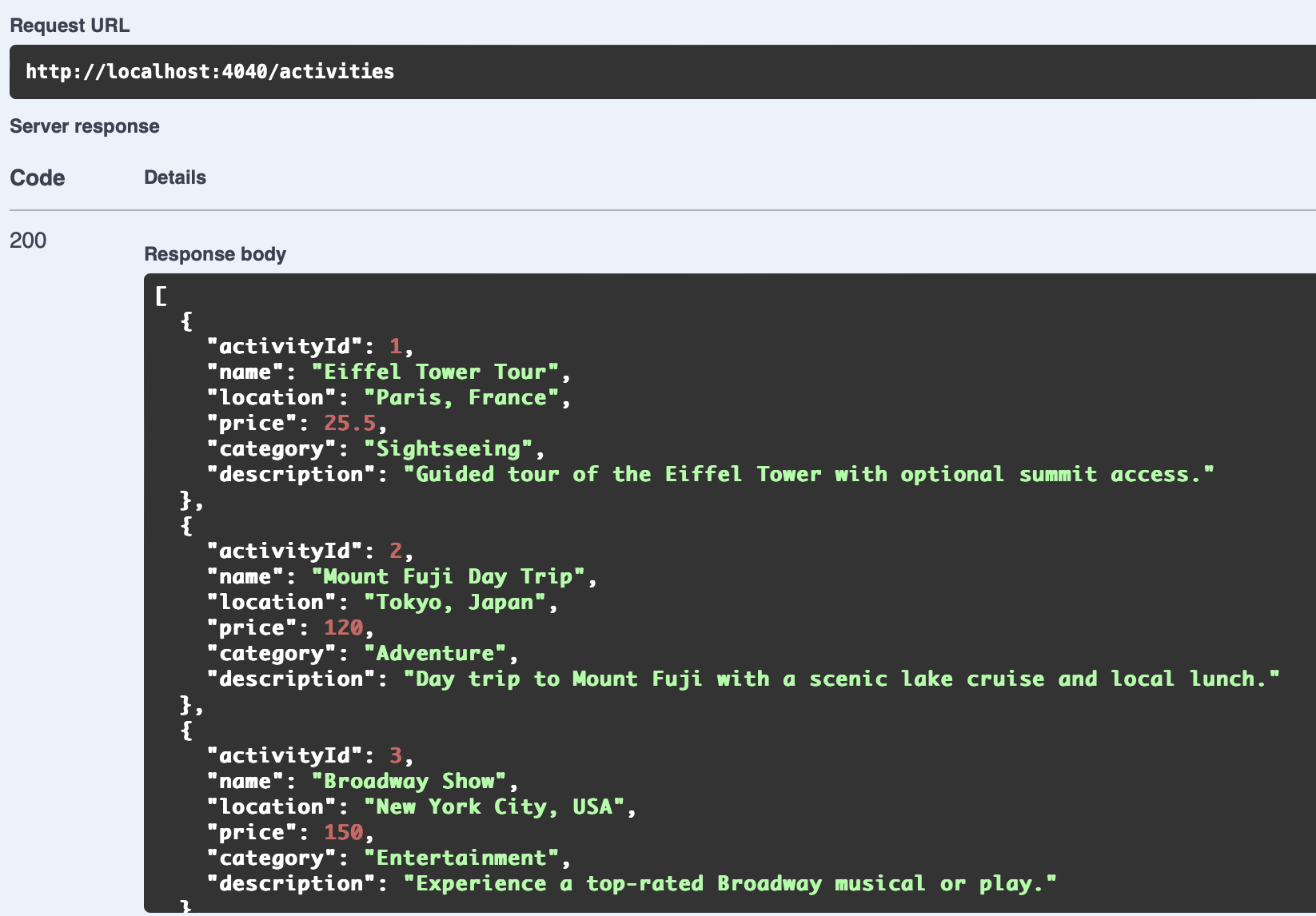
****

1. **Check if activity exists**

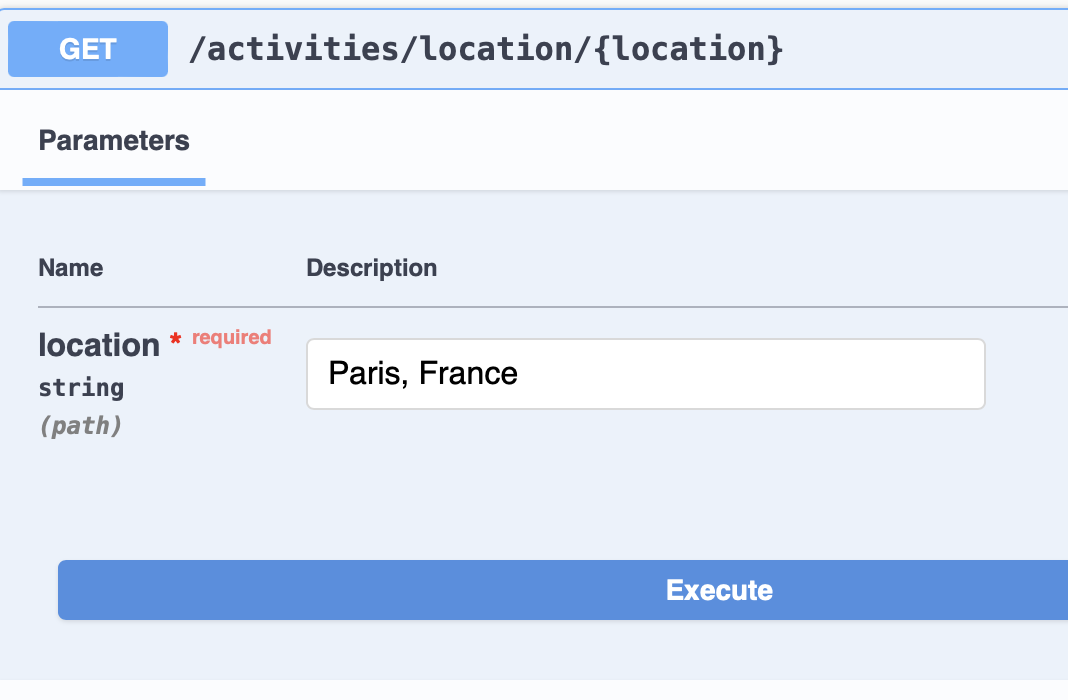
****

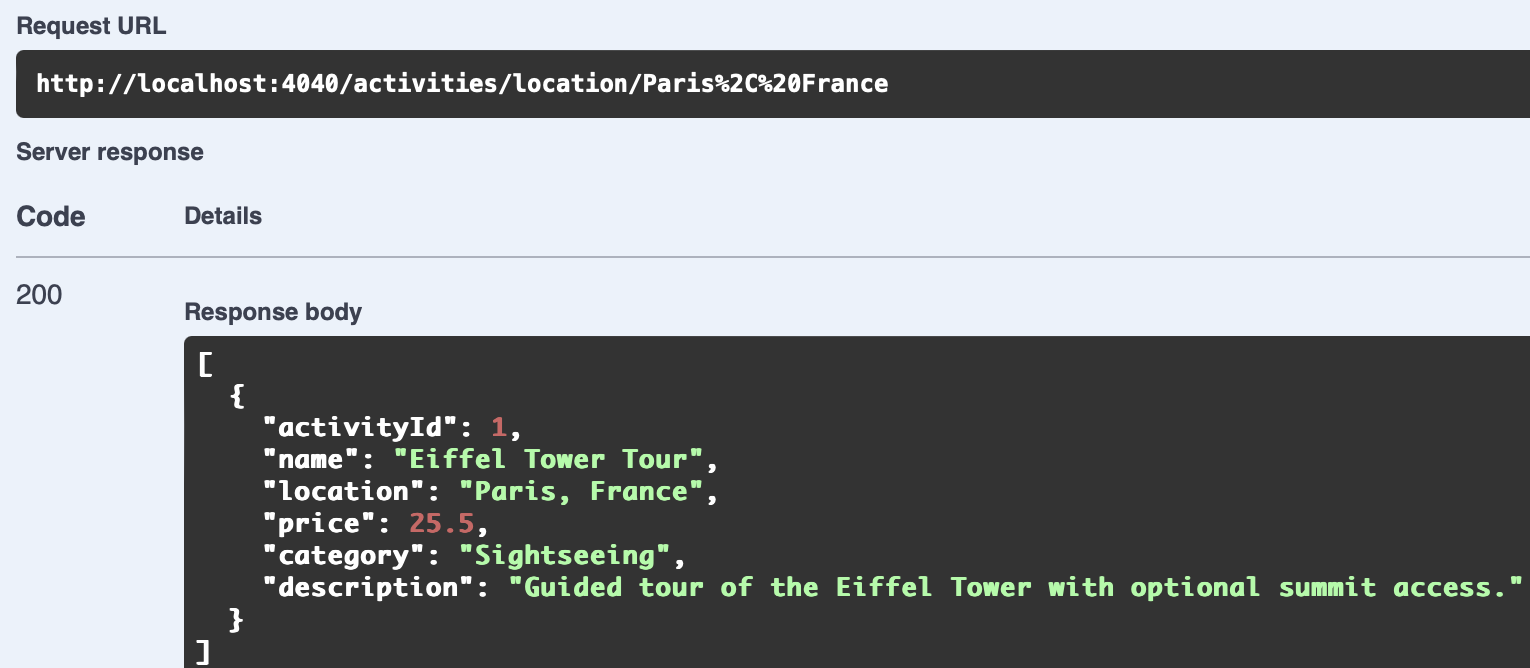
1. **Get all activities**

****

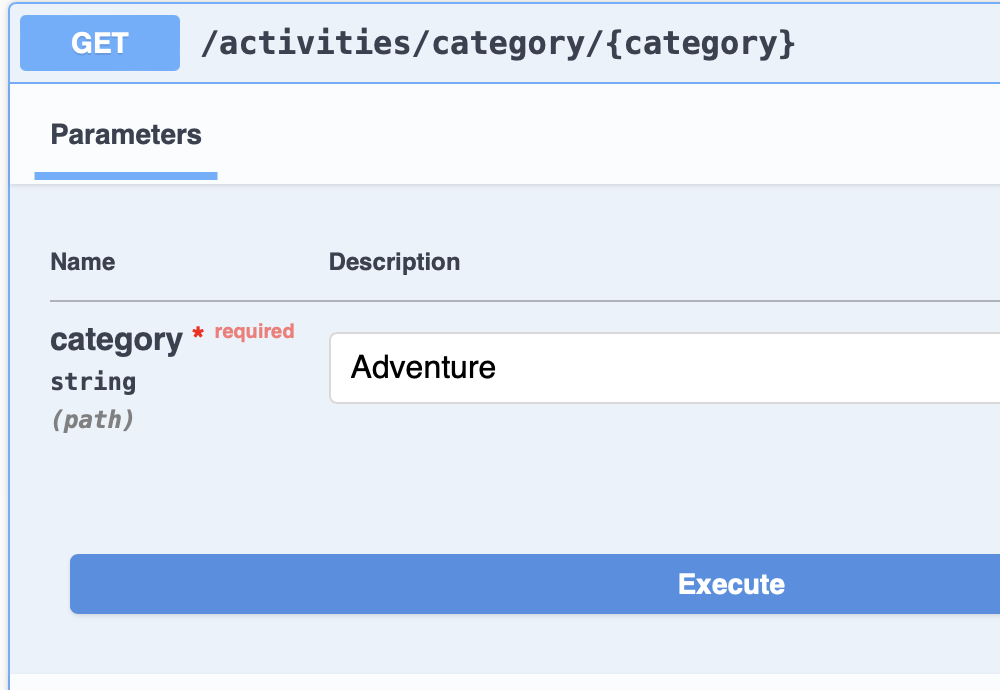
****

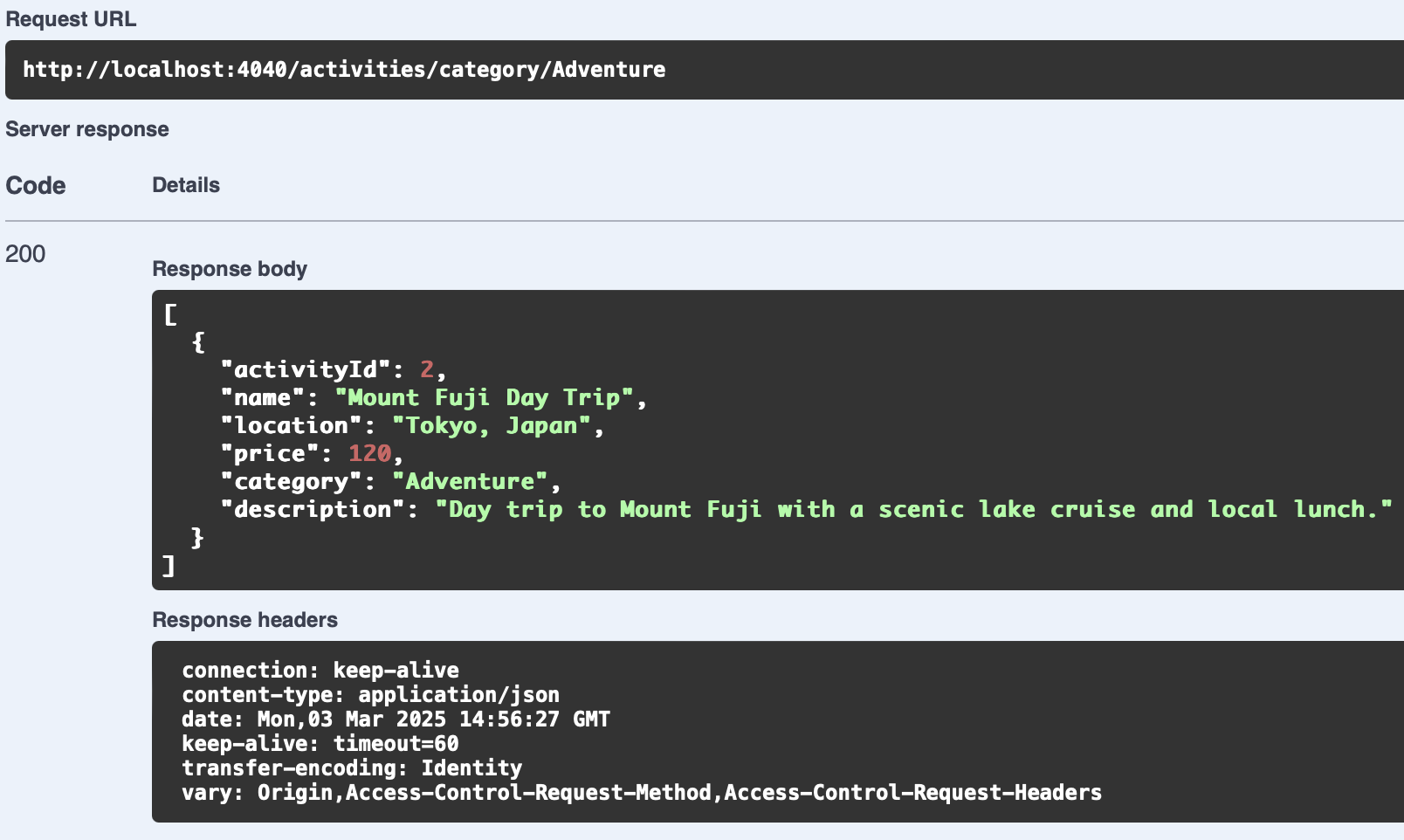
1. **Get activity by location**

****

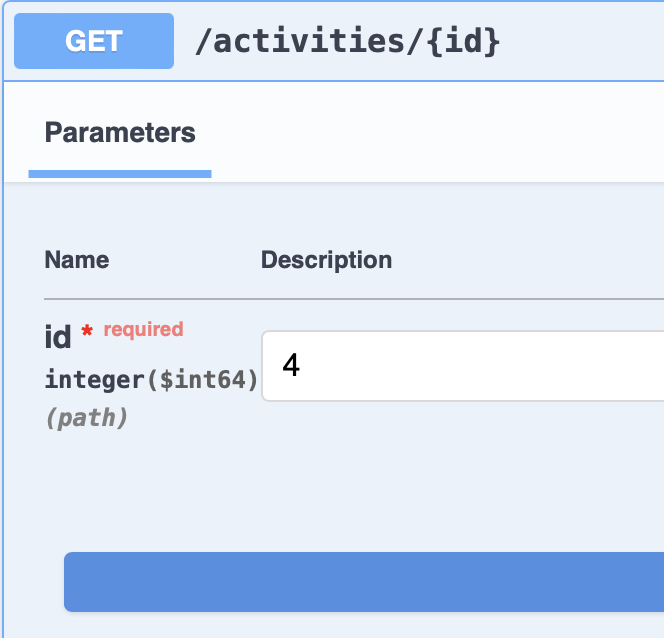
****

1. **Get activity by category**

****

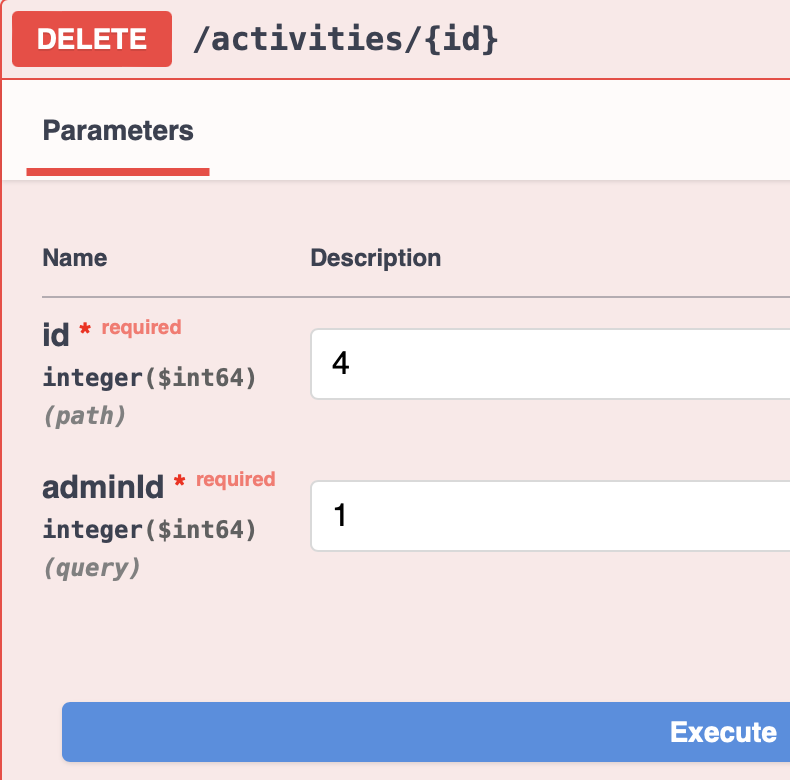
****

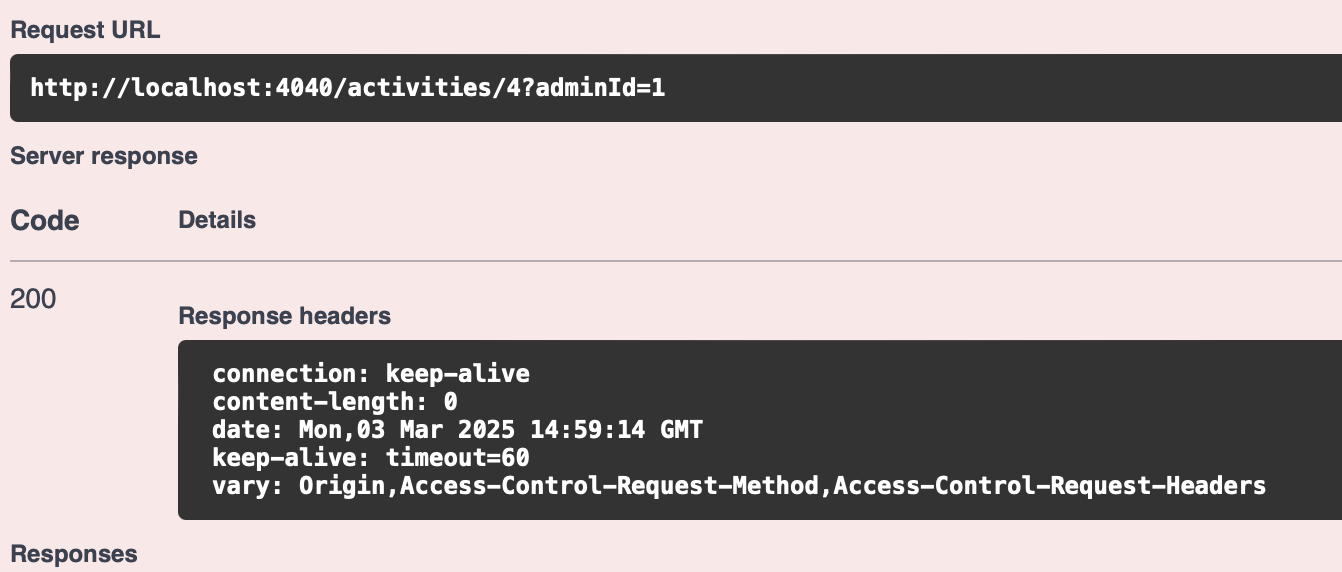
1. **Get activity by id**

****

****

1. **Delete an activity by id**

****

****

**Booking Microservice -** responsible for managing activity bookings within user itineraries. It interacts with the User, Itinerary, and Activity microservices using Feign Clients to ensure validation before confirming a booking.

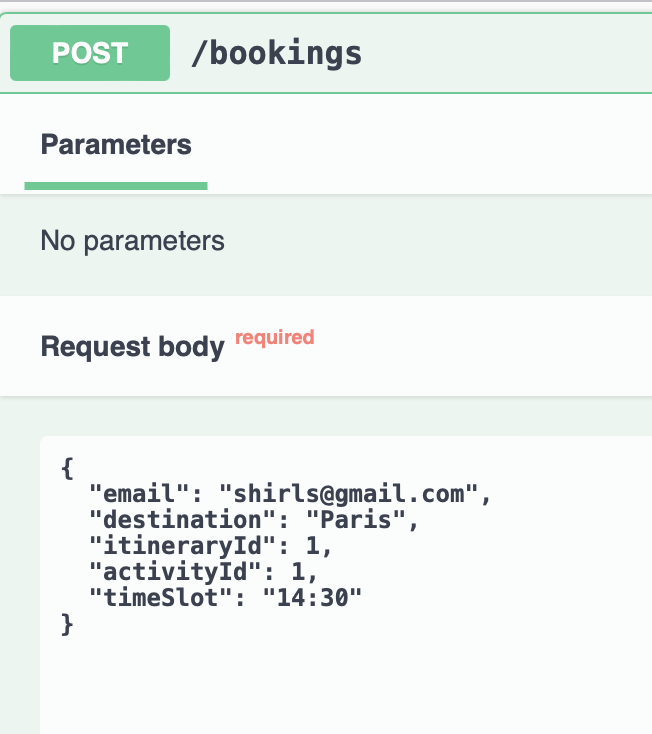
| **METHODS** | | | |
| --- | --- | --- | --- |
| **Method Name** | **Purpose** | **How It Works** | **Access Control** |
| createBooking(BookingRequest request) | Creates a new booking for an itinerary. | 1. Fetches user details using email.  2. Retrieves itineraries for the user in the specified destination.    3. Selects an itinerary (if multiple exist, user must specify).  4. Checks if the activity exists.  5. Ensures the activity is not already booked at the selected time.  6. Saves the new booking in the database. | Only registered users can book activities. |
| getUserItineraries(String email, String destination) | Fetches a user's itineraries for a given destination. | 1. Calls the User Service to retrieve the user by email.  2. Calls the Itinerary Service to fetch itineraries for the user and destination. | Only registered users can fetch itineraries. |
| cancelBooking(Long bookingId, String email) | Cancels a pending booking if the user is authorized. | 1. Fetches the booking by ID.  2. Retrieves user details using email.  3. Checks if the booking is in "PENDING" status.  4. Deletes the booking from the database. | Users can cancel their own pending bookings. |
| getUserBookings(String email) | Retrieves all bookings for a user. | 1. Calls the User Service to fetch user details.  2. Fetches all bookings associated with the user. | Only registered users can view their bookings. |

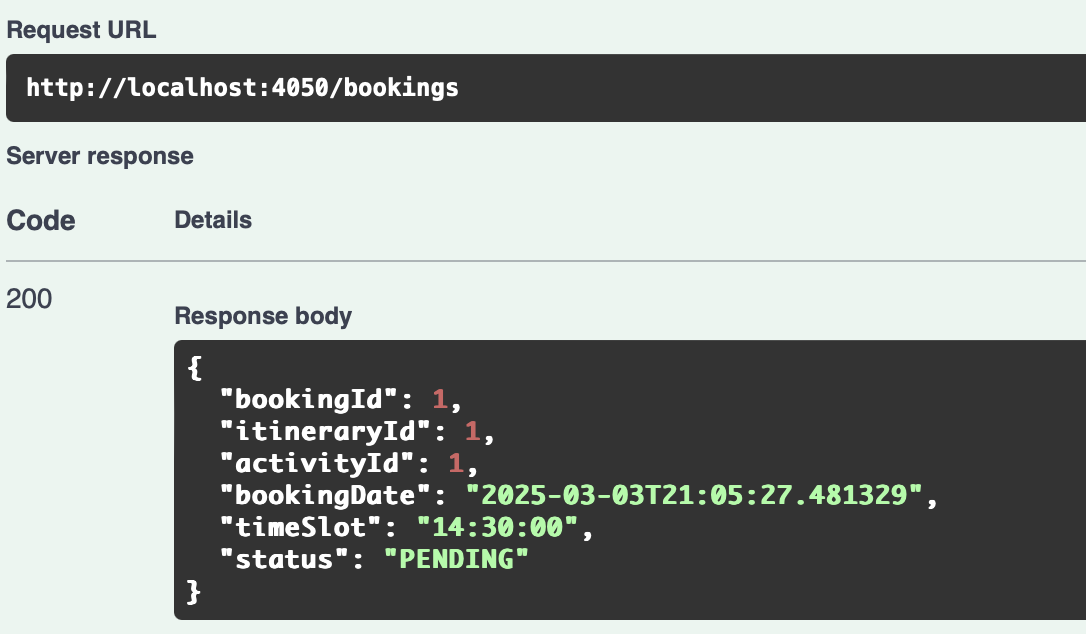
**HTTP METHODS**

| **HTTP Method** | **Endpoint** | **Request Parameters/Body** | **Functionality** | **Response** |
| --- | --- | --- | --- | --- |
| **POST** | /bookings | BookingRequest(Request Body) | Creates a new booking after validating user, itinerary, and activity. | 200 OK (Created Booking) |
| **DELETE** | /bookings/{bookingId} | bookingId (Path Variable), email (Query Param) | Cancels a pending booking if the user is authorized. | 200 OK ("Booking canceled successfully") |
| **GET** | /bookings/history | email (Query Param) | Retrieves all bookings for a user. | 200 OK (List of Bookings) |

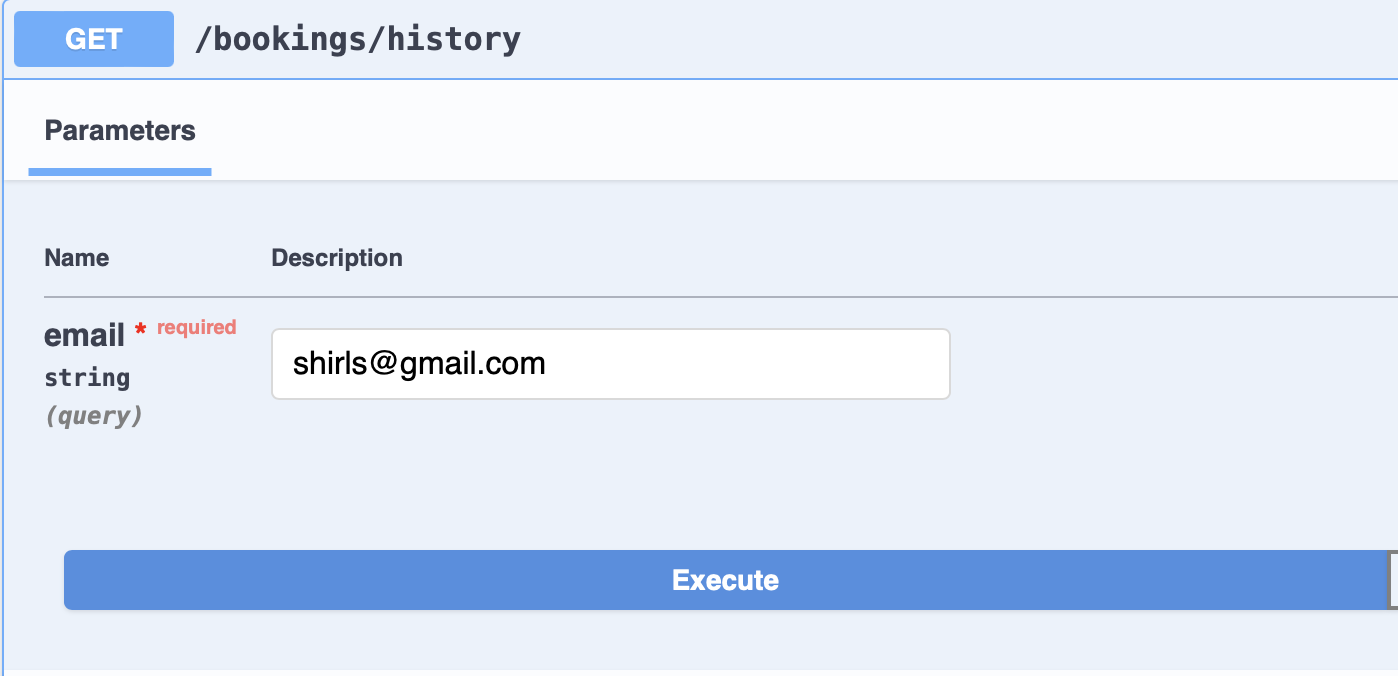
**Output**

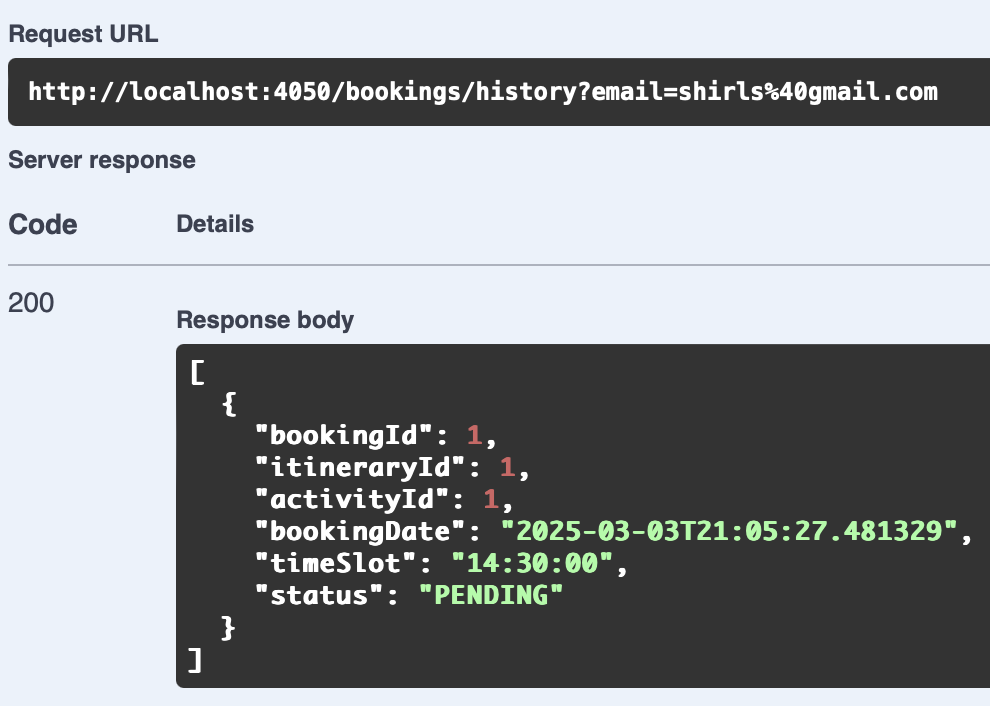
1. **Create a booking**

****

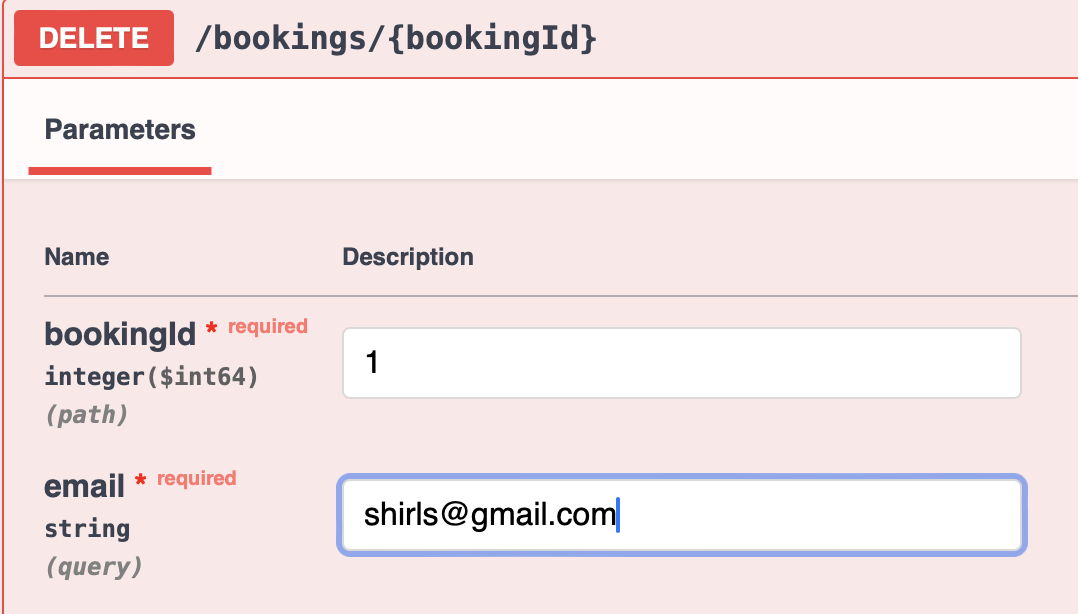
****

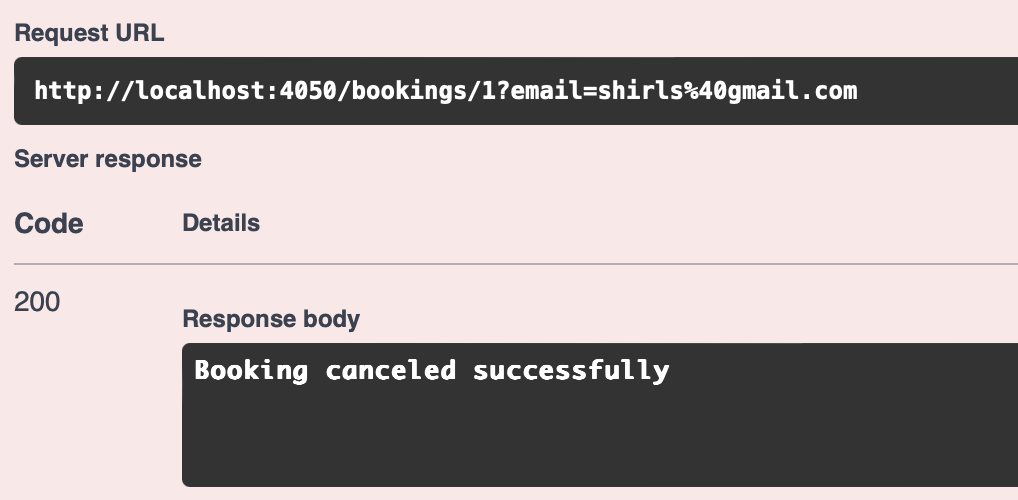
1. **Get bookings history**

****

****

1. **Delete a booking**

****

****