URI: /user

Method: GET

ip:port/user/<id>

Description: returns the user's data.

Parameters:

id	<id>	User ID

Fields for a valid response:

Key	Value	Description
id	<id>	User ID
username	<username>	User's username
email	<email>	User's email
password	<SHA256 hash of password>	Hash of User's password

Method: POST

Description: Creates a new user in the database.

Body Parameters:

Key	Value	Description
command	"create"	-
id	<id>	User ID
username	<username>	User's username
email	<email>	User's email
password	<password>	User's password

Edge cases:

1. All fields are required.
2. No need to worry about the validity of an email, or the strength of the password.
3. Users will be identified using <id>. Respond with status code 409 if you encounter a duplicate id.

Fields for a valid response:

Key	Value	Description
id	<id>	User ID
username	<username>	User's username
email	<email>	User's email
password	<SHA256 hash of password>	Hash of User's password

Edge cases:

1. Don't use status code 201, just stick to 200 for a valid creation.

Method: POST
Description: Updates a current user in the database.

Body Parameters:

Key	Value	Description
command	"update"	-
id	<id>	User ID
username	<username>	User's username
email	<email>	User's email
password	<password>	User's password

Edge cases:

1. All of the fields are not required. Only update the ones that are provided.
2. Command and id fields are required. Sending a payload with only these two fields will still result in a success (i.e. no changes to db).
3. There is no authentication (i.e. we don't require a valid password to make the updates)

Fields for a valid response:

Key	Value	Description
id	<id>	User ID
username	<username>	User's username
email	<email>	User's email
password	<SHA256 hash of password>	Hash of User's password

Edge cases:

1. Within the response, always provide all the fields.

Method: POST
Description: Deletes a user in the database.

Key	Value	Description
command	“delete”	-
id	<id>	User ID
username	<username>	User’s username
email	<email>	User’s email
password	<password>	User’s password

Edge cases:

1. All of the fields are required.
2. There is authentication (i.e. we require valid entries to make a deletion).

Fields for a valid response: Just an empty response is enough with the correct status code.

Edge cases:

1. Don’t use status code 204, just stick to 200 for a valid deletion.
2. The tests for delete (and even the other POST requests) could be followed up with a GET request to check that your api implementation works as intended. So, prioritize the GET request and ensure that it works for the best results. <- Applies to all the URI’s

URI: /product

Method: GET
ip:port/product/<id>
Description: returns the product data.

Parameters:

id	<id>	Product ID

Fields for a valid response:

Key	Value	Description
id	<id>	Product ID
name	<name>	Product Name
description	<description>	Product Description
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Method: POST
Description: Creates a new product in the database.

Body Parameters:

Key	Value	Description
command	"create"	-
id	<id>	Product ID
name	<name>	Product Name
description	<description>	Product Description
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Edge cases:

1. All fields are required.
2. Quantity and price cannot be negative.
3. Products will be identified using <id>, so send a response with a 409 status code if you encounter a duplicate ID.

Fields for a valid response:

Key	Value	Description
id	<id>	Product ID
name	<name>	Product Name
description	<description>	Product Description
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Edge cases:

1. Don't use status code 201, just stick to 200 for a valid creation.

Method: POST
Description: Updates a current user in the database.

Body Parameters:

Key	Value	Description
command	"update"	-
id	<id>	Product ID
name	<name>	Product Name
description	<description>	Product Description
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Edge cases:

1. All of the fields are not required. Only update the ones that are provided.
2. Command and id fields are required for a valid request. Sending a payload with only these two fields will still result in a success (i.e. no changes to db).

Fields for a valid response:

Key	Value	Description
id	<id>	Product ID
name	<name>	Product Name
description	<description>	Product Description
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Edge cases:

1. Within the response, always provide all of the fields.

Method: POST
Description: Deletes a product in the database.

Key	Value	Description
command	"delete"	-
id	<id>	Product ID
name	<name>	Product Name
price	<price>	Product Price
quantity	<quantity>	Product's quantity in stock

Edge cases:

1. All of the fields are required.

Fields for a valid response: Just an empty response is enough with the correct status code.

Edge cases:

1. Don't use status code 204, just stick to 200 for a valid deletion.

URI: /order

Method: POST
Description: Creates a new order in the database.

Body Parameters:

Key	Value	Description
command	“place order”	-
product_id	<product_id>	ID of the product that was purchased
user_id	<user_id>	ID of who ever is ordering <product_id>
quantity	<quantity>	# of products the user wants to buy.

Edge cases:

1. All fields are required for a valid order.
2. Quantity cannot go over the available limit.

Fields for a valid response:

Key	Value	Description
id	<id>	Order ID
product_id	<product_id>	ID of the product that was purchased
user_id	<user_id>	ID of who ever is ordering <product_id>
quantity	<quantity>	# products the user wants to buy.
status	<status message>	Status of the order. Should always be one of the below.

Edge cases:

1. Don’t use status code 201, just stick to 200 for a valid creation.
2. Status message: “Exceeded quantity limit”, “Success”, “Invalid Request”.

Status codes you should think about: <- applies to all URI's
200 - "OK",
400 - "Bad Request",
401- "Unauthorized",
404 - "Not Found",
409 - "Conflict",
500 - "Internal Server Error"

You may not need to have all of these for every api endpoint.

If you ever encounter an invalid payload for the user and product endpoints, the response should look like the following: {}

If you ever encounter an invalid payload for the order end point, you need to add the status field, as follows:
{
"status": "<Status message>"
}

Edge case: Let status code 400 take priority over 401, 404, 409.

Important note: Ensure that you respond with an internal server error (500 Status code) in case something goes wrong internally, so that the entirety of your api can continue to receive and handle other requests. You can do this using try catch statements.