

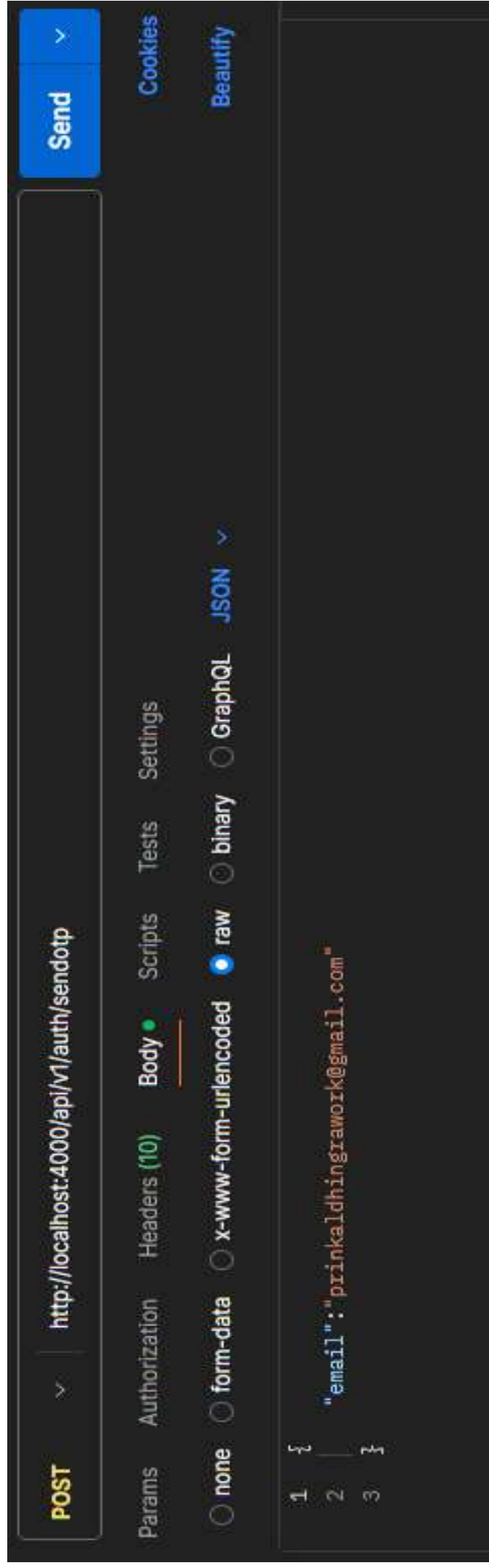


# Geekhaven

Prinkal Dhingra (IIT2023086)

# Authentication and Authorization

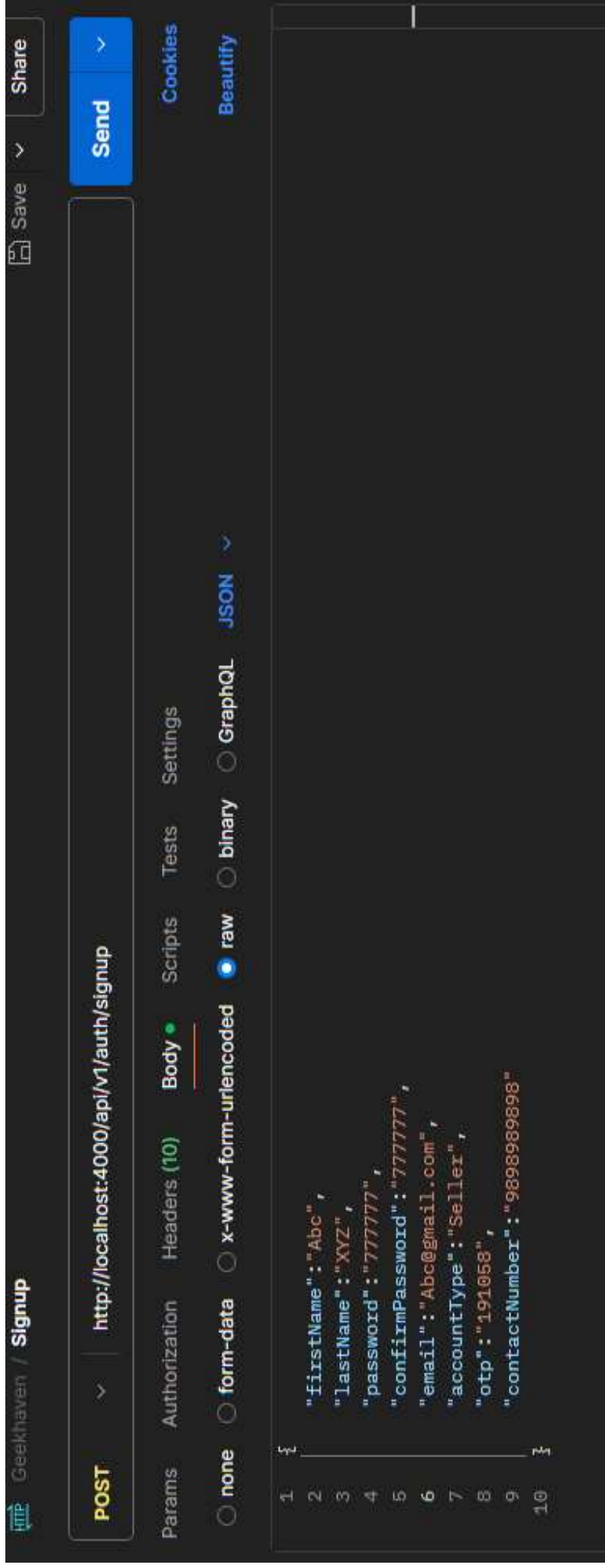
- OTP – After the user fills the registration form he requests an otp . To complete the sign up process we need to verify the otp. Refer to the photo below for the otp.



**NOTE:** If you are using college ID the mail will be in spam folder.

# • Sign-Up

For sign up refer to the image below.



NOTE : Enter the email id you entered for the otp and the otp you got in first step . You can set the Account type to Buyer,Seller,Admin.

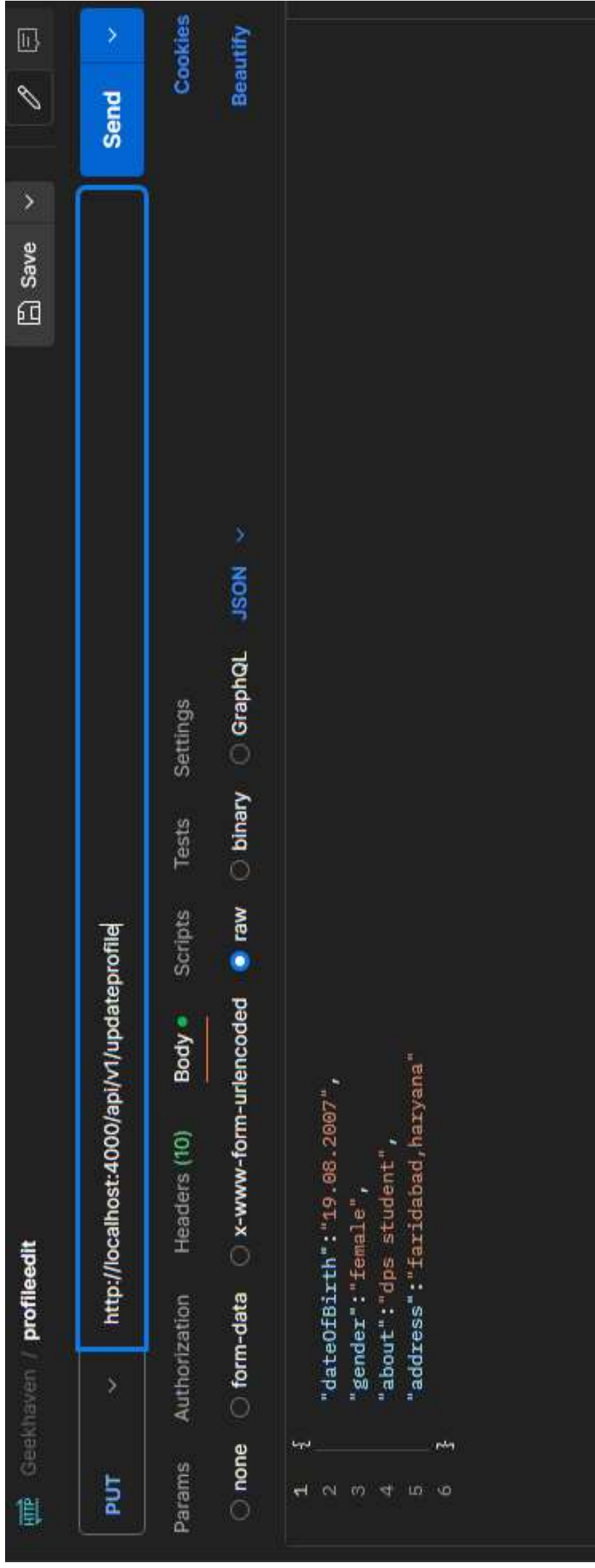
# • Login

For logging into your account using postman refer the photo :



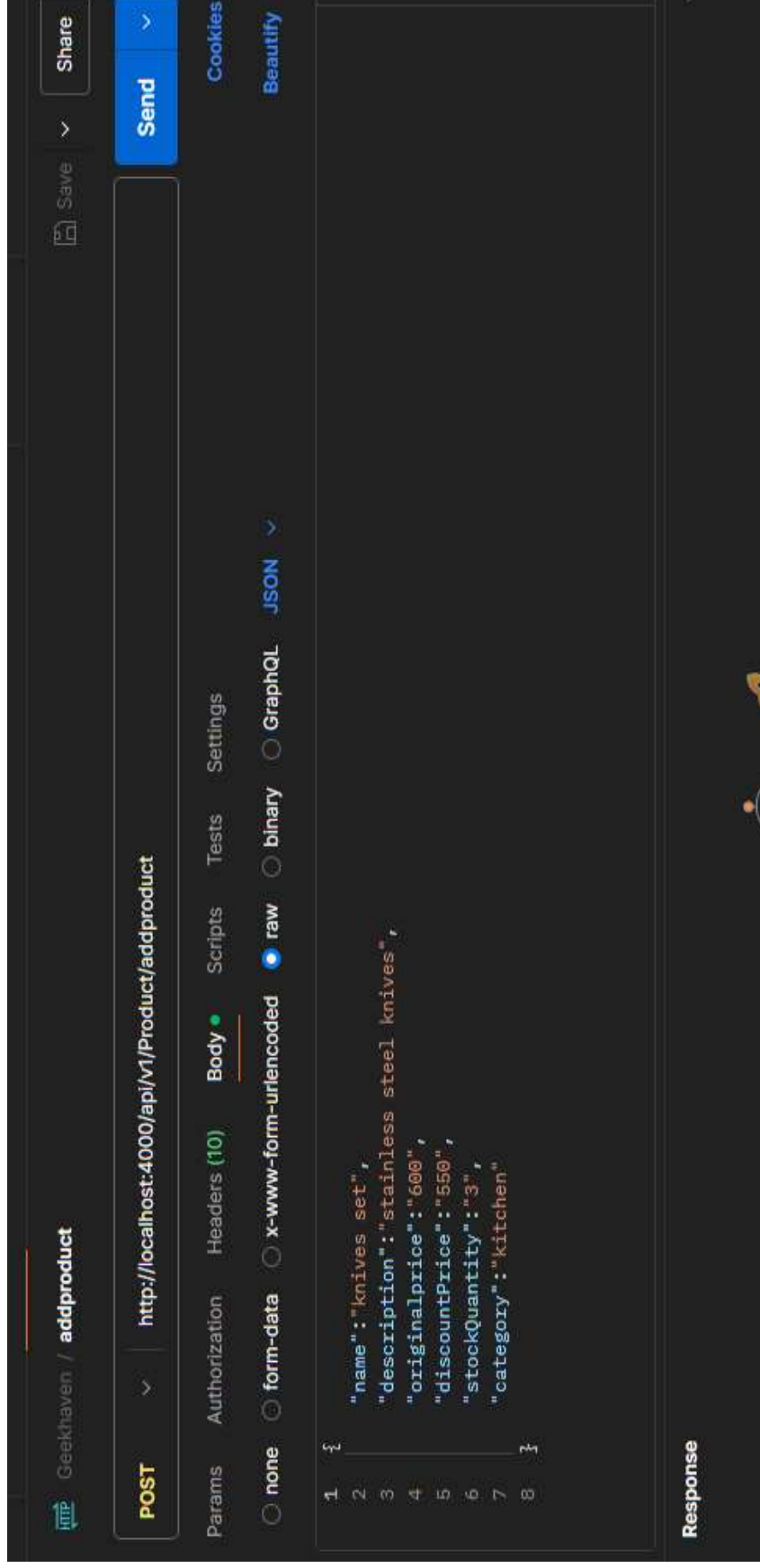
# • Update Profile

For updating your profile you need to **first login** , after that refer the photo:



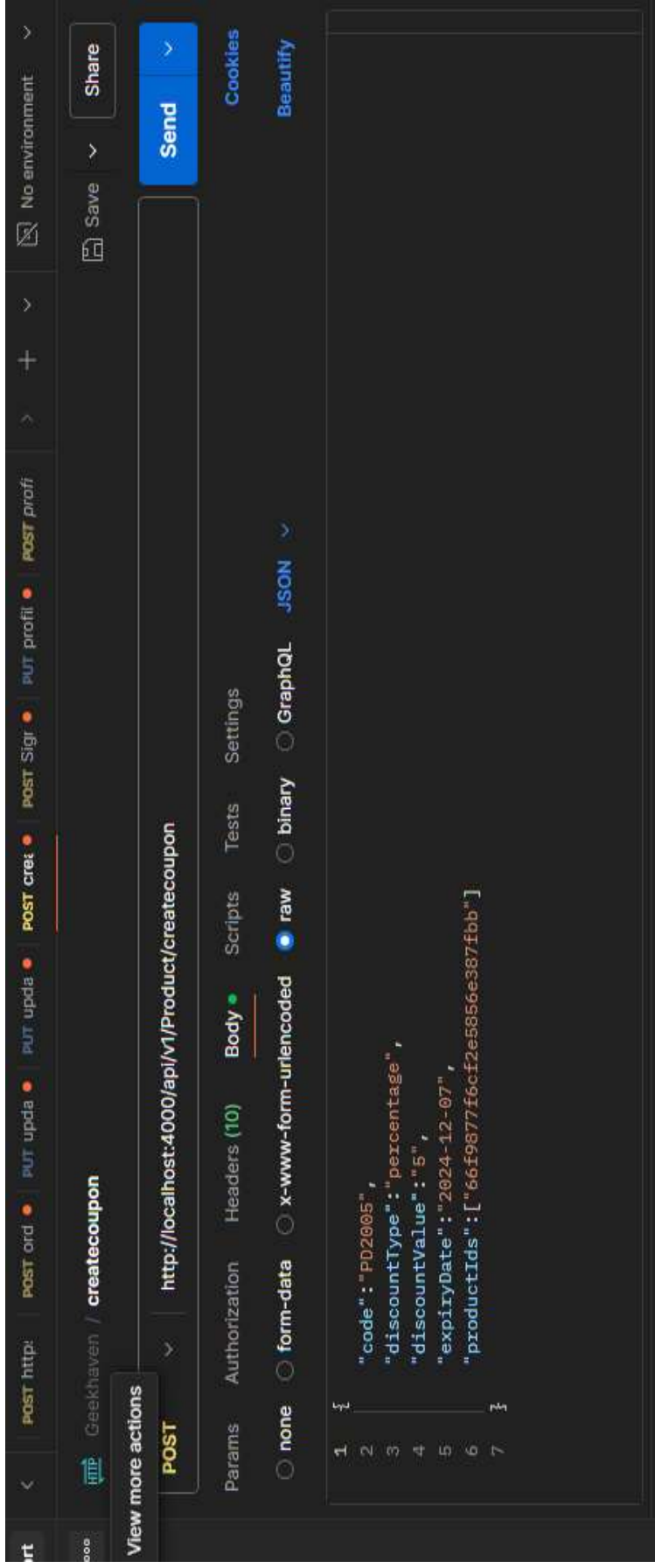
- **Add a new product**

To add a new product you need to first login as a seller as it is a protected route , then refer the photo



# • Create Coupon

To create a coupon you need to first login as a seller as only seller is allowed to create a coupon :

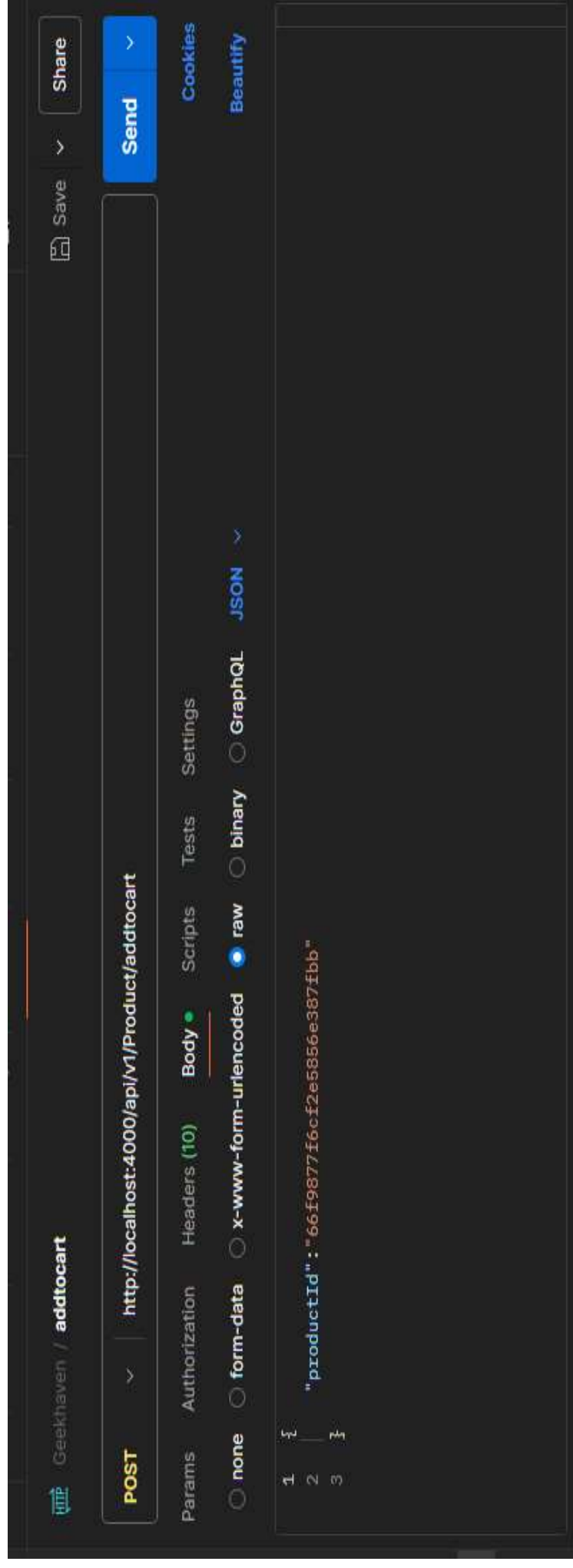


NOTE: You can add all the product ids for which you wish to apply the coupon but the products should belong to the particular seller.



- **Add to cart**

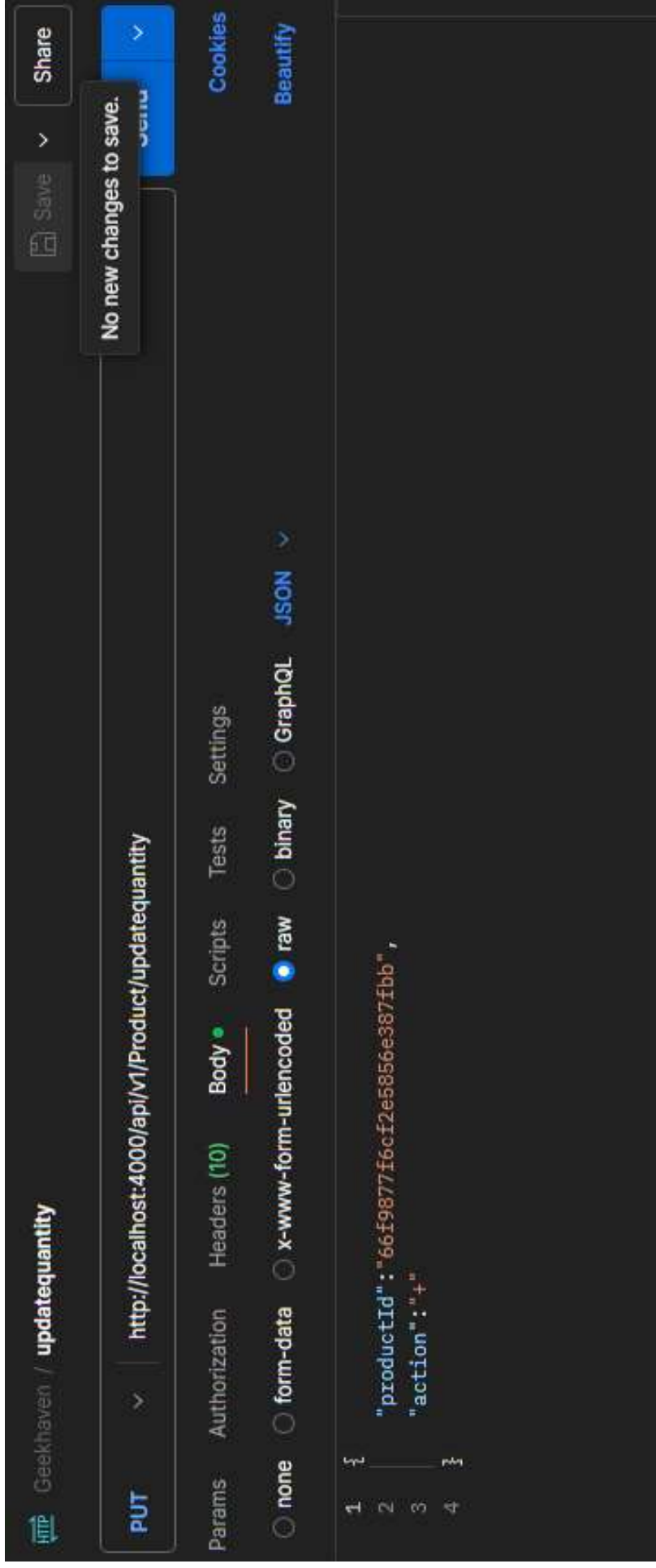
To add an item to cart you need to first login as a buyer , then refer the photo . Send a valid product id in the following format:





- **Increase quantity of a product**

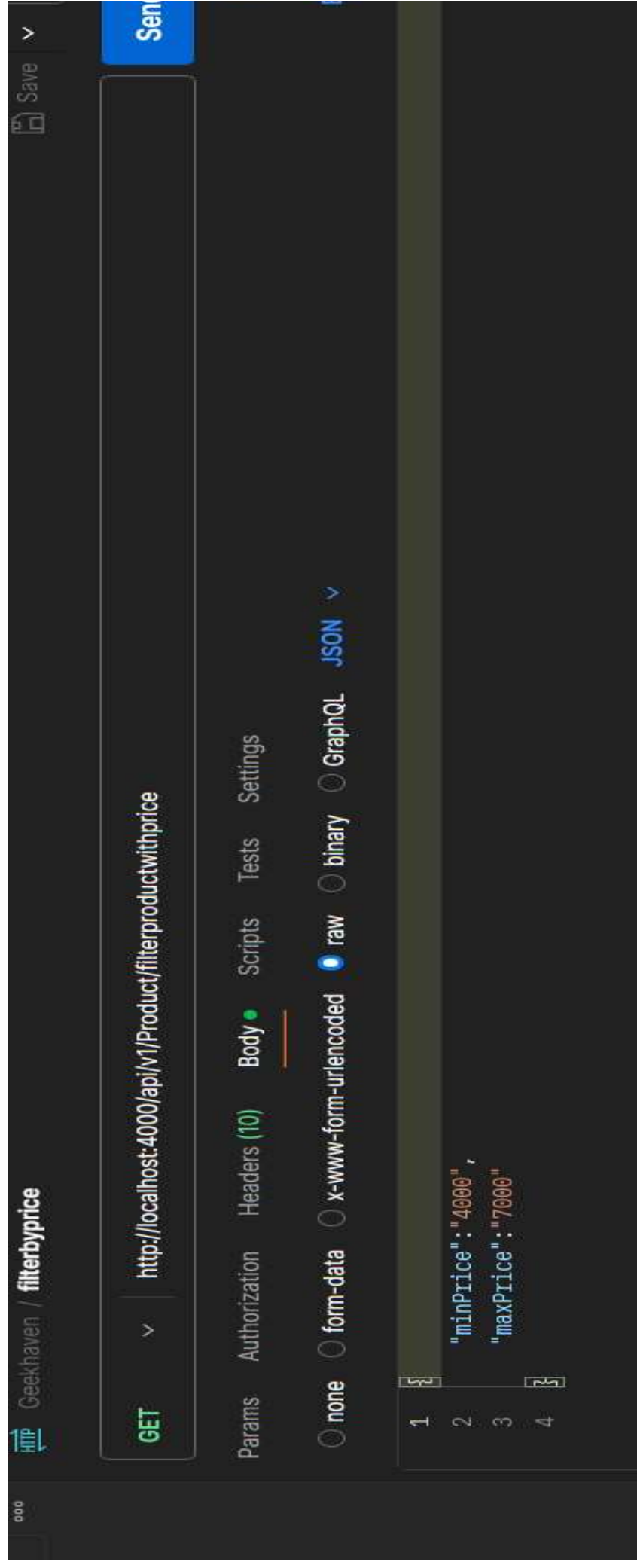
To increase the quantity of a product we are assuming that the buyer will click the "+" button in the frontend:



NOTE : To decrease the quantity change the action to "-"

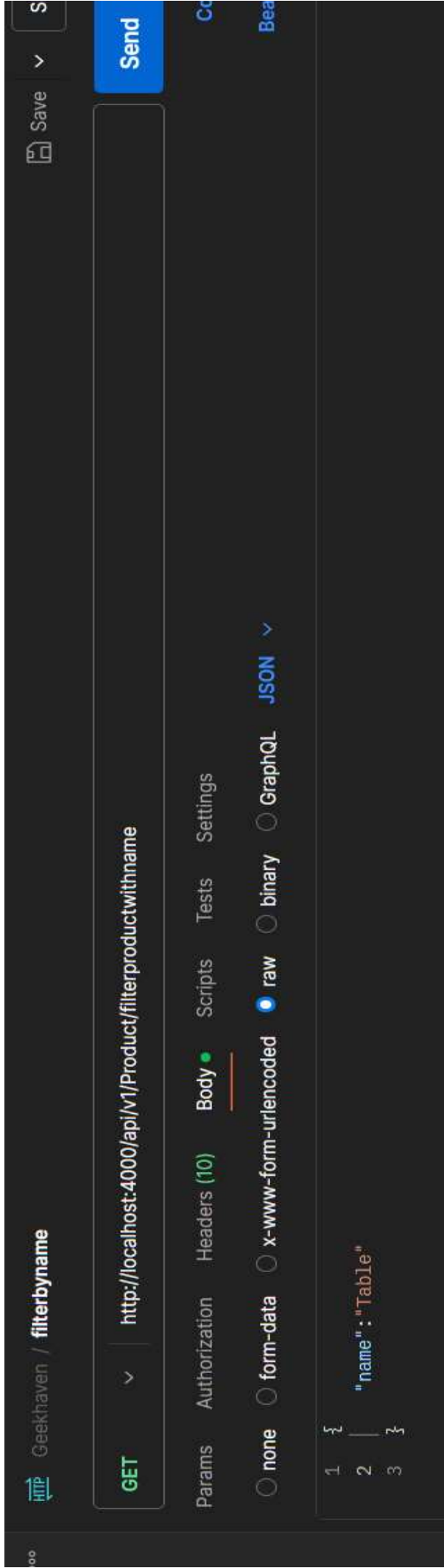
- **Filter by price**

This feature can only be accessed by the buyer :



- **Filter by name**

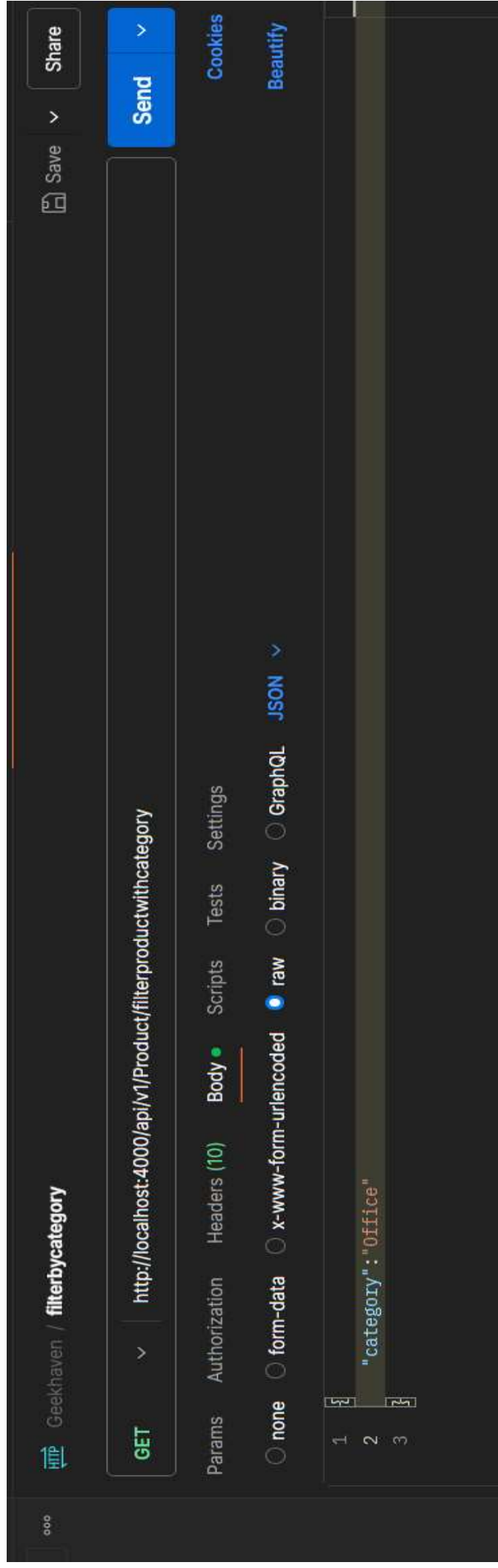
As followed above , we can filter products with name as well:



Please provide the name that you have given to the products to access this feature.

# • Filter by Category

Same as filter by name



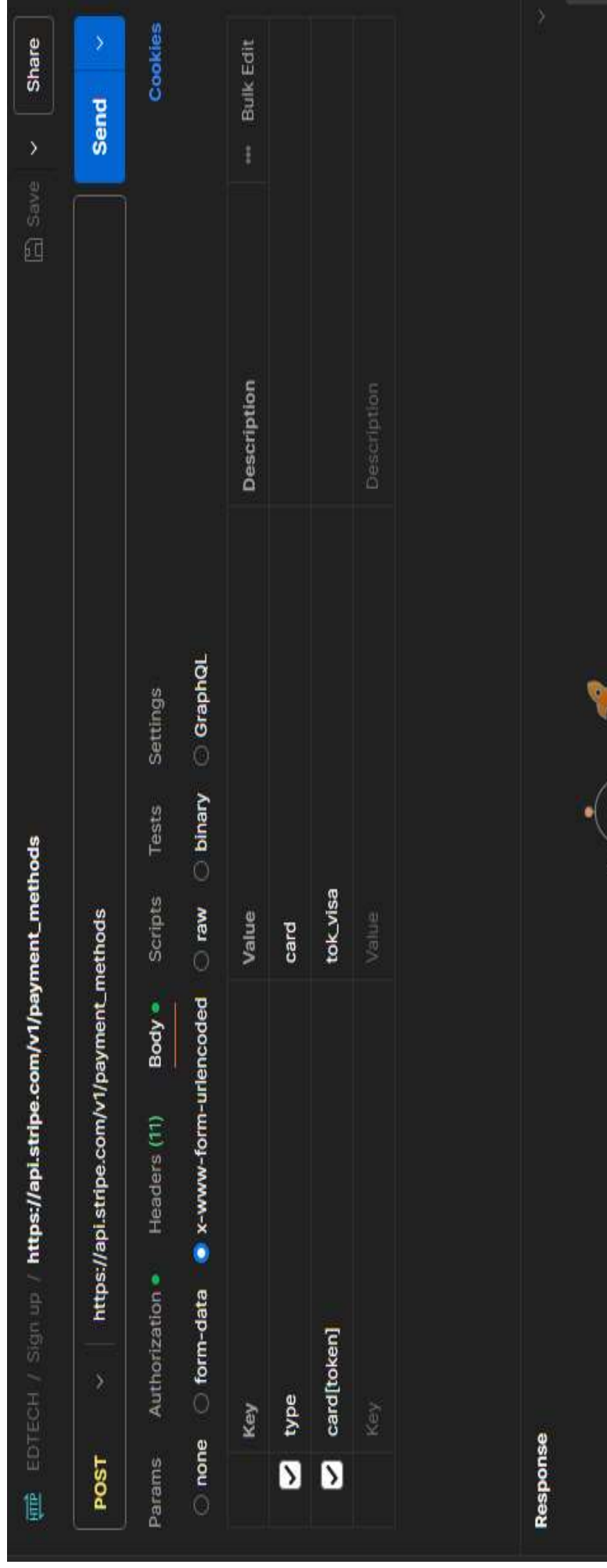
# • Add review

To add a review for a product you need to first login as a buyer as it is a protected route then refer the photo



# • Order – Card Payment

To initiate a card payment we have used stripe test mode . Open a new request and then refer the photo:



The screenshot shows a REST client interface with the following details:

- URL:** `https://api.stripe.com/v1/payment_methods`
- Method:** **POST**
- Params:** Authorization, Headers (11), Body, Scripts, Tests, Settings
- Form:** ☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL
- Body (x-www-form-urlencoded):**

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> type	card			
<input checked="" type="checkbox"/> card[token]	tok_visa			
<input type="checkbox"/> Key	Value	Description		
- Buttons:** Save, Send, Cookies, Share
- Response:** (Empty)

Fill the header as shown in the next photo.

# Filling the header

EDTECH / Sign up / [https://api.stripe.com/v1/payment\\_methods](https://api.stripe.com/v1/payment_methods)

POST

[https://api.stripe.com/v1/payment\\_methods](https://api.stripe.com/v1/payment_methods)

Save

Share

Send

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

Headers

9 hidden

Key

Value

☒

Authorization

Bearer sk\_test\_51Q44Y0KYY2tnjDeVcSj1HIXX...

Description

Bulk Edit

Presets

☒

Content-Type

x-www-form-urlencoded

Description

Key

Value

Description

In the value you can provide your own stripe test key or use the key provided below : Just paste give key in the value section of authorization.

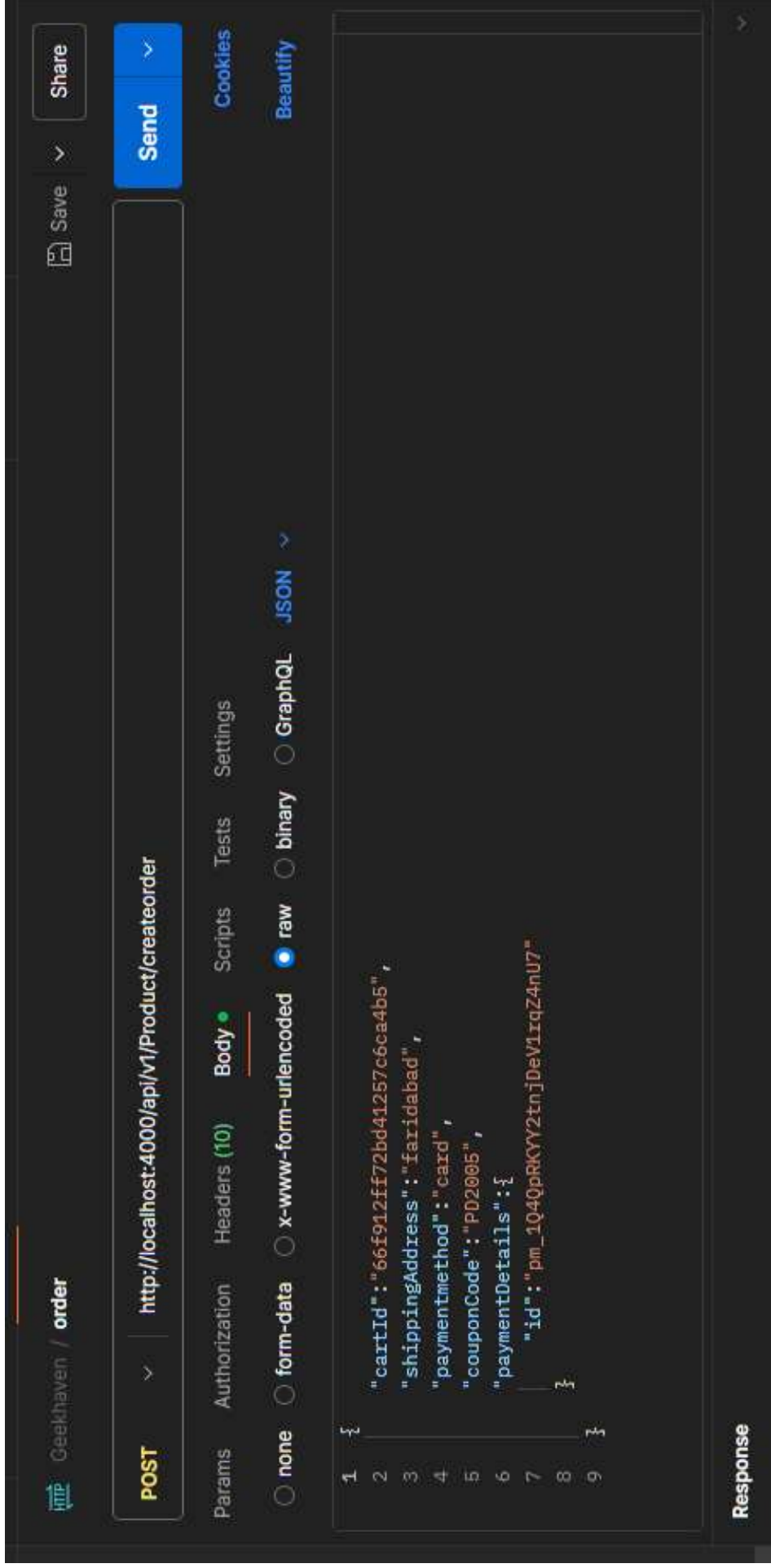
Test key – Bearer

sk\_test\_51Qho44Y0KYY2tnjDeVcSj1HIXXV37250KlghcXfyw8ciEW6qBwZCC4mmLlaiYwC0soiWc4GQ2PZ04F6XHYqyWL00kNC8sqLI



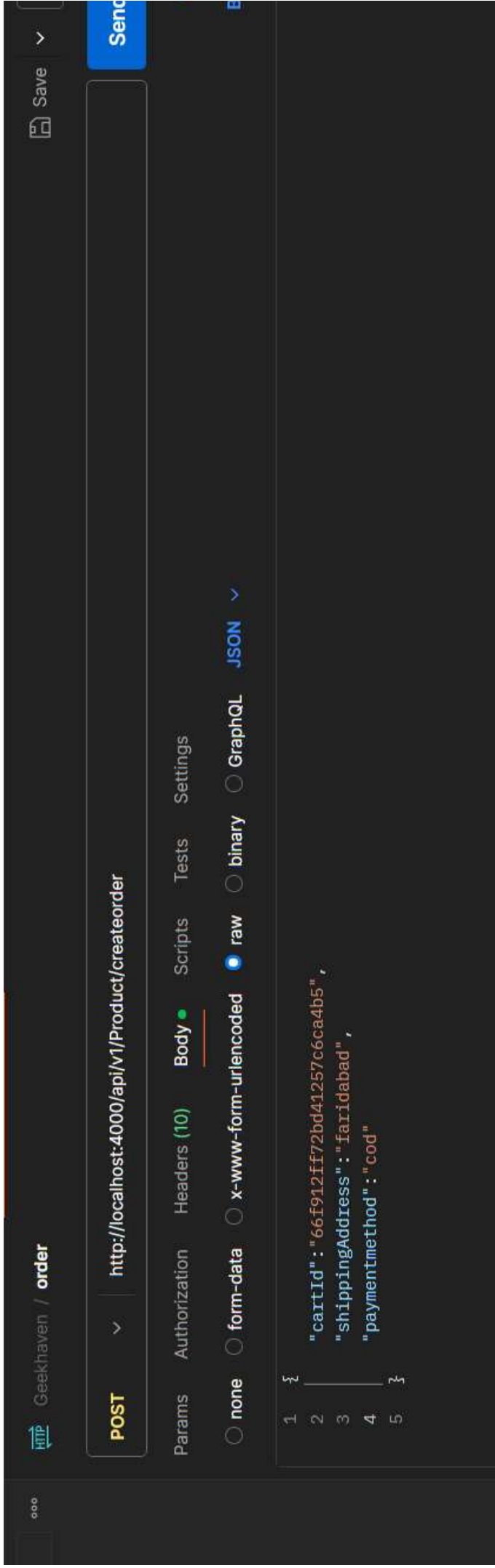
# Continue to place order

After you have successfully sent the above stripe request you will get a payment id. In id use the id you got in the previous step.



# • Order – Cash On Delivery

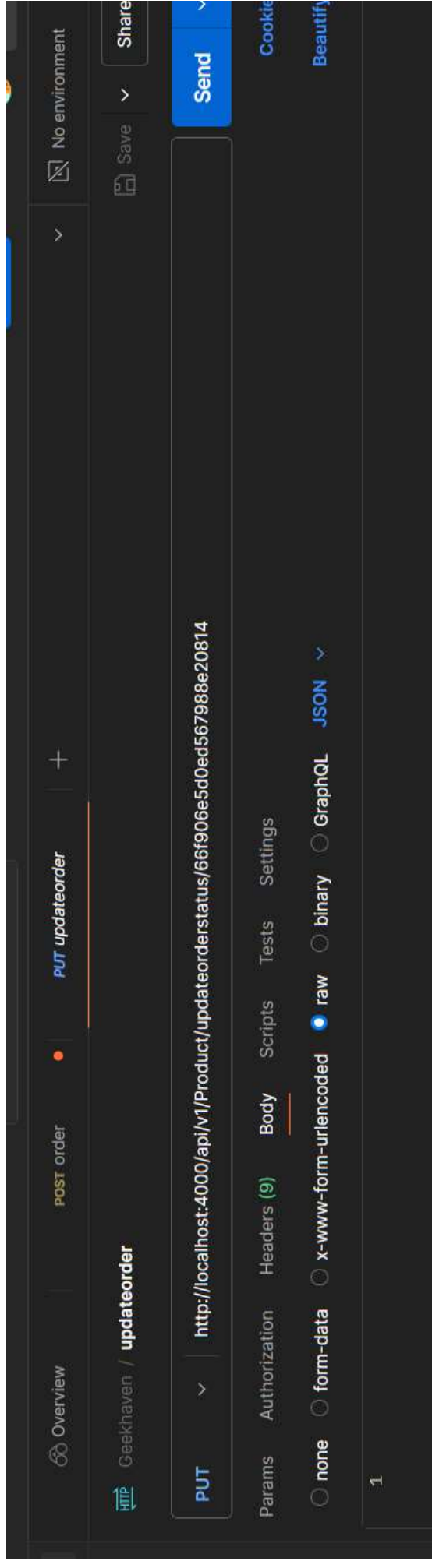
To access this method simply do the following:



NOTE: Please provide the correct cart id .

# • Update order Status

Status update of an order can only be done by the admin. First login as the admin :

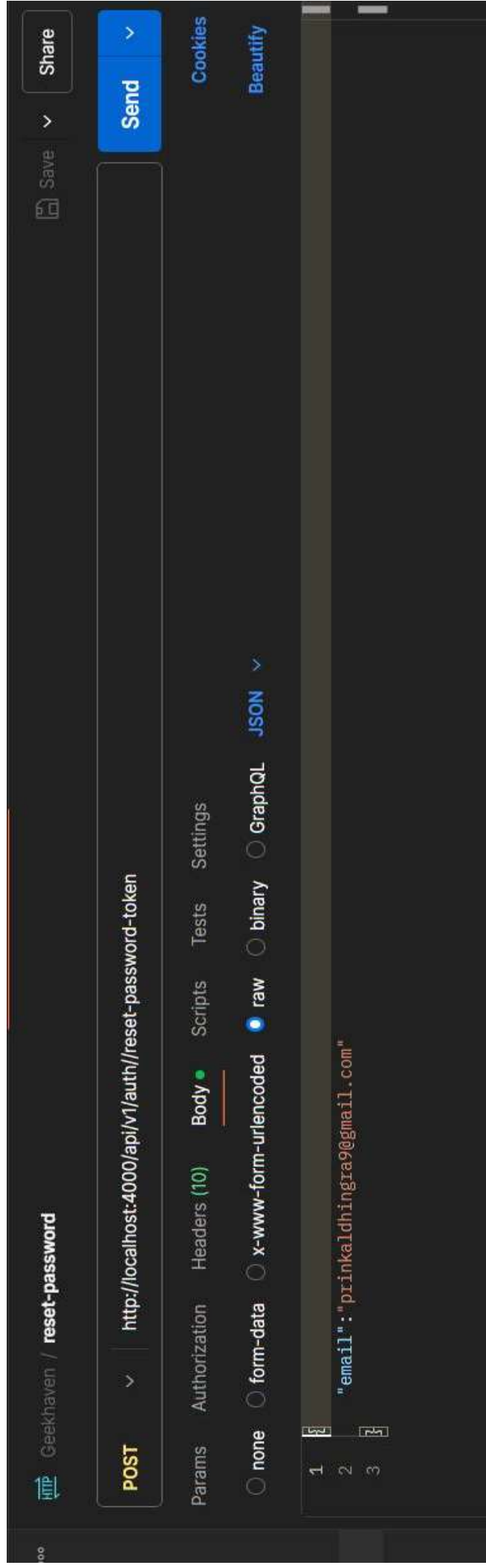


NOTE: You only need to provide the order id in the parameter as the flow is as follows :

Processing -----> Dispatched -----> Out for delivery -----> Delivered

# • Reset-Password-Link

To reset the password write the email id where you want to send the link:



# *Working of the API*

## **1. sendOtp API**

- **Purpose:** Generates and sends an OTP (One-Time Password) to the user's email for verification.
- **Flow:**
  - Extracts the email from the request body. Generates a random 6-digit numeric OTP using the otp-generator library.
  - Ensures the generated OTP is unique by checking it against the database.
  - Saves the OTP to the database associated with the email.
  - Responds with a success message along with the generated OTP.

## 2. signUp

- **Purpose:** Handles the user registration process.
- **Flow:**
  - Receives user details such as `firstName`, `lastName`, `email`, `password`, `confirmPassword`, `accountType`, `otp`, and `contactNumber`.
  - Validates that all required fields are provided and ensures that the password and `confirmPassword` match.
  - Checks if a user with the given email already exists in the database.
  - Verifies the provided OTP against the most recently created OTP for that email.
  - Hashes the password using `bcrypt`.
  - Creates a user profile with default values for additional details.
  - Sends a welcome email using the `mailSender` utility.
  - Returns a success message with the created user.

# 3. Login API

- **Purpose:** Handles user login by verifying credentials and issuing a JWT token.
- **Flow:**
  - Receives the email and password from the request body.
  - Checks if both fields are provided.
  - Finds the user by email and populates the user's profile details.
  - If the user is found, compares the password using bcrypt.
  - If the password is correct, creates a JWT token with a 2-hour expiration and sets the token as a cookie.



## 4.Change Password API

- Retrieves user data (`userDetails`) from the database
- Gets `oldPassword` and `newPassword` from `req.body`
- Compares `oldPassword` with stored password using `bcrypt`
- Hashes `newPassword` using `bcrypt.hash()`.
- Updates user password in the database via `User.findByIdAndUpdate()`.
- Attempts to send a confirmation email using `mailSender()`.

## 5.Add Items to Cart

- **Request:** The API expects productId in the request body. It identifies the user from req.user.id.
- **Functionality:**
  - It checks if the productId is valid.
  - If the user's cart already exists:
    - It checks if the product is already in the cart. If yes, it increments the quantity by 1.
    - If the product is not in the cart, it adds the product with a quantity of 1.
  - If the cart does not exist, it creates a new cart with the product and quantity.

# 6.Update Cart Quantity

- **Request:** The API expects productId and an action (+ or -) in the request body, which specifies whether to increment or decrement the product quantity.
- **Functionality:**
  - It validates the productId and finds the user's cart.
  - If the product exists in the cart:
    - For action "+", it increases the quantity by 1.
    - For action "-", it decreases the quantity by 1 (if the quantity is more than 1). If the quantity is less than or equal to 1, it returns an error message.
  - If the product is not found in the cart, it returns an error.
- **Response:** Returns a success message with the updated cart.

# 7.Delete Item from Cart

- **Request:** The API expects productId in the request body.
- **Functionality:**
  - It validates the productId and checks if the cart exists for the user.
  - If the product exists in the cart, it removes the product.
  - If there are no items left in the cart, the cart itself is deleted from the database

# 8.createCoupon

- This API allows a seller to create a coupon that can be applied to specific products.
- The seller sends a request with details like coupon code, discount type, discount value, expiry date, and the list of product IDs (productIds) the coupon should be applied to.
- The sellerId is obtained from the logged-in seller (typically attached to the request object).
- The system checks if all the productIds provided in the request belong to the seller by querying the Product model.
  - If any product doesn't belong to the seller, it sends a 400 response indicating the issue.
- If the validation passes, a new coupon is created with the provided details, including the applicableProducts (product IDs) and the sellerId.

# 9.validateCoupon

- The buyer sends a request containing a coupon code and a specific productId to validate whether the coupon applies to that product.
- The system looks up the coupon by the provided code in the Coupon collection.
- If the coupon doesn't exist, it returns an error.
- It then checks if the coupon has expired by comparing the current date with the coupon's expiryDate.
- If the coupon is expired, it returns with an error message.
- It further checks if the coupon is applicable to the provided productId by verifying if the product ID is included in the coupon's applicableProducts array.

# 10. Create Order API

- **Request Data:** cartId, shippingAddress, paymentmethod, paymentDetails, and optionally couponCode.
- **Validate Payment:** If the payment method is 'card', it ensures that the paymentDetails.id is provided.
- **Fetch Cart:** Retrieves the user's cart from the database based on cartId and populates product details.
- **Calculate Total Price:** Loops through the cart items and calculates the total price using the discountPrice of each product and its quantity.
- **Coupon Validation:** If a couponCode is provided, it checks the coupon's validity and applicability to the cart items, and applies the discount.
- **Payment Processing:**
  - For 'card', it processes the payment using Stripe's API with the total price.
  - For 'cash on delivery (COD)', it stores a pending payment status.
- **Create Order:** Saves the order to the database with the order details (cart items, user, payment info, etc.).



# 11. Get All Orders by User ID API

- **Request Data:** The `userId` is passed as a URL parameter.
- **Steps:**
- **Fetch Orders:** Retrieves all orders from the database associated with the specified `userId`.
- **Check if Orders Exist:** If no orders are found, it returns an error response.

# 12.Update Order Status API

- **Request Data:** The orderId is passed as a URL parameter.
- **Find Order:** Fetches the order by ID from the database.
- **Automatic Status Transition:**
  - If the order is in the Processing stage, the status is updated to Dispatched, and product stock is adjusted based on the ordered quantities.
  - If in the Dispatched stage, it is updated to Out for delivery.
  - If in the Out for delivery stage, it is updated to Delivered, and the payment status is set to Succeeded.
  - If the status is already Delivered, it throws an error.
- **Update Stock:** When transitioning to Dispatched, the stock of each product is reduced on the ordered quantity.
- **Send Status Update Email:** Sends an email to the user notifying them of the status change.

# 13.Add a New Product

- **Purpose:** Adds a new product to the database and associates it with a seller.
- **Flow:**Extracts the product details (name, description, prices, stock, etc.) and validates that all required fields are provided.
- Creates a new Product in the database with the provided details.
- Associates the newly created product with the seller by updating the seller's products array with the new product's ID.

# 14.Show All Products

- **Purpose:** Fetches all products from the database.
- **Flow:** Queries the Product collection to retrieve all products.
- Returns the product details like name, description, original price, and stock quantity.
- If successful, it returns a list of all products; otherwise, it handles any errors.

# 15.Delete a Product

- **Purpose:** Deletes a specific product from the database.
- **Flow:**Extracts the product ID from the request body.
- Finds the product by ID and checks if it exists.
- If found, deletes the product from the Product collection and removes the product ID from the seller's products array in the User collection.
- Confirms product deletion, or returns an error if something goes wrong.

## 16.Add a Review for a Product

- **Purpose:** Adds or updates a review for a specific product.
- **Flow:**Extracts the user ID, rating, comment, and product ID from the request.
- Finds the product by ID and checks if the user has purchased the product or not.
- If a review already exists from the user, it updates the existing review; otherwise, it adds a new review.
- Recalculates the product's average rating and saves the changes to the product.

# 17.Filter Products

## A.Filter by Price Range (getProductsByPriceRange):

- **Purpose:** This API filters products based on a specified price range.
- **Flow:**The API extracts minPrice and maxPrice from the request body and validates that they are numbers.
  - It constructs a filter object that specifies a discountPrice range using MongoDB's query operators (\$gte for greater than or equal, and \$lte for less than or equal).
  - It then queries the Product collection to find products that fall within the defined price range.
  - If products are found, it returns them; if none are found, it returns a "not found" message.



## B. filter by name or category

- **Purpose:** This API filters products based on their names.
- **Flow:**
  - It extracts the product name or category from the request body and validates its presence.
  - The API uses a case-insensitive regular expression to search for products whose name or category contain the specified string.
  - It returns a list of matching products or a "not found" message if no matches are found.

# 18.Edit a Product

- **Purpose:** Updates the details of an existing product and notifies users if there's a price change.
- **Flow:**Extracts the product ID from the request parameters and the updated product details from the request body.
- Finds the product by ID and updates its details (name, description, prices, stock, etc.).
- If the discountPrice has decreased, the API finds all users who have added this product to their wishlist.
- Sends a notification email to each of these users, informing them of the price drop.
- Returns the updated product or an error if the product is not found or the update fails.

# 19.updateProfile

- **Purpose:** Updates the user's profile information.
- **Process:**Retrieves the required fields (dateOfBirth, gender, about, address) from the body and the user ID from the authenticated user.
- Checks if address and gender are provided , if not the it returns an error.
- Finds the user by ID and retrieves their associated profile ID from additionalDetails
- Updates the profile using Profile.findByIdAndUpdate() with the new data and returns updated user details, populating the profile information.

## 20.deleteAccount

- **Purpose:** Deletes a user account and their profile.
- **Process:**Retrieves the user ID from the authenticated user.
- Finds the user by ID to get their profile ID from additionalDetails.
- Deletes the profile using `Profile.findByIdAndDelete()` and then deletes the user using `User.findByIdAndDelete()`.

# 21.resetPasswordToken

- **Purpose:** This endpoint is responsible for initiating the password reset process by generating and sending a password reset link to the user's email.
- **Working:**
- **Input:** The API expects the user's email in the request body.
- **Validation:** It checks if the email is provided; if not, it returns with an error message.
- **User Lookup:** It searches for the user in the database using the provided email.
  - If the user does not exist, it returns an appropriate error message.
- **Token Generation:** If the user exists, it generates a unique token.
- **Database Update:** It updates the user's document in the database with the generated token and an expiration time for the token (5 minutes from the current time).
- **Email Sending:** A password reset link is constructed using the token and sent to the user's email via the mailSender utility.

# 22.resetPassword

- **Purpose:** This endpoint allows the user to reset their password using the token received in the email.
- **Working:**
- **Input:** The API expects the token, new password, and password confirmation in the request body.
- **Validation:** It checks if all required fields (token, password, confirmPassword) are provided; if any are missing, it returns an error message.
- **Token Verification:** It searches for a user in the database with the provided token.
  - If no user is found, it returns a response indicating that the token is invalid.
- **Token Expiry Check:** It verifies whether the token has expired. If the token is expired, it returns a 500 status with an appropriate message.
- **Password Confirmation:** It checks if the new password matches the confirmation password. If they do not match, it returns an error message.
- **Password Hashing:** If the validation passes, it hashes the new password using bcrypt.
- **Database Update:** It updates the user's password in the database with the hashed password.

## 23.Add Item to Wishlist

- **Purpose:** Adds a product to a user's wishlist.
- **Process:**
- **Input:** Expects productId from the request body and retrieves userId from the logged user.
- **Validation:** Checks if productId is valid.
- **Check Wishlist:** Looks for an existing wishlist for the user:
  - If found, checks if the product is already in the wishlist. If it is, returns an error.
  - If not found, creates a new wishlist.
- **Update:** Adds the productId to the wishlist and saves it.

## 24.Remove Item from Wishlist

- **Purpose:** Removes a product from a user's wishlist.
- **Process:**
- **Input:** Expects productId from the request body and retrieves userId.
- **Validation:** Checks if productId is valid.
- **Check Wishlist:** Looks for the user's wishlist:
  - If not found, returns an error.
- **Remove:** Checks if the product is in the wishlist and removes it if found. If not found, returns an error.



The background of the slide is a photograph of a modern building's exterior. The building features a curved, metallic facade with horizontal lines, possibly a series of panels or a ribbed structure. The sky is a clear, deep blue. The text "Thank You" is centered over the image.

# Thank You