

**IS 213: Enterprise Solution Development**

**G1 | Group 7**

**Term 2 AY 2021/2022**

**Instructor: Jiang Ling Xiao**

**Group Members:**

**Xu Zhehai**

**Goh Lin Kai**

**Tan Qing Yu**

**Lee Car Men**

**Tan Yee Khai**

**Tian Feng**

# Introduction

Gym-goers often like to keep things simple. For example, there are countless sources of protein from food from fishes to nuts but gym-goers all over the world only stick to the good old chicken breast. Unfortunately, this love for simplicity has not been replicated on websites selling gym equipment; our team found they are often filled with a plethora of useless information and variable add-ons meant to squeeze more out of the customer. To tackle this pain point, our team came up with HUUGE, a one-stop, lightweight e-commerce platform for like-minded potential gym-goers to have affordable clothing and gym equipment they can sweat in, finding inspiration in and a community they can become best at. On our platform you will find all sorts of workout equipment you will ever need to embark on your fitness journey; regardless of whether your end goal is to become Mr Olympia, a national powerlifter, or just to trim belly fats accumulated during quarantine. Whether you’re the store owner, a delivery driver or a customer, our simple and crisp UI ensures only critical information is displayed and you are not overloaded with unnecessary information. The cherry on top - we provide free delivery too!

# Technical Overview Diagram(s)

# 

# User Scenarios

## Scenarios 1 - User checkouts the order Using Cart

| *Figure 1 - User check out using cart* |
| --- |

**Figure 1**

**(Prerequisite: customer need to have at least one item inside the cart)**

1. Customer uses the UI to check out the cart.
2. UI will invoke the Place an Order complex microservice which then will invoke the Orders service, via [POST] /orders/create to create a new order.
3. Orders service will return the order information to Place an Order complex microservice which then invokes Payment service, via [POST] /payment
4. Payment service will query the cart total amount chargeable to the customer and invoke Stripe API gateway to process the payment.
5. Customers will get redirected with Stripe payment URL and invoke the [POST]/process\_payment that invoke the [GET]/payment to check on payment status
6. Stripe API upon successful payment, it will redirect the customer back to our UI and the complex microservice continues to invoke the Orders service, via [POST] /orders/{order\_id} to update the payment status of the order.
7. Place an Order complex microservice will then invoke Inventory via [PUT]/inventory to update the inventory item quantity.
8. Customer cart will be cleared via [PUT]/updateCart/{custID}
9. Lastly, delivery services will be invoked via [POST]/delivery that create a order delivery record

### (Micro)Services

| Service Name | Operational information (e.g., HTTP URL or AMQP exchange type and keys, if any) | Description of the functionality | Link to API Docs in appendix / external web pages that describe detailed inputs/outputs |
| --- | --- | --- | --- |
| Cart | **[POST]** /newCart/{custID} | *Create a new cart for a customer*  *(Only triggers if the user access the cart page for the first time)* | [*API Documentation*](#_s9wdolhmzkm0) |
| **[PUT]** /updateCart/{custID} | *Update customer’s cart* | [*API Documentation*](#_7mri4xl3iv5j) |
| **[GET]** /GET/{custID} | *Get and retrieve customer’s cart items* | [*API Documentation*](#_but0rg3xeu1z) |
| *Place order* | **[POST]** /place\_order | *Place an order* | [*API Documentation*](#_l361mupqpox7) |
| **[POST]** /process\_payment | *Process the payment by waiting for the customer to make payment.* | [*API Documentation*](#_l361mupqpox7) |
| *order* | **[POST]** /order | *Create order and order\_item* | [*API Documentation*](#_oddm7x88lg1) |
| **[PUT]**/order/{order\_id} | *Update order status* | [*API Documentation*](#_4voncun8typj) |
| *payment* | **[POST]**/payment | *Create Stripe URL for payment* | [*API Documentation*](#_n4ip4s3d0l4i) |
| **[GET]**/payment | *Get payment status and update order service* | [*API Documentation*](#_gmwzz2snx63g) |
| *inventory* | **[PUT]**/inventory | *Update inventory item quantity* | [*API Documentation*](#_voc8dv9nhxmk) |
| *delivery* | **[POST]**/delivery | *Create delivery record* | [*API Documentation*](#_eicndx81od51) |

### Beyond the Labs

* Kong Service Routing - Improving the security by allow one or multiple routes pointed to a single service.
* Kong load balancer is a device that **acts as a reverse proxy and distributes network or application traffic across a number of servers**. Load balancers are used to increase capacity (concurrent users) and reliability of applications.

## Scenarios 2 - User checkouts using Product page (Buy Now)

| *Figure 2 - User Check out via Buy Now* |
| --- |

**Figure 2**

1. Customer uses the UI to check using the Product page using Buy Now.
2. Same steps as [Scenario 2 - Steps 2-9](#_bnej7rcx940y)

### (Micro)Services

| Service Name | Operational information (e.g., HTTP URL or AMQP exchange type and keys, if any) | Description of the functionality | Link to API Docs in appendix / external web pages that describe detailed inputs/outputs |
| --- | --- | --- | --- |
| Inventory | **[GET]**/inventory | *Get all the inventory items* | [*API Documentation*](#_bbqbm1szxuuv) |

### Beyond the Labs

* Kong Service Routing - Improving the security by allow one or multiple routes pointed to a single service.
* Kong load balancer is a device that **acts as a reverse proxy and distributes network or application traffic across a number of servers**. Load balancers are used to increase capacity (concurrent users) and reliability of applications.

## Scenarios 3 - Driver process order (Accept/delivered)

|  |
| --- |

**Figure 3**

**(Prerequisite: Driver need to sign in the account)**

The account has to be registered and approved as a Driver (role). The driver can accept the open order and after the driver has delivered the item to the doorstep to the customer, in the pending delivering order page, the driver can process the order to be delivered.

**Order & Delivery Status chart:**

| Order Status | 0 - Order created | 1 - Payment unsuccessful | 2 - Payment successful | 3 - Delivering | 4 - Delivered |
| --- | --- | --- | --- | --- | --- |
| Delivery Status |  |  | 0 - Preparing | 1 - Delivering | 2 - Delivered |

**Driver process the delivery (Accepts/Delivered the order)**

1. Driver starts by clicking on the All Open Orders button on the driver UI, then the UI invokes the Order service via HTTP POST to retrieve the list of available orders for delivery.
2. The UI will display all the orders that are available for delivery, and the user clicks on the accept button. The accept button triggers the complex microservice process delivery via HTTP POST and Delivery and Order service will be concurrently invoked via HTTP POST.
3. The Delivery service will capture and assign the driverName, driverID and change the status of the order records to 3/4 (Delivering/Delivered). And the Order service will capture the orderID and change the status of delivery records to 1/2 (Delivering/Delivered).
4. After the complex microservice process delivery completes, this will return a success message on the UI and reload the page.

### (Micro)Services

| Service Name | Operational information (e.g., HTTP URL or AMQP exchange type and keys, if any) | Description of the functionality | Link to API Docs in appendix / external web pages that describe the detailed inputs/outputs |
| --- | --- | --- | --- |
| Order | **[POST]** /order/status/{status} | Retrieve all available orders that has not been picked up. | [*API Documentation*](#_7z2yhfliu944) |
| **[PUT]** /order/orderID | Update the driver information and order status. | [*API Documentation*](#_vn1ydnvr31li) |
| Process-Delivery | **[POST]** /process-delivery | Process the delivery | [*API Documentation*](#_1cixwl5f568c) |
| Delivery | **[POST]** /delivery/status | Retrieve all delivery status that is either 1 (Delivering) or 2 (Delivered) | [*API Documentation*](#_j6wthruqp22q) |
| Delivery | **[PUT]** /delivery/orderID | Update the delivery status to either 1 or 2. | [*API Documentation*](#_fr7a6nkxcu03) |

**API Specification Document**

Index

[Authentication Method](#_8mhn48d7p5q6)

[a. Delivery](#_2et92p0)

[1. Get all delivery records.](#_6n7xjq8jbp0q)

[Request](#_tyjcwt)

[Response](#_3dy6vkm)

[2. Get a specific delivery record.](#_3i3whcz8xmof)

[Request](#_4d34og8)

[Response](#_15nvqyi5ngmg)

[3. Create a delivery record.](#_eicndx81od51)

[Request](#_3rdcrjn)

[Response](#_3x8zegqv5o3)

[4. Update delivery record.](#_fr7a6nkxcu03)

[Request](#_35nkun2)

[Response](#_1ksv4uv)

[b. Inventory](#_el7mqg7ctnjm)

[1. Get all inventory records.](#_bbqbm1szxuuv)

[Request](#_ona0dxazfmr1)

[Response](#_nbvzmaqeix0b)

[2. Find a product by its productID](#_da3va21krrbl)

[Request](#_qin5bjv1yrd0)

[Response](#_5ux1pq1bagj)

[3. Add a new product to the inventory.](#_3py0cv2ba9r1)

[Request](#_njt61zlgifzc)

[Response](#_sqkdes8lfqw5)

[4. Update inventory.](#_voc8dv9nhxmk)

[Request](#_soswtnqq85w9)

[Response](#_focwla3oxbls)

[5. Update inventory by individual items.](#_sqkyse6yop9y)

[Request](#_4cpf3awqpnfd)

[Response](#_2raa5c43e1x5)

[6. Delete equipment from inventory.](#_3zqm2hlkprt1)

[Request](#_rki491g4rwy9)

[Response](#_a92jgceva2xs)

[c. Order](#_stcxm0pk5i69)

[1. Get all order records.](#_300ox25dgikc)

[Request](#_7sqx5by3dsws)

[Response](#_2n9uhmg0f206)

[2. Find an order by its order\_ID](#_glon5mg48vy3)

[Request](#_16q1cfgeg732)

[Response](#_rzq6m6istwht)

[3. Find all orders with a specified status](#_7z2yhfliu944)

[Request](#_4b4y61fpumol)

[Response](#_n65cwd87cmdp)

[4. Create order.](#_oddm7x88lg1)

[Request](#_brls70bn30tf)

[Response](#_7fuq9dof6319)

[5. Update status of an order for a specified order\_ID](#_4voncun8typj)

[Request](#_sn5cehpudiie)

[Response](#_vpax8zlnbgmd)

[6. Update delivery address of an order for a specified order\_ID](#_vn1ydnvr31li)

[Request](#_se84unkg7o6k)

[Response](#_nax8zf1exdwv)

[d. Payment](#_xe2ax4nky955)

[1. Create payment request](#_n4ip4s3d0l4i)

[Request](#_pqrqyydba8uj)

[Response](#_qbvhncotq5dj)

[2. Get payment status and update order](#_gmwzz2snx63g)

[Request](#_c7t19fal20rj)

[Response](#_dwlr1u2xzxyy)

[e. Cart](#_f8e2gf1nyxo7)

[1. Create a new cart for customer](#_s9wdolhmzkm0)

[Request](#_wo4cnx6bka5d)

[Response](#_o9q431b85xzu)

[2. Retrieve cart for customer](#_but0rg3xeu1z)

[Request](#_nrlp5s1uhn5z)

[Response](#_d1agmxx6i0zo)

[3. Update cart for customer](#_7mri4xl3iv5j)

[Request](#_mrdd1lkmpkvi)

[Response](#_kz33dx8al01q)

[e. Place Order](#_l361mupqpox7)

[1. Customer places an order.](#_bb1k4y3s5dcc)

[Request](#_sgadado76vy9)

[Response](#_zhxqkita9uqk)

[f. Process Delivery](#_1cixwl5f568c)

[1. Process Delivery](#_rgpp61dz2sv2)

[Request](#_k15e2bbtc7uj)

[Response](#_vnsndv5zpfgi)

[Conventions](#_1y810tw)

[Status Codes](#_rpv0956xhm11)

Authentication

## **Authentication Method**

Our application uses Flask’s built-in user session management **flask-login.**

Our authentication system consists of the following:

1. Users
2. Password Hashing System
3. Permissions (Access Control)
4. Password Strength Checking
5. Session

**Users**

The attributes of users are:

1. email (**string**) - unique
2. username (**string**) - unique
3. password (**string**)
4. delivery (**string**) - delivery address of the user

**Password Hashing System**

Our application uses Werkzeug’s **generate\_password\_hash** and **check\_password\_hash** to hash passwords when users sign up.

Sign-up:

hashed\_password = generate\_password\_hash(

form.password.data, method='sha256')

Log-in:

if check\_password\_hash(user.password, form.password.data):

login\_user(user, remember=False)

**Permissions (Access Control)**

Users have to be authenticated before they are granted access to access-controlled pages. For example, the user’s role must be “Driver” for them to access /driver/ pages. If an unauthorized user attempts to access an access-controlled page, a warning will flash that they’re not authorized to view the page and the user will be automatically kicked out of the page.

@app.route('/driver-accepted')

@login\_required

@role\_required('Driver') # user role required

def driver\_accepted\_orders():

return render\_template('driver/accepted-order.html', name=current\_user.username, id=current\_user.id, email=current\_user.email)

if current\_user.role != role:

flash('role unauthorized to view, forced logout')

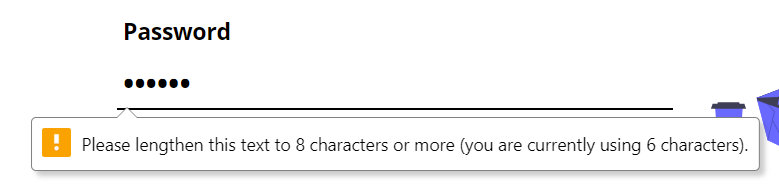
return login\_manager.unauthorized()

**Password Strength Checking**

Our authentication system requires the password to satisfy all the following conditions:

* At least 1 upper-case character
* At least 1 lower-case character
* At least 1 special character (@$!#%\*?&)
* Between 8 to 16 characters

The UI will return the user an error message and a suggestion if the password does not satisfy all the conditions. For example, if the user tries to sign up with a password that is only 6 characters long, the following error is displayed.



**Session**

Flask uses volatile sessions, where the session cookie expires when the browser is closed. However, we want our users’ experience to be seamless hence we introduced a 10-minute session instead where users do not have to log back in if they close their browser as long as it is within a 10-minute timeframe of logging in.

@app.before\_request

def timeout\_before\_request():

session.permanent = True

app.permanent\_session\_lifetime = timedelta(

minutes=10)

session.modified = True

Methods

# **a. Delivery**

## **1. Get all delivery records.**

Get all the delivery records that are currently in the database.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/delivery/ |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of all the delivery records is given**  {  "code": 200,  "data": {  "delivery": {  "deliveryID": 001,  "driverID": 001,  "driverName": "Sam Low",  "dateCreated": "2022-03-31 09:41:42.221432",  "deliveryStatus": "1",  "orderID": "001"  },  …  {  "deliveryID": 002,  "driverID": 001,  "driverName": "Sam Low",  "dateCreated": "2022-03-31 09:49:12.210011",  "deliveryStatus": "1",  "orderID": "002"  }]  }  }  deliveryID (**integer**) - unique identifier for the delivery  driverID (**integer**) - identifier for the driver  driverName(**string**) - name of the driver  dateCreated(**DateTime**) - date and time the delivery request was created  deliveryStatus(**string**) - status of the delivery (0 = preparing, 1 = out  for delivery, 2 = delivered)  orderID(**string**) - identifier for the order |
| 404 | Description - There are no delivery records.  {  "code": 404,  "message": "There is no delivery."  } |

## **2. Get a specific delivery record.**

Retrieve a specific delivery record using the orderID.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/delivery/<orderID> |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| GET | orderID | string | Yes |

**order\_id**

ID of the order you want to update the delivery status for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the delivery record requested is given**  {  "code": 200,  "data": {  "deliveryID": 001,  "driverID": 001,  "driverName": "Sam Low",  "dateCreated": "2022-03-31 09:41:42.221432",  "deliveryStatus": "1",  "orderID": "001"  }  }  deliveryID (**integer**) - unique identifier for the delivery  driverID (**integer**) - identifier for the driver  driverName(**string**) - name of the driver  dateCreated(**DateTime**) - date and time the delivery request was created  deliveryStatus(**string**) - status of the delivery (0 = preparing, 1 = out  for delivery, 2 = delivered)  orderID(**string**) - identifier for the order |
| 404 | Description - Order not found.  {  "code": 404,  "data": {  "orderID": "001"  },  "message": "Order not found."  } |

## **3. Create a delivery record.**

After payment is successful, a delivery record will be created for the orderID. No driver has been assigned yet, so by default the delivery status will be 0 (preparing).

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/delivery |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| POST | orderID | string | Yes |

**order\_id**

ID of the order you want to create the delivery record for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the delivery record created is given**  Example response:-  {  "code": 200,  "data": {  "deliveryID": 001,  "driverID": NULL  "driverName": NULL  "dateCreated": "2022-03-31 09:41:42.221432",  "deliveryStatus": "0",  "orderID": "001"  }  } |
| 500 | Description - An error occurred while creating the order.  {  "code": 500,  "message": "An error occurred while creating the order. NameError"  } |

## **4. Update delivery record.**

After the driver has self-assigned an orderID or delivered the item, the driver will update the delivery record to reflect the delivery status accordingly.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/delivery/<orderID> |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| PUT | orderID | string | Yes |

**order\_id**

ID of the order you want to update the delivery record for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the delivery record updated is given**  Example response:-  {  "code": 200,  "data": {  "deliveryID": 001,  "driverID": 001,  "driverName": "Sam Low",  "dateCreated": "2022-03-31 09:41:42.221432",  "deliveryStatus": "1",  "orderID": "001"  }  } |
| 404 | Description - Order not found.  {  "code": 404,  "data": {  "orderID": "001"  },  "message": "Order not found."  } |
| 500 | Description - An error occurred while updating the order.  {  "code": 500,  "data": {  "orderID": "001"  },  "message": "An error occurred while updating the order. NameError"  } |

## **2. Get a delivery record based on delivery status.**

Retrieve all delivery records using delivery status.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/delivery/status/ |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| POST | driverID | string | Yes |
| POST | deliveryStatus | String | Yes |

**driverID and deliveryStatus**

ID of the driverID and matching the deliveryStatus you want to get the orders for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the delivery record requested is given**  {  "code": 200,  "data": {  "deliveryID": 001,  "driverID": 001,  "driverName": "Sam Low",  "dateCreated": "2022-03-31 09:41:42.221432",  "deliveryStatus": "1",  "orderID": "001"  }  }  deliveryID (**integer**) - unique identifier for the delivery  driverID (**integer**) - identifier for the driver  driverName(**string**) - name of the driver  dateCreated(**DateTime**) - date and time the delivery request was created  deliveryStatus(**string**) - status of the delivery (0 = preparing, 1 = out  for delivery, 2 = delivered)  orderID(**string**) - identifier for the order |
| 404 | Description - Order not found.  {  "code": 404,  "data": {  "orderID": "001"  },  "message": "Orders not found for drivers."  } |

# **b. Inventory**

## **1. Get all inventory records.**

Get all the inventory records that are currently in the database.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/inventory/ |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of all the inventory records is given**  Example response:-  {  "code": 200,  "data": {  "inventories": [{  "productID": 001,  "name": "20kg Barbell",  "product\_type": "Weights",  "price": 99.99,  "quantity": 42  },  …  {  "productID": 052,  "name": "25kg Plate",  "product\_type": "Weights",  "price": 169.99,  "quantity": 40  }]  }  }  }  productID (**integer**) - unique identifier for the product  name (**string**) - name of the product  product\_type(**string**) - category of the product  price(**float**) - price of the product in 2 decimal places  quantity(**integer**) - quantity of the product in the inventory |
| 404 | Description - There is no inventory.  {  "code": 404,  "message": "There is no inventory."  } |

## **2. Find a product by its productID**

Find a product that matches the productID queried.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/inventory/productID |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| GET | productID | integer | Yes |

**productID**

ID of the product you want to retrieve.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the inventory record that matches the product ID is given**  Example response:-  {  "code": 200,  "data": {  "productID": 001,  "name": "20kg Barbell",  "product\_type": "Weights",  "price": 99.99,  "quantity": 42  }  }  } |
| 404 | Description - No equipment matching the productID was found.  {  "code": 404,  "message": "Equipment not found."  } |

## **3. Add a new product to the inventory.**

Add a new product to the inventory using its productID.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/inventory/productID |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| POST | productID | integer | Yes |

**productID**

ID of the product you want to add to the inventory.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the inventory record added is given**  Example response:-  {  "code": 201,  "data": {  "productID": 003,  "name": "40kg Barbell",  "product\_type": "Weights",  "price": 169.99,  "quantity": 20  }  }  } |
| 400 | Description - productID already exists, which signals equipment already exists.  {  "code": 400,  "data": {  "productID": 003  },  "message": "Equipment already exists."  } |
| 500 | Description - An error occurred while creating the inventory.  {  "code": 500,  "data": {  "productID": 003  },  "message": "An error occurred creating the inventory."  } |

## **4. Update inventory.**

Update inventory with new equipment quantities after purchase

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/inventory |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of the updated inventory records is**  **given**  Example response:-  {  "code": 200,  "data": {  "inventories": [{  "productID": 001,  "name": "20kg Barbell",  "product\_type": "Weights",  "price": 99.99,  "quantity": 22  },  …  {  "productID": 052,  "name": "25kg Plate",  "product\_type": "Weights",  "price": 189.99,  "quantity": 12  }]  }  }  } |
| 404 | Description - No equipment was found in the inventory.  {  "code": 404,  "data": {  "productID": 001  },  "message": "Equipment not found."  } |

## **5. Update inventory by individual items.**

Update a specific equipment with a new quantity.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/updateInventory/{productID} |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| PUT | productID | integer | Yes |

**productID**

ID of the order you want to update the delivery record for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the updated inventory record of the specific item is given**  Example response:-  {  "code": 201,  "data": {  "productID": 001,  "name": "20kg Barbell",  "product\_type": "Weights",  "price": 99.99,  "quantity": 17  }  }  } |
| 404 | Description - No equipment was found in the inventory.  {  "code": 404,  "data": {  "productID": 001  },  "message": "Equipment not found."  } |

## **6. Delete equipment from inventory.**

Delete a specific equipment from the inventory.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **DELETE** | gym/inventory/productID |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| DELETE | productID | integer | Yes |

**productID**

ID of the product you want to delete from the inventory..

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The productID of the equipment that has been deleted is given**  Example response:-  {  "code": 201,  "data": {  "productID": 051  }  }  } |
| 404 | Description - No equipment was found in the inventory with that productID.  {  "code": 404,  "data": {  "productID": 051  },  "message": "Equipment not found."  } |

# **c. Order**

## **1. Get all order records.**

Get all the order records that are currently in the database.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/order/ |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of all the order records is given**  Example response:-  {  "code": 200,  "data": {  "orders": [{  "order\_id": 001,  "cust\_id": "001",  "date\_created": "2022-03-31 10:44:42.241112",  "total\_price": 339.98,  "delivery\_address": "70 Stamford Road, Singapore fffffff178901",  "status": "1",  "modified": "2022-03-31 10:44:42.241112"  },  …  {  "order\_id": 009,  "cust\_id": "009",  "date\_created": "2022-03-31 14:21:12.372612",  "total\_price": 99.99,  "delivery\_address": "100 Victoria St, Singapore fffffff188064",  "status": "2",  "modified": "2022-03-31 10:44:42.241112"  }]  }  }  }  order\_id (**integer**) - unique identifier for the order  cust\_id (**string**) - identifier for the customer  date\_created (**string**) - date and time the order was created  total\_price (**float**) - total price of the order  delivery\_address(**string**) - delivery address of the customer who placed the order  status (**string**) - status of the delivery (0 = preparing, 1 = out  for delivery, 2 = delivered)  modified (**DateTime**) - date and time of the last modification to the order |
| 404 | Description - There are no orders in the database.  {  "code": 404,  "message": "There are no orders."  } |

## **2. Find an order by its order\_ID**

Get the order record of a specific order by its order\_ID.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/order/order\_id |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| GET | order\_id | integer | Yes |

**order\_ID**

ID of the order you want to find.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of all the order queried is given**  Example response:-  {  "code": 200,  "data": {  "order\_id": 001,  "cust\_id": "001",  "date\_created": "2022-03-31 10:44:42.241112",  "total\_price": 339.98,  "delivery\_address": "70 Stamford Road, Singapore fffffff178901",  "status": "1",  "modified": "2022-03-31 10:44:42.241112"  }  }  } |
| 404 | Description - There are no orders matching the order\_ID in the database.  {  "code": 404,  "data": {  "order\_id": 001  },  "message": "Order not found."  } |

## **3. Find all orders with a specified status**

Get the order records of all orders with a specified status

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/order/status |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| GET | status | string | Yes |

**status**

Status you want to find all the orders for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the order created is given**  Example response:-  {  "code": 200,  "data": {  "order\_id": 001,  "cust\_id": "001",  "date\_created": "2022-03-31 10:44:42.241112",  "total\_price": 339.98,  "delivery\_address": "70 Stamford Road, Singapore fffffff178901",  "status": "1",  "modified": "2022-03-31 10:44:42.241112"  }  }  } |
| 500 | Description - There are no orders in the database.  {  "code": 404,  "message": "There are no orders."  } |

## **4. Create order.**

Create an order.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/order |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of all the order records with the specified status is given**  Example response:-  {  "code": 200,  "data": {  "orders": [{  "order\_id": 007,  "cust\_id": "007",  "date\_created": "2022-03-31 10:31:41.512542",  "total\_price": 339.98,  "delivery\_address": "23 Serangoon Central, Singapore fffffff556083",  "status": "2",  "modified": "2022-03-31 10:44:42.241112"  },  …  {  "order\_id": 009,  "cust\_id": "009",  "date\_created": "2022-03-31 14:21:12.372612",  "total\_price": 99.99,  "delivery\_address": "100 Victoria St, Singapore fffffff188064",  "status": "2",  "modified": "2022-03-31 10:44:42.241112"  }]  }  }  } |
| 500 | Description - An error occurred while creating the order.  {  "code": 500,  "message": "An error occurred creating the order. NameError"  } |

## **5. Update status of an order for a specified order\_ID**

Update the status of the order with a specified order\_ID.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/order/order\_id |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| PUT | order\_id | integer | Yes |

**order\_id**

order\_id of the order you want to update.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the order updated is given**  Example response:-  {  "code": 200,  "data": {  "order\_id": 001,  "cust\_id": "001",  "date\_created": "2022-03-31 10:44:42.241112",  "total\_price": 339.98,  "delivery\_address": "70 Stamford Road, Singapore fffffff178901",  "status": "2",  "modified": "2022-03-31 11:21:42.512412"  }  }  } |
| 404 | Description - There are no orders matching the order\_ID in the database.  {  "code": 404,  "data": {  "order\_id": 001  },  "message": "Order not found."  } |
| 500 | Description - There was an error while updating the order.  {  "code": 500,  "data": {  "order\_id": 001  },  "message": "An error occurred while updating the order. NameError"  } |

## **6. Update delivery address of an order for a specified order\_ID**

Update the status of the order with a specified order\_ID.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/order/order\_id |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| PUT | order\_id | integer | Yes |

**order\_id**

order\_id of the order you want to update.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the order updated is given**  Example response:-  {  "code": 200,  "data": {  "order\_id": 001,  "cust\_id": "001",  "date\_created": "2022-03-31 10:44:42.241112",  "total\_price": 339.98,  "delivery\_address": "30 Victoria St, Singapore 187996",  "status": "2",  "modified": "2022-03-31 11:21:42.512412"  }  }  } |
| 404 | Description - There are no orders matching the order\_ID in the database.  {  "code": 404,  "data": {  "order\_id": 001  },  "message": "Order not found."  } |
| 500 | Description - There was an error while updating the order.  {  "code": 500,  "data": {  "order\_id": 001  },  "message": "An error occurred while updating the order. NameError"  } |

# **d. Payment**

## **1. Create payment request**

Return the Stripe payment URL for an order.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/payment |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the payment request is given**  Example response:-  {  "code": 201,  "data": {  "token": "plink\_1KjEfV2eZvKYlo2ClZlYcowq"  "orderID": 001  "payment\_url": "https://buy.stripe.com/test\_5kAcPyaI"  }  }  token (**string**) - tokenized sensitive information for payment  orderID (**integer**) - identifier for the order  payment\_url (**string**) - url to make payment on Stripe |
| 500 | Description - Could not connect to Stripe.  {  "code": 500,  "message": "An error occurred while connecting to Stripe."  } |
| 500 | Description - An error occurred while creating the transaction.  {  "code": 500,  "message": "An error occurred creating the transaction."  } |

## **2. Get payment status and update order**

Get the payment status and update the respective order

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/payment |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **The data of the payment result is given**  Description - There was an error while updating the order.  Example response:-  {  "code": 200,  "data": {  "token": "plink\_1KjEfV2eZvKYlo2ClZlYcowq"  "orderID": 001  "status": "2"  }  }  status (**string**) - 1 = open or expired, 2 = complete |
| 500 | Description - An error occurred while creating the transaction.  {  "code": 500,  "message": "An error occurred updating order status."  } |

# **e. Cart**

## **1. Create a new cart for customer**

Create a new cart for the customer using custId

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/newCart/custId |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| POST | custId | string | Yes |

**custId**

custId of the customer you want to create a cart for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **An array containing the data of the cart is given**  Example response:-  {  "code": 201,  "data": [{  "price": 99.99  "quantity": 1  "productID": "001"  "productName": "20kg Barbell"  },  …  {  "price": 169.99  "quantity": 1  "productID": "003"  "productName": "40kg Barbell"  }]  }  }  price (**integer**) - price of the product  quantity (**integer**) - quantity of the product  productID (**string**) - unique identifier or the product  productName (**string**) - name of the product |
| 400 | Description - Customer already has a cart.  {  "code": 400,  "message": "Cart already create for user 001"  } |
| 500 | Description - An error occurred creating the cart for the user.  {  "code": 500,  "message": "An error occurred creating the cart for user 001."  } |

## **2. Retrieve cart for customer**

Retrieve the existing cart for the customer using custId

### **Request**

| **Method** | **URL** |
| --- | --- |
| **GET** | gym/newCart/custId |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| GET | cusdId | string | Yes |

**custId**

custId of the customer you want to retrieve the cart for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | **An array containing the data of the cart is given**  Example response:-  {  "code": 200,  "data": {  "price": 99.99  "quantity": 1  "productID": "001"  "productName": "20kg Barbell"  },  …  {  "price": 169.99  "quantity": 1  "productID": "003"  "productName": "40kg Barbell"  }]  }  } |
| 400 | Description - Customer does not have an existing cart.  {  "code": 400,  "message": "No cart found for user 001"  } |

## **3. Update cart for customer**

Update the existing cart for the customer using custId

### **Request**

| **Method** | **URL** |
| --- | --- |
| **PUT** | gym/updateCart/custId |

| **Type** | **Params** | **Values** | **Required?** |
| --- | --- | --- | --- |
| PUT | cusdId | string | Yes |

**custId**

custId of the customer you want to update the cart for.

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **An array containing the data of the updated cart is given**  Example response:-  {  "code": 201,  "data": {  "price": 99.99  "quantity": 2  "productID": "001"  "productName": "20kg Barbell"  },  …  {  "price": 169.99  "quantity": 2  "productID": "003"  "productName": "40kg Barbell"  }]  }  } |
| 500 | Description - An error occurred updating the cart.  {  "code": 500,  "message": "An error occurred updating the cart."  } |

# **e. Place Order**

## **1. Customer places an order.**

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/place\_order |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | Description - paymentURL created  {  "code": 201,  "orderResult": [orderResult](https://docs.google.com/document/d/1VALafTpn7MJGzqOXXiv-bV-pUTXZILN0/edit#heading=h.7fuq9dof6319),  "paymentResult": [paymentResult](https://docs.google.com/document/d/1VALafTpn7MJGzqOXXiv-bV-pUTXZILN0/edit#heading=h.qbvhncotq5dj)  } |
| 400 | Description - JSON error.  {  "code": 400,  "message": "Invalid JSON input"  } |
| 500 | Description - Payment error.  {  "code": 500,  "message": "place\_order payment error"  } |
| 500 | Description - Place\_order internal error.  {  "code": 500,  "message": "place\_order internal error"  } |

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/process\_payment |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 200 | Description - Payment successfully processed.  {  "code": 200,  "data": {  "status" : deliveryResult,  "paymentURL" : paymentURl.com  }  } |
| 201 | Description - Payment successful, order’s payment status updated  {  "code": 201,  "orderStatus": [orderStatus](#_4voncun8typj)  } |
| 500 | Description - Payment error.  {  "code": 500,  "message": "process payment payment error"  } |

# **f. Process Delivery**

## **1. Process Delivery**

After the driver accepts and completes the orders, this will process the delivery.

### **Request**

| **Method** | **URL** |
| --- | --- |
| **POST** | gym/process-delivery |

### **Response**

| **Status** | **Response** |
| --- | --- |
| 201 | **The data of the processed delivery is given**  {  "code": 201,  "message": "Process delivery successful!"  "deliveryResult": "2",  "orderResult": "4",  }  deliveryResult (**string**) - 0 - preparing, 1 - delivering, 2- delivered  orderResult (**string**) - 3 - delivering, 4 - delivered |
| 400 | Description - JSON error.  {  "code": 400,  "message": "Invalid JSON input"  } |
| 500 | Description - Place\_order internal error.  {  "code": 500,  "message": "place\_order internal error"  } |

Glossary

## **Conventions**

* All the possible responses are listed under ‘Response’ for each method. Only one of them is issued per request server.
* All responses are in JSON format.

## **Status Codes**

All status codes are standard HTTP status codes. The below ones are used in this API.

2XX - Success of some kind

4XX - Error occurred in client’s part

5XX - Error occurred in server’s part

| **Status Code** | **Description** |
| --- | --- |
| 200 | OK |
| 201 | Created |
| 400 | Bad request |
| 404 | Resource not found |
| 500 | Internal Server Error |
| 503 | Service Unavailable |