Backend 3.2_CW Exercises

for collections

ex01: add description to a collection:

challenge

- In the right panel, you'll see the collection details.
- In the "Description" field, add collection description:

  Variations of GET Requests This collection guides you through various exercises involving GET requests, including querying data, using route parameters, handling headers, and implementing error handling. The exercises are designed to help you understand how to interact with RESTful APIs using different types of GET requests.

  🏷️ How to use this collection:

  Step 1: Send requests Explore and practice CRUD operations (GET) by using the various GET requests included in this collection. Each request demonstrates a specific concept.

  Step 2: View responses After sending requests, observe the response tab for valuable information such as status codes (e.g., 200 OK), response time, and data size.

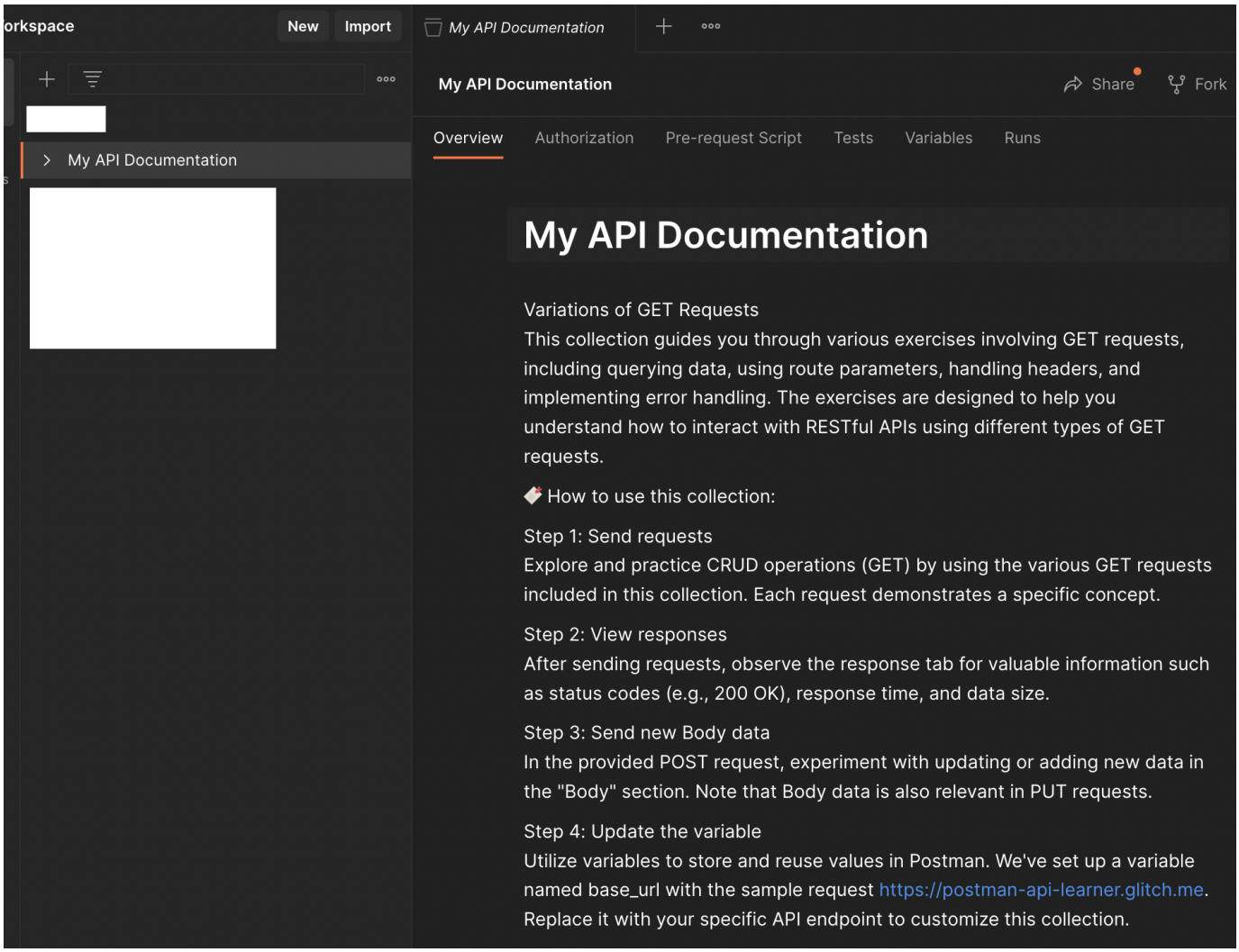
  Step 3: Send new Body data In the provided POST request, experiment with updating or adding new data in the "Body" section. Note that Body data is also relevant in PUT requests.

  ```
  { "name": "Add your name in the body" }
  ```

  COPY

  Step 4: Update the variable Utilize variables to store and reuse values in Postman. We've set up a variable named base_url with the sample request https://postman-api-learner.glitch.me. Replace it with your specific API endpoint to customize this collection.

- Solution



for folders

ex02: add description to folder

challenge

- Create a new folder “Car Inventory management”.
- Click on the newly created folder.
- In the right panel, you'll see the folder details.
- In the "Description" field, copy and paste the provided folder description:

Cars Model

The Cars model represents vehicles in our inventory. It provides essential information about each car, allowing users to retrieve details, filter results, and understand the characteristics of the vehicles available.

Attributes:

- `id`: This property is a unique identifier for each car. It's a numeric value that distinguishes one car from another.
- `make`: The make of the car represents the manufacturer or brand name. It's a string that identifies the car's brand, such as "Toyota," "Honda," or "Ford."
- `model`: The model of the car specifies the specific model name. It's a string that describes the car's model, such as "Camry," "Civic," or "Mustang."
- `year`: The year indicates the manufacturing year of the car. It's a numeric value that represents the year the car was produced.

Functionality:

- `GET All Cars`: Retrieve a list of all cars in the inventory. This function returns an array of car objects, each containing the car's id, make, model, and year.
- `GET Car by ID`: Retrieve detailed information about a specific car based on its unique id. The response includes the car's make, model, and year.
- `Filter Cars`: Users can filter cars based on certain criteria, such as make or model. This allows them to narrow down the search and find cars that match their preferences.
- `Error Handling`: In cases where no cars match the specified filters, the API responds with an error message to indicate that no cars were found.
- `Headers`: Some special requests can be made using headers. For instance, users can request to retrieve featured cars with certain criteria based on headers.

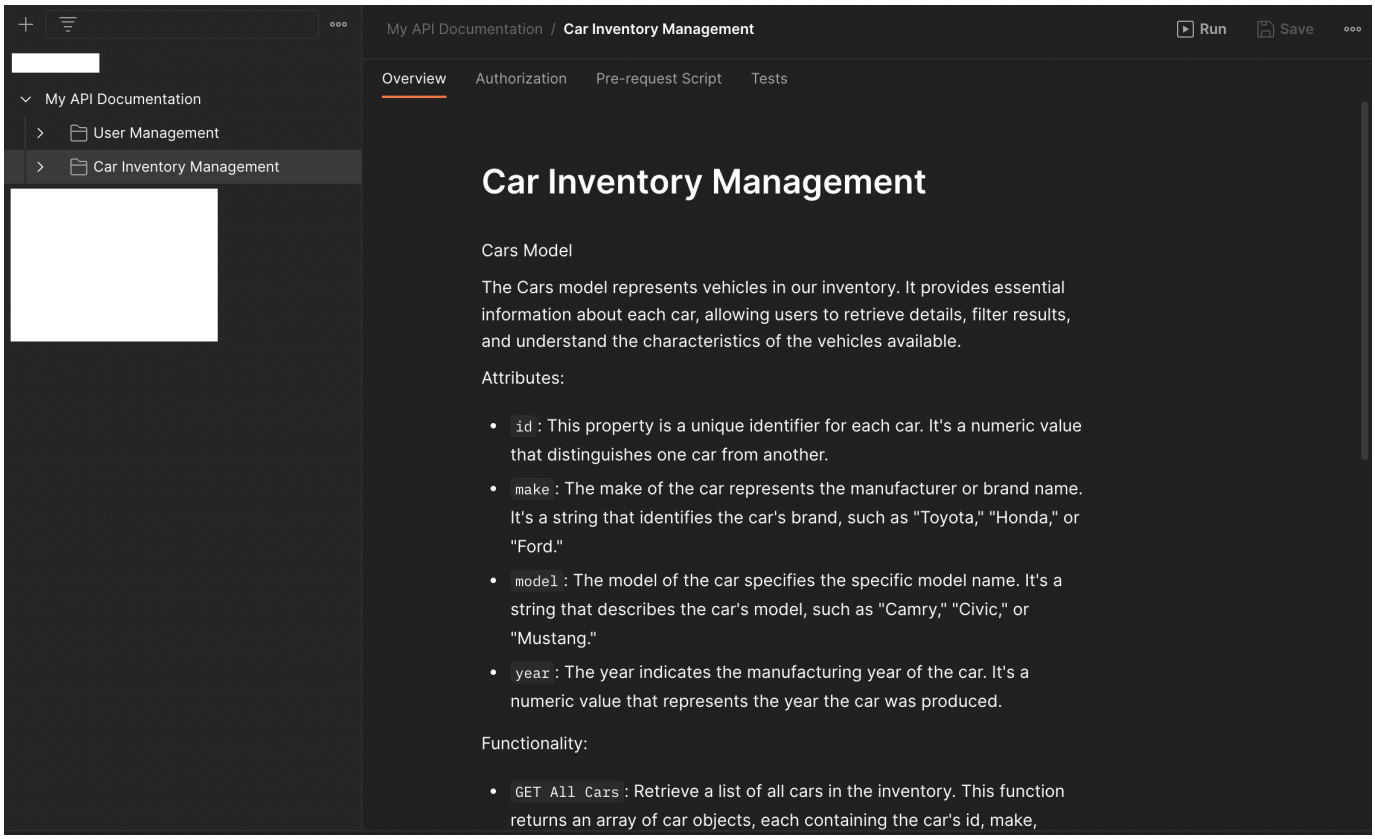
Example JSON:

Here's an example JSON representation of a car object:

```
{ "id": 1, "make": "Toyota", "model": "Camry", "year": 2022 }
```

COPY

- Solution



ex01.1: users

challenge

Add description to the user management folder you created.

ex01.2: movie

challenge

Create a new Movies collection.

Add description to the movies collection.