



IS213 Enterprise Solution Development

G4 Team H

Project

Stephen Pang,

Lim Yin Shan,

Khoo Chee Kuang,

Gary Lim,

Isaac Einstein Fong,

Tan Hua Juan

**Table of Content**

[**Introduction**](#_gjdgxs)4

[**Technical Overview Diagram**](#_i9a8q4bsds3p)4

[**User Scenarios**](#_quq7otco60bn)5

[[User Scenario 1]: User goes to location with virtual box and opens it.](#_3j3e6mufl45f) 5

[(Micro)Services](#_xvra5a6zkti9) 5

[Beyond the labs](#_8ru4b16mu52z) 6

[[User Scenario 2]: Users plant a virtual treasure box at their location.](#_doi0dg2gclii) 6

[(Micro)Services](#_8elp8963w09e) 7

[Beyond the Labs](#_lsbtzalv2xmy) 7

[[User Scenario 3]: Users purchase premium membership with real-life money.](#_t2un77pr1zha) 8

[(Micro)Services](#_myt726qtylxb) 8

[Beyond the labs](#_dnt5m7i1cv1x) 8

[[User Scenario 4]: User purchase incentives from our in-game shop using virtual currency.   
(refer to User Scenario 4 in appendix)](#_epo2mqqm6o1v) 9

[Beyond the labs](#_of8pdxdm1dxa) 9

[[User Scenario 5]: Users view the leaderboard.  
(refer to User Scenario 5 in appendix)](#_sb4augbtf5bd) 9

[**Remaining Beyond the Labs not covered above**](#_bvmz5nsmctj0)9

[**Appendix**](#_php4y6ys4kw8)9

[**Architecture Diagram**](#_m7h1x9uk30h0)10

[**Extra User Scenario Diagrams**](#_9ky6ts8o0cc2)11

[[User Scenario 4]: User purchase incentives from our in-game shop using virtual currency.](#_fozuy8baq480) 11

[(Micro)Services](#_xgtjfyjma3ck) 11

[[User Scenario 5]: Users view the leaderboard.](#_x4t9mxhaaqwi) 12

[(Micro)Services](#_mr5ywjhg9hta) 12

[**SOA Diagram**](#_tz309e7vzafm)12

[**AMQP Communication Patterns Diagram**](#_90n789vyff4o)13

[**API Documentation**](#_alklh8ui12oa)13

[Activity API](#_wjsw1uxss0az) 13

[Error API](#_z9jrqbp4w1cv) 14

[Box API](#_t7gyv7j7gir) 15

[Box Opening API](#_fgdk0tjde6yw) 18

[Create Box API](#_r1maf67waen) 19

[Geolocation API](#_bmio7pf6ith2) 19

[Ingame Shop API](#_xua1suqyl7f) 20

[Order API](#_psdqlonu054g) 22

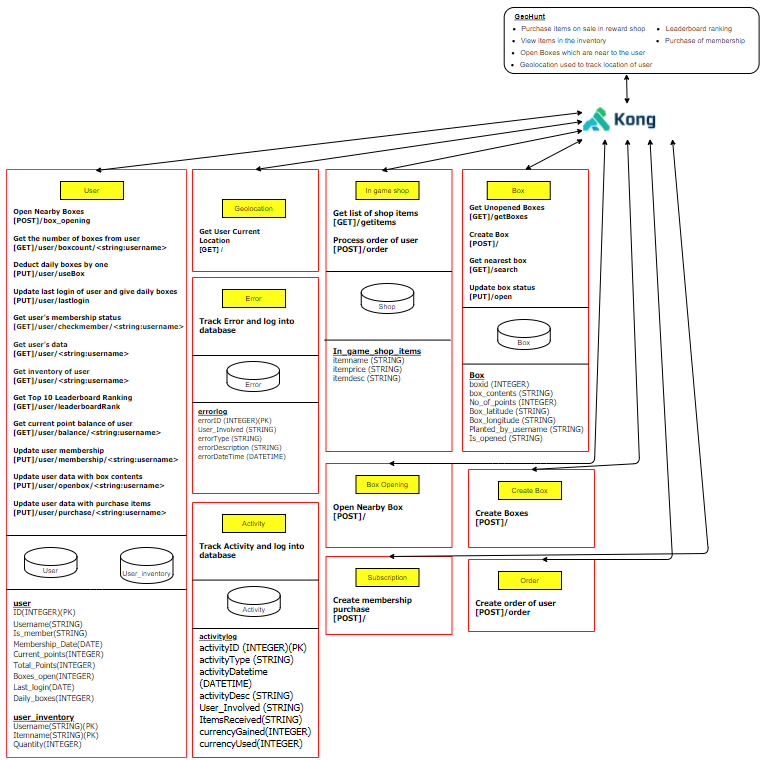
[Subscription API](#_uj0nn5gahfxd) 23

[User API](#_80yn384s3rqh) 23

# **Introduction**

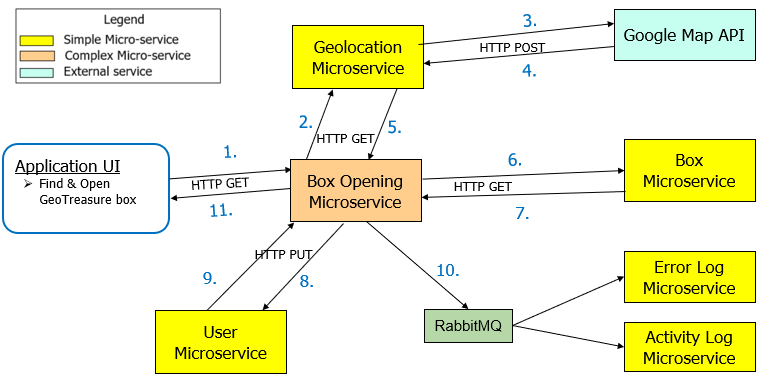
Ever since the lockdown, people have been reluctant to travel out of their homes. There has also been a decline in the variety of attractions and places that are open to the public, resulting in even more reasons to stay at home. To incorporate both fun and exploration, we propose GeoHunt, an application which aims to promote outdoor travel to places that users have never been to before. This community driven initiative will allow users to plant GeoBoxes at their favourite locations for others to find. By travelling to the location where GeoBoxes are placed, users can gain points to spend in the GeoHunt in-game store, and also discover new locations. GeoHunt application also had additional features such as premium membership subscription that comes with several perks including double points, etc.

# **Technical Overview Diagram**

****

# **User Scenarios**

## **[User Scenario 1]: User goes to location with virtual box and opens it.**



1. When the user spots a nearby box, he can click on the “Dig Treasure” button which will invoke the **Box Opening** microservice**.**
2. Get user's current location**.**
3. Create the user's current location coordinates {latitude, longitude}.
4. Returns the user's current location.
5. Upon receiving the user's current location, returns the user’s location details.
6. Check if the location of the virtual box is nearby the user's location using location (latitude, longitude) index matching.
7. If a nearby box is found, return nearest box details. Otherwise, an error message will be returned.
8. Update user’s inventory and game points**.**
9. Returns user update status (successful/ failed).
10. If the user's balance and inventory is updated successfully, this box opening activity is sent as an activity log via **RabbitMQ**. If unsuccessful, the activity will instead be sent as an error log. Depending on the routing key each message has, the message will then be redirected to the respective microservice.
11. If successful, returns content of the nearby box and game points earned to be displayed. Otherwise, an error will be alerted.

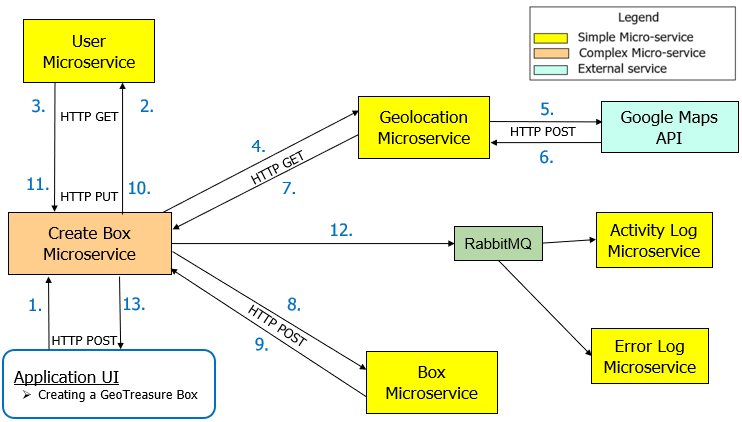
### **(Micro)Services**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **Operational information (e.g., HTTP URL or AMQP exchange type and keys, if any)** | **Description of the functionality** | **Input (if any)** | **Output (if any)** |
| **Geolocation** | [GET] / | Get user’s current location | {username} | {username, latitude, longitude} |
| **Box** | [GET]  /getBoxes | Get a list of boxes that are unopened | - | {list of unopened boxes} where a box is {boxid, box\_contents, no\_of\_points, box\_latitude, box\_longitude, planted\_by\_username, is\_opened} |
| [GET]  /search | Get list of nearby boxes that are within 50m from user’s current location inputted | {latitude, longitude} | {list of nearby boxes} where a box is {boxid, box\_contents, no\_of\_points, box\_latitude, box\_longitude, planted\_by\_username, is\_opened} |
| [PUT]  /open | Update “is\_opened” to 'Y' after box opened | {boxid} | {status of box opening} |
| **Box Opening** (complex) | [POST] / | Invoke geolocation microservice to get user’s current location, invoke box microservice to get nearest box, and then invoke user microservice to update user’s inventory and game points on successful box opening | {username} | {status of box opening} |
| **User** | [GET]  /user/{username} | Get user’s details (check whether username exists) | {username} | {id, username, password, is\_member, membership\_date, current\_points, total\_points, boxes\_open} |
| [PUT] /user/openbox/{username} | Update user’s inventory and game points for box opening | { box\_contents, box\_latitude, box\_longitude, boxid, is\_opened, no\_of\_points, planted\_by\_username } | {status of user’s inventory and game points update after box opening is successful} |
| **Error Log** | Topic exchange with  [BKEY] \*.error | Log error from Box Opening microservice using [BKEY] \*.error | {errorID, user\_Involved, errorType, errorDescription, errorDatetime} | - |
| **Activity Log** | Topic exchange with  [BKEY] \*.activity | Log activity from Box Opening microservice using [BKEY] \*. activity | {id, activityDatetime, activityDesc, user\_Involved, ItemsReceived, currencyGained, currencyUsed} | - |

### **Beyond the labs**

1. **Business logic error-handling**: If the user is the one who planted the box and attempts to open the box, he will not be able to open his planted box and an error message will be alerted. This is done by having an if-else statement which checks whether current user is the same as the user who planted that particular box by running a database check.
2. **Business logic error-handling**: If the user is trying to open a box that is not planted nearby his current location, he will not be able to open the box and an error message will be alerted. This is done by having an if-else statement which checks whether the user’s current location is nearby the box location using location (latitude, longitude) index mapping; if the condition is not met, an error message will be thrown.
3. **Google Map**: External Google Map Geolocation API is used to retrieve user’s current location; it will be used to check whether the location of the virtual box which the user attempts to open is nearby his location.

## **[User Scenario 2]: Users plant a virtual treasure box at their location.**



1. When the user wants to create a box at his current location, the user can click on the “Plant a box” button to invoke the **Create Box** microservice**.**
2. Get the number of daily boxes the user has left to plant**.**
3. Returns the numbers of daily boxes the user is left to plant.
4. If there are remaining boxes left to plant, get the user's current location**.**
5. Create the user's current location coordinates {latitude, longitude}**.**
6. Returns the user's current location.
7. Upon receiving the user's current location, returns the user’s location details.
8. Create a box at the user's current location**.**
9. Returns the status of box creation (successful/ failed).
10. Update the remaining number of daily boxes left to plant.
11. Returns whether the update is successful.
12. If successful, this box creation activity is sent as an activity log via **RabbitMQ**. If unsuccessful, the activity will instead be sent as an error log. Depending on the routing key each message has, the message will then be redirected to the respective microservice.
13. Returns box creation status, which displays an alert for successful/ failed box creation.

### **(Micro)Services**

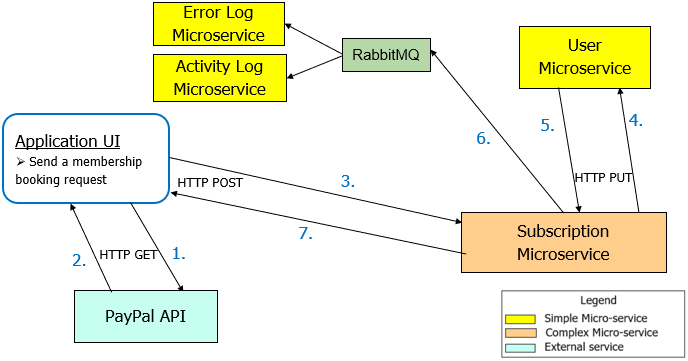
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **Operational information** | **Description of the functionality** | **Input (if any)** | **Output (if any)** |
| **Create box** (complex) | [POST] / | Invoke geolocation microservice to retrieve user’s current location, and invoke Box microservice to create a box based on user’s current location | {username, latitude, longitude} | {status of box creation} |
| **User** | [GET] /user/boxcount/{username} | Get the daily box count remaining for user | {username} | {daily\_boxes} |
| [PUT] /user/useBox | Update the daily box count remaining for the user, and return whether deduction of boxes is successful. | {username} | {status of update} |
| **Geolocation** | [GET] / | Get user’s current location | {username} | {username, latitude, longitude} |
| **Box** | [POST] / | Create a box | {username, latitude, longitude} | {status of box creation} |
| **Error Log** | Topic exchange with  [BKEY] \*.error | Log error from Create Box microservice using [BKEY] \*.error | {errorID, user\_Involved, errorType, errorDescription, errorDatetime} | - |
| **Activity Log** | Topic exchange with  [BKEY] \*.activity | Log activity from Create Box microservice using [BKEY] \*. activity | { id, activityDatetime, activityDesc, user\_Involved, ItemsReceived, currencyGained, currencyUsed} | - |

### Beyond the Labs

1. **Google Map API**: External Google Map Geolocation API will be used to retrieve latitude and longitude of the user’s current location, which will then be set as the location of the user's planted box during our box creation process.

## 

## **[User Scenario 3]: Users purchase premium membership with real-life money.**



1. To apply for membership, the user can click on the “Buy membership” on the navigation bar to invoke **PayPal API** to make payment**.**
2. Returns payment status (successful/ failed).
3. Update membership status for the user.
4. If payment status is successful, update membership status and date for the user.
5. Returns membership update status (successful/ failed).
6. If successful, this membership booking activity is sent as an activity log via **RabbitMQ**. If unsuccessful, the activity will instead be sent as an error log. Depending on the routing key each message has, the message will then be redirected to the respective microservice.
7. Returns the membership booking status, and redirects the user to success or error page depending on membership booking status.

### **(Micro)Services**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **Operational information** | **Description of the functionality** | **Input (if any)** | **Output (if any)** |
| **Subscription**  (complex) | [POST] / | Invoking User microservice to update membership | {username} | {status of membership update} |
| **User** | [GET] /user/{username} | Get user’s details (check whether user exists) | {username} | {id, username, password, is\_member, membership\_date, current\_points, total\_points, boxes\_open} |
| [GET] /user/checkmember/{username} | Get user’s membership status | {username} | {username, is\_member} |
| [PUT] /user/membership/ {username} | Update user’s membership status and date | {username, membership-date} | {status of membership update} |
| **Error Log** | Topic exchange with  [BKEY] \*.error | Log error from Subscriptionmicroservice using [BKEY] \*.error | {errorID, user\_Involved, errorType, errorDescription, errorDatetime} | - |
| **Activity Log** | Topic exchange with  [BKEY] \*.activity | Log activity from Subscription microservice using [BKEY] \*. activity | {id, activityDatetime, activityDesc, user\_Involved, ItemsReceived, currencyGained, currencyUsed} | - |

### **Beyond the labs**

1. **Business logic error-handling**: If the user is an existing member, the “Buy membership” button will be deactivated on his navigation bar, meaning he will not be able to click on the button to access the membership booking payment page; this prevents existing members to sign up for membership more than once. This is done by running a database check to verify whether the current user is a member.
2. **PayPal API:** External API PayPal is used to provide a payment gateway for our users to pay for their membership bookings. PayPal API is invoked by the Subscription microservice to allow users to make payment for their membership booking using PayPal UI page. If the payment is successful, **Subscription** microservice will then invoke **User** microservice to update user’s membership status, and user will then be redirected to payment confirmation page stating their transaction is successful and membership status is updated.

## **[User Scenario 4]: User purchase incentives from our in-game shop using virtual currency (refer to *User Scenario 4* in appendix)**

1. User navigates to the in-game currency store and selects the item he wishes to purchase. After clicking the “Purchase” button, the order information will be sent to the **Order** microservice**.**
2. Create an order request based on the items added to the user's cart.
3. Returns the order request.
4. If the user has sufficient in-game currency, proceed to update the user's balance and inventory**.**
5. Returns successful purchase or an error message.
6. If successful, this purchase activity is sent as an activity log via **RabbitMQ**. If unsuccessful, the activity will instead be sent as an error log. Depending on the routing key each message has, the message will then be redirected to the respective microservice.
7. Returns the user purchase status (successful/ failed), which is being displayed to the user.

### **Beyond the labs**

1. **Business logic error-handling**: If the user does not have sufficient points to purchase items, an error will be thrown to suggest removing some items from the cart. This is done by running a function to calculate the total points required for the purchase and conducting a database check to verify that the user’s points are sufficient for purchase.
2. **Business logic error-handling**: An error will be thrown when the user attempts to click “Purchase” while his cart is empty.

## 

## **[User Scenario 5]: Users view the leaderboard (refer to *User Scenario 5* in appendix)**

1. User clicks the “Leaderboard” button on the navigation bar to invoke the **User** microservice, to get the list of top 10 users (based on the number of boxes opened and the total points earned respectively).
2. Returns the top 10 users in each category, and displays them on the Leaderboard.

# **Remaining Beyond the Labs not covered above**

Use **Kong** for Orchestration Microservice API Gateway as Kong provides a flexible abstraction layer that securely manages communication between clients and microservices such as load balancing, security, monitoring and versioning. As Geohunt is a game application, is it bound to have iteration and updates in the future which means having an api gateway makes the services more loosely coupled. (refer to Architecture diagram in appendix)

# 

# 

# 

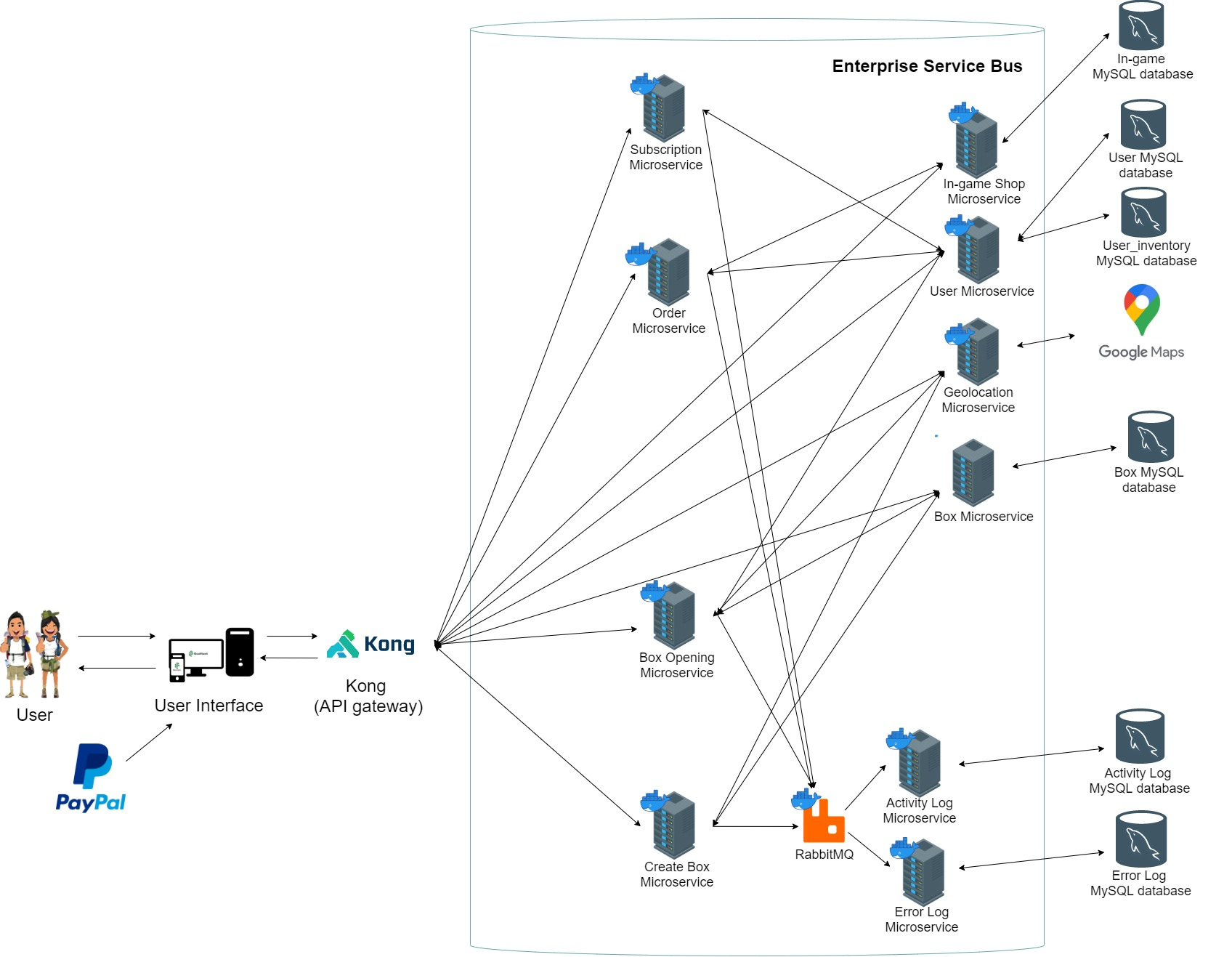
# 

# 

# 

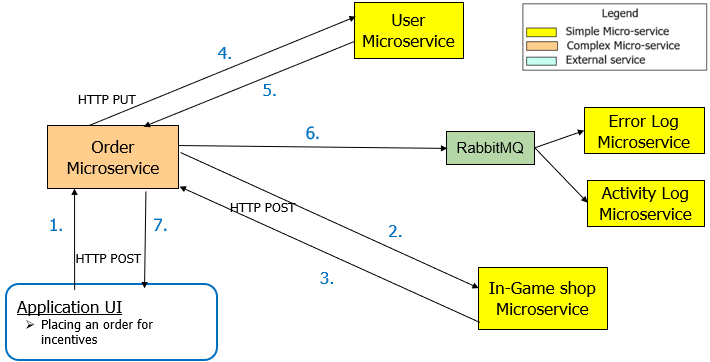
# **Appendix**

# **Architecture Diagram**



# **Extra User Scenario Diagrams**

## **[User Scenario 4]: User purchase incentives from our in-game shop using virtual currency.**

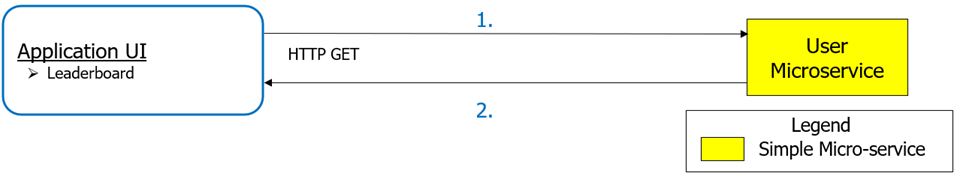
****

### **(Micro)Services**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **Operational information** | **Description of the functionality** | **Input (if any)** | **Output (if any)** |
| **Order**  (complex) | [POST] /order | Invoke In-game shop microservice to create order, and User microservice to update purchase | { username, data: [ {item, quantity}, {item, quantity}, ... ]} | {status of item purchase} |
| **User** | [PUT]  /user/purchase/{username} | Update user’s purchase items into inventory, and game point balance | {data:[ {item,price,quantity}, {item,price,quantity}, ...] } | {status of item purchase} |
| [GET] /user/{username} | Get user’s details (check whether user exists) | {username} | {id, username, password, is\_member, membership\_date, current\_points, total\_points, boxes\_open} |
| **In-game shop** | [GET] /getitems | Get the list of shop items for display | - | {[{item,price, description}, { item, price, description }, ..]} |
| [POST] /order | Create an order for items presented in user’s cart | {data: [{item,quantity}, {item,quantity}, ..]} | {[{itemname, price, quantity}, {itemname, price, quantity}, ..] } |
| **Error Log** | Topic exchange with  [BKEY] \*.error | Log error from Ordermicroservice using [BKEY] \*.error | {errorID, user\_Involved, errorType, errorDescription, errorDatetime} | - |
| **Activity Log** | Topic exchange with  [BKEY] \*.activity | Log activity from Ordermicroservice using [BKEY] \*. activity | {id, activityDatetime, activityDesc, user\_Involved, ItemsReceived, currencyGained, currencyUsed} | - |

### 

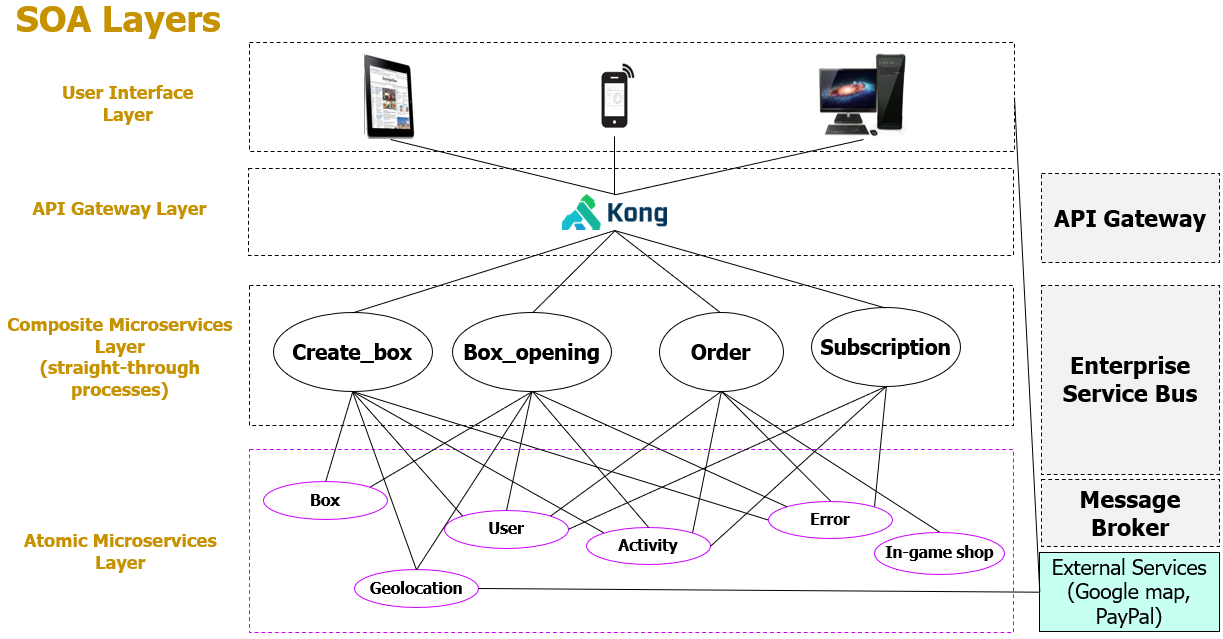
## **[User Scenario 5]: Users view the leaderboard.**



### **(Micro)Services**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **Operational information** | **Description of the functionality** | **Input (if any)** | **Output (if any)** |
| **User** | [GET]  /user/leaderboardRank | Get top 10 users, based on number of boxes opened and total points respectively | - | {top10\_points: [ {username1, total\_points},{username2, total\_points}, ...],  top10\_boxes: [ {username1, boxes\_open},{username2, boxes\_open}, ...]} |

# **SOA Diagram**



# **AMQP Communication Patterns Diagram**

# 

# **API Documentation**

## **Activity API**

|  |  |
| --- | --- |
| Functionality - Create an activity log and insert into activity database | |
| Exchange Name | order\_topic |
| Exchange Type | topic |
| Queue Name | **Activity\_log** |
| Binding Key | **\*.activity** |
| Body of incoming message | Successful activity as a JSON string  Example 1: User Plants a Box  {  'user': 'Michelle',  'action': 'Box Creation',  'success': 'Box created successfully at coordinates > lat: 1.3727946 lng:  103.94921649999999'  }  Example 2: User Opens a Box  {  'user': 'Meghan',  'action': 'Box Opening',  'success': "Updated user's balance and inventory successfully",  'items\_received': '$3 Li-Ho Voucher',  'currencyGained': 116  }  Example 3: User buys item from in-game shop  {  'user': 'Michelle',  'action': 'Purchase from Shop',  'success': 'Purchase successful',  'currencyUsed': 64.0,  'items\_received': '$1 Popular Voucher(1)'  }  Example 4: User purchases membership  {  'user': 'Torrence',  'action': 'Subscription Payment',  'success': 'Membership has been successfully applied'  } |
| Response | No Response as it is AMQP (Fire-and-Forget) |

## **Error API**

|  |  |
| --- | --- |
| Functionality – Create an error record and insert into error log database | |
| Exchange Name | order\_topic |
| Exchange Type | topic |
| Queue Name | **Error** |
| Binding Key | **\*.error** |
| Body of incoming message | The failed actions done on the GeoHunt app.  Example 1: User tries to open their own box.  {  'user': 'Michelle',  'action': 'Box Opening',  'error': 'Box found but cannot be open as it is a self-planted box.'  }  Example 2: User tries to search for a box, but there are no boxes found.  {  'user': 'Sarah Baumert',  'action': 'Box Opening',  'error': 'There are no nearby boxes for you to open'  }  Example 3: User tries to buy from in-game shop but do not have enough points  {  'user': 'Michelle',  'action': 'Purchase from Shop',  'error': 'You do not have enough points to purchase everything in your cart!'  } |
| Response | No Response as it is AMQP (Fire-and-Forget) |

## **Box API**

|  |  |
| --- | --- |
| GET **/getBoxes** – Get all boxes that are not opened yet | |
| Parameters | No parameters |
| Responses | Content type – application/json |
| Code – 200 | Description – successful retrieval  Example Value  {  "boxes": [  {  "box\_contents": "$30 GrabVoucher",  "box\_latitude": "1.2345",  "box\_longitude": "1.5345",  "boxid": 1,  "is\_opened": "N",  "no\_of\_points": 54,  "planted\_by\_username": "Michelle"  },  {  "box\_contents": null,  "box\_latitude": "1.4567",  "box\_longitude": "1.8924",  "boxid": 2,  "is\_opened": "N",  "no\_of\_points": 203,  "planted\_by\_username": "Michelle"  },  {  "box\_contents": "$3 Li-Ho Voucher",  "box\_latitude": "0.9343",  "box\_longitude": "0.82354",  "boxid": 3,  "is\_opened": "N",  "no\_of\_points": 100,  "planted\_by\_username": "Michelle"  },  {  "box\_contents": null,  "box\_latitude": "1.456",  "box\_longitude": "1.4555",  "boxid": 4,  "is\_opened": "N",  "no\_of\_points": 123,  "planted\_by\_username": "Meghan"  },  {  "box\_contents": "$5 Capitaland Voucher",  "box\_latitude": "1.2983886999999998",  "box\_longitude": "103.85641989999999",  "boxid": 20,  "is\_opened": "N",  "no\_of\_points": 254,  "planted\_by\_username": "Michelle"  }  ],  "code": 200  } |
| Code - 404 | Description - No unopened boxes found  Example Value  {  "code": 404,  "message": "There are no boxes on the map at all!"  } |
|  |  |
| POST **/** – Create a box | |
| Parameters | No parameters |
| Body | Username and Geolocation coordinates  Example Value  {  "username":"Michelle",  "latitude":1.245343,  "longitude":0.9724245  } |
| Responses | Content type – application/json |
| Code - 201 | Description - Successful box deploying  Example Value  {  "code": 201,  "message": "Your box has been deployed successfully"  } |
| Code - 500 | Description - Error with box creation  Example Value  {  "code": 500,  "message": "An error occurred while creating the order.  (mysql.connector.errors.InterfaceError) 2003: Can't connect to MySQL  server on 'host.docker.internal:3306' (111 Connection  refused)\n(Background on this error at: http://sqlalche.me/e/rvf5)"  } |
|  |  |
| GET **/search** – Searches for an unopened box | |
| Parameters | latitude, longitude |
| Responses | Content type – application/json |
| Code - 200 | Description - Search finds an unopened box  Example Value  {  "code": 200,  "result": {  "box\_contents": "$3 Li-Ho Voucher",  "box\_latitude": "1.245343",  "box\_longitude": "0.9724245",  "boxid": 23,  "is\_opened": "N",  "no\_of\_points": 205,  "planted\_by\_username": "Michelle"  }  } |
| Code - 404 | Description - No box nearby  Example Value  {  "code": 404,  "message": "There is no box nearby."  } |
| Code - 500 | Description - Error occurred when searching for box  Example Value  {  "code": 500,  "message": "An error occurred while searching for a box: <error>"  } |
|  |  |
| PUT **/open** – Update unopened box status to OPENED | |
| Parameters | No parameters |
| Body | Box Id  Example Value  {  "boxid": 2  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Box status updated successfully  {  "code": 200,  "message": "Box status updated to open successfully."  } |
| Code - 500 | Description - Error occurred while updating box status  {  "code": 500,  "message":"An error occurred while updating box status: <Error> "  } |

## **Box Opening API**

|  |  |
| --- | --- |
| POST **/** – Opens a Box at user’s location and updates user’s points / inventory | |
| Parameters | No parameters |
| Body | Username  Example Value  {  "user”: “Meghan”  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Box successfully opened and update user’s points and inventory with box contents  Example Value  {  "code": 200,  "rewards": {  "points": 240,  "prize": "$10 GrabVoucher"  }  } |
| Code - 404 | Description - No nearby boxes for user to open  Example Value  {  "code": 404,  "message": "There are no nearby boxes for you to open."  } |
| Code - 500 | Description - User found their own box / Other internal errors  Example Value 1  {  "code": 500,  "message": "You found your box! Unfortunately, you cant open your own  box."  }  Example Value 2  {  "code":500,  "message":"The site encountered an internal error: " + <Error>  } |

## **Create Box API**

|  |  |
| --- | --- |
| POST **/** – Checks if user has enough boxes, then creates a box | |
| Parameters | No parameters |
| Body | Description - Username  Example Value  {  "user”: “Meghan”  } |
| Responses | Content type – application/json |
| Code - 201 | Description - Box has been successfully created and deployed  Example Value  {  "code": 201,  "message": "Your box has been deployed successfully"  } |
| Code - 404 | Description - User has no boxes to deploy  Example Value  {  "code": 404,  "message": "You have no boxes to plant."  } |
| Code - 500 | Description - Other internal errors  Example Value  {  "code": 500,  "message": "Error occurred while checking how many boxes user has."  } |

## **Geolocation API**

|  |  |
| --- | --- |
| GET **/** – Get user’s current location | |
| Parameters | No parameters |
| External API | Google Maps |
| Responses | Content type – application/json |
| Code - 200 | Description - Successful retrieval of user’s location.  Example Value  {  "code": 200,  "result": {  "accuracy": 2266,  "location": {  "lat": 1.3727946,  "lng": 103.94921649999999  }  }  } |
| Code - 500 | Description - Other internal errors  {  "code": 500,  "result": "An error occurred while obtaining user's geolocation: " + str(e)  } |

## **Ingame Shop API**

|  |  |
| --- | --- |
| GET **/getitems** – Get all items from the shop database | |
| Parameters | No parameters |
| Responses | Content type – application/json |
| Code - 200 | Description - Retrieve items from the shop database  Example Value  {  "code": 200,  "shop\_items": [  {  "description": "Redeem to get a $1 Popular Voucher.",  "itemname": "$1 Popular Voucher",  "price": 80  },  {  "description": "Redeem this to get a $10 Voucher that you can use on  Grab.",  "itemname": "$10 GrabVoucher",  "price": 350  },  {  "description": "Redeem this to get a $20 Gongcha Voucher.",  "itemname": "$20 Gongcha Voucher",  "price": 550  },  {  "description": "Redeem to get a $3 Li\_Ho Voucher.",  "itemname": "$3 Li-Ho Voucher",  "price": 130  },  {  "description": "Redeem this to get a $5 Capitaland Voucher.",  "itemname": "$5 Capitaland Voucher",  "price": 220  }  ]  } |
| Code - 500 | Description - Other internal errors  Example Value  {  "code": 500,  "message": "An error occurred while retrieving shop items: <error>”  } |
|  |  |
| POST **/order** – Create an order of user’s purchase | |
| Parameters | No parameters |
| Body | Description - Cart Data containing items that user selected  Example Value  {  "data": [  {  "item": "$20 Gongcha Voucher",  "quantity": 3  },  {  "item": "$3 Li-Ho Voucher",  "quantity": 10  }  ]  } |
| Responses | Content type – application/json |
| Code - 201 | Description - Order successfully created  Example Value  {  "code": 201,  "data": [  {  "itemname": "$20 Gongcha Voucher",  "price": 550,  "quantity": 3  },  {  "itemname": "$3 Li-Ho Voucher",  "price": 130,  "quantity": 10  }  ]  } |
| Code - 500 | Description - Other Internal Errors  Example Value  {  "code": 500,  "message": "An error occurred while creating order: <error>"  } |

## **Order API**

|  |  |
| --- | --- |
| POST **/order** – Completes user’s shop purchase and update user’s inventory | |
| Parameters | No parameters |
| Body | Description - Username and Cart Data containing items that user selected  Example Value  {  "username": "Sarah Baumert",  "data": [  {  "item": "$20 Gongcha Voucher",  "quantity": 1  },  {  "item": "$5 Capitaland Voucher",  "quantity": 1  }  ]  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Purchase successful  Example Value  {  "code": 201,  "currencyUsed": 770,  "items\_received": "$20 Gongcha Voucher(1),$5 Capitaland Voucher(1)",  "message": "Purchase is successful. Your items have been added to your  inventory."  } |
| Code - 404 | Description - User’s cart is empty  Example Value  {  "code": 404,  "message": "Your cart is empty, Please select at least 1 item before you  checkout."  } |
| Code - 500 | Description - Not enough points to purchase  Example Value  {  "code": 500,  "message": "You do not have enough points to purchase everything in your  cart!"  } |

## **Subscription API**

|  |  |
| --- | --- |
| POST **/** – Updates user’s membership status after purchase | |
| Parameters | No parameters |
| Body | Description - Status code of membership purchase and username  Example Value  {  "code": 200,  "username": "Chrissa"  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Member transaction is successful and updates user’s membership  Example Value  {  "code": 200,  "data": {  "boxes\_open": 0,  "current\_points": 0,  "daily\_boxes": 3,  "is\_member": "Y",  "last\_login": null,  "membership\_date": "Sun, 11 Apr 2021 00:00:00 GMT",  "total\_points": 2,  "username": "Chrissa"  },  "message": "Membership has been successfully applied"  } |
| Code - 500 | Description - User is already an existing member  Example Value  {  "code": 500,  "message": "An error occurred: user is a existing subscribed member"  } |

## **User API**

|  |  |
| --- | --- |
| GET **/user/boxcount/<username>** – Find number of boxes the user has. | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose box count you want to return  Example Value  Michelle |
| Responses | Content type – application/json |
| Code – 200 | Description – successful retrieval  Example Value  {  "code": 200,  "data": {  "daily\_boxes": 5,  "username": "Michelle"  }  } |
| Code – 404 | Description – User not found  Example Value  {  "code": 404,  "message": "User is not found"  } |
|  |  |
| PUT **/user/useBox** – Deducts used box from user’s number of boxes | |
| Parameters | No parameters |
| Body | Username  Example Value  {  "username": "Yuri"  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Deduction successful  Example Value  {  "code": 200,  "message": "Deduction of box is successful"  } |
| Code - 404 | Description - User not found  Example Value  {  "code": 404,  "message": "User not found."  } |
| Code - 500 | Description - User has zero boxes  Example Value  {  "code": 500,  "message": "You have zero boxes."  } |
|  |  |
| PUT **/user/lastlogin** – Records user’s last login date and replenishes boxes | |
| Parameters | No parameters |
| Body | Username  Example Value  {  "username": "Yuri"  } |
| Responses | Content type – application/json |
| Code - 200 | Description - User logged in for the first time today  Example Value  {  "code": 200,  "message": "Welcome back, you have received 3 boxes. Happy planting!"  }  Description - User has already logged in today  Example Value  {  "code": 200,  "message": "User logged in today already."  } |
| Code - 404 | Description - User not found  Example Value  {  "code": 404,  "message": "User not found."  } |
|  |  |
| GET **/user/checkmember/<username>** – Checks membership status of user | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose membership you want to check  Example Value  Michelle |
| Responses | Content type – application/json |
| Code – 200 | Description – successful retrieval of user’s membership  Example Value  {  "code": 200,  "user": {  "is\_member": "Y",  "username": "Michelle"  }  } |
| Code - 404 | Description - User not found  Example Value  {  "code": 404,  "message": "User not found."  } |
|  |  |
| GET **/user/<username>** – Gets user’s details | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose details you want to return  Example Value  Michelle |
| Responses | Content type – application/json |
| Code – 200 | Description – Successful retrieval of user’s details  Example Value  {  "code": 200,  "user": {  "boxes\_open": 6,  "current\_points": 54,  "daily\_boxes": 5,  "is\_member": "Y",  "last\_login": "Mon, 12 Apr 2021 00:00:00 GMT",  "membership\_date": "Tue, 06 Apr 2021 00:00:00 GMT",  "total\_points": 6015,  "username": "Michelle"  }  } |
| Code - 404 | Description - User not found  Example Value  {  "code": 404,  "user": "Not found."  } |
|  |  |
| GET **/user/getInventory/<username>** – Retrieves user’s inventory | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose inventory you want to return  Example Value  Michelle |
| Responses | Content type – application/json |
| Code – 200 | Description – Successful retrieval of user’s inventory  Example Value 1  {  "code": 200,  "inventory": [  {  "itemname": "$1 Popular Voucher",  "quantity": 8  },  {  "itemname": "$10 GrabVoucher",  "quantity": 8  },  {  "itemname": "$20 Gongcha Voucher",  "quantity": 6  },  {  "itemname": "$3 Li-Ho Voucher",  "quantity": 3  }  ]  }  Example Value 2  {  "code": 200,  "inventory": []  } |
|  |  |
| GET **/user/leaderboardRank** – Retrieves Top 10 users on the Leaderboards | |
| Parameters | No parameters |
| Responses | Content type – application/json |
| Code – 200 | Description – Successful retrieval  Example Value  {  "code": 200,  "top10\_boxes": [  {  "boxes\_open": 12,  "username": "Torrence"  },  {  "boxes\_open": 11,  "username": "Sarah Baumert"  },  {  "boxes\_open": 8,  "username": "Yuri"  },  {  "boxes\_open": 6,  "username": "Michelle"  },  {  "boxes\_open": 6,  "username": "Meghan"  },  {  "boxes\_open": 5,  "username": "Lunatone"  },  {  "boxes\_open": 4,  "username": "Mieka"  },  {  "boxes\_open": 4,  "username": "Alison"  },  {  "boxes\_open": 3,  "username": "Monique"  },  {  "boxes\_open": 0,  "username": "Christine"  }  ],  "top10\_points": [  {  "total\_points": 6015,  "username": "Michelle"  },  {  "total\_points": 5323,  "username": "Torrence"  },  {  "total\_points": 2056,  "username": "Sarah Baumert"  },  {  "total\_points": 1530,  "username": "Yuri"  },  {  "total\_points": 1243,  "username": "Meghan"  },  {  "total\_points": 502,  "username": "Alison"  },  {  "total\_points": 499,  "username": "Mieka"  },  {  "total\_points": 112,  "username": "Lunatone"  },  {  "total\_points": 55,  "username": "Christine"  },  {  "total\_points": 52,  "username": "Monique"  }  ]  } |
| Code - 404 | Description - No records found  Example Value  {  "code": 404,  "inventory": “there are no records found”  } |
|  |  |
| GET **/user/balance/<username>** – Retrieves user’s current point balance | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose current point balance you want to check  Example Value  Michelle |
| Responses | Content type – application/json |
| Code – 200 | Description – Successful retrieval of user’s current point balance  Example Value  {  "code": 200,  "data": {  "current\_points": 54,  "username": "Michelle"  }  } |
| Code - 404 | Description – User does not exist  Example Value  {  "code": 404,  "message": "There is no such user existing"  } |
|  |  |
| PUT **/user/membership/<username>** – Updates user’s membership status | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username whose membership status you want to update  Example Value  Michelle |
| Body | Username and membership-date  Example Value  {  "username": "Michelle",  "membership-date": "2021-04-11"  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Membership update successful  Example Value  {  "code": 200,  "data": {  "boxes\_open": 8,  "current\_points": 1253,  "daily\_boxes": 3,  "is\_member": "Y",  "last\_login": "Mon, 12 Apr 2021 00:00:00 GMT",  "membership\_date": "Sun, 11 Apr 2021 00:00:00 GMT",  "total\_points": 1530,  "username": "Yuri"  },  "message": "Membership has been successfully applied"  } |
| Code - 403 | Description - Parameter username and body username does not match  Example Value  {  "code": 403,  "message": "Usernames do NOT match!"  } |
| Code - 404 | Description - Username does not exist  {  "code": 404,  "data": {  "username": "Sarah"  },  "message": "username not found"  } |
| Code - 500 | Description - User is already a subscribed member  Example Value  {  "code": 500,  "data": {  "boxes\_open": 6,  "current\_points": 54,  "daily\_boxes": 5,  "is\_member": "Y",  "last\_login": "Mon, 12 Apr 2021 00:00:00 GMT",  "membership\_date": "Tue, 06 Apr 2021 00:00:00 GMT",  "total\_points": 6015,  "username": "Michelle"  },  "message": "user is a existing subscribed member"  } |
|  |  |
| PUT **/user/openbox/<username>** – Update user’s points and inventory with box contents | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username of the person who opened the box  Example Value  Yuri |
| Body | Details of box that is to be opened  Example Value  {  "box\_contents": "$30 GrabVoucher",  "box\_latitude": "1.2345",  "box\_longitude": "1.5345",  "boxid": 1,  "is\_opened": "N",  "no\_of\_points": 54,  "planted\_by\_username": "Michelle"  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Box successfully opened and user data updated.  Example Value  {  "code": 200,  "data": {  "item\_won": "$30 GrabVoucher",  "points\_earned": 54  }  } |
| Code - 404 | Description - User not found  Example Value  {  "code": 404,  "data": {  "username": "Alisons"  },  "message": "user not found"  } |
| Code - 500 | Description - Other internal errors  Example Value  {  "code": 500,  "message": "an error occurred updating user inventory: <error>"  } |
|  |  |
| PUT **/user/purchase/<username>** – Updates user point balance and inventory with purchase details | |
| Parameters |  |
| Name | Description |
| username  string  *(path)* | Username of the person who opened the box  Example Value  Alison |
| Body | Details of purchase  Example Value  {  "data": [  {  "itemname": "$1 Popular Voucher",  "price": 80,  "quantity": 2  },  {  "itemname": "$3 Li-Ho Voucher",  "price": 130,  "quantity": 1  }  ]  } |
| Responses | Content type – application/json |
| Code - 200 | Description - Successful updating of purchase  Example Value  {  "code": 200,  "currencyUsed": 290,  "items\_received": "$1 Popular Voucher(2),$3 Li-Ho Voucher(1)",  "message": "Purchase is successful. Your items have been added to your  inventory."  } |
| Code - 404 | Description - No items selected / username not found  Example Value 1  {  "code": 404,  "message": "Your cart is empty, Please select at least 1 item before you  checkout."  }  Example Value 2  {  "code": 404,  "data": {  "username": "Alisons"  },  "message": "user not found"  } |
| Code - 500 | Description - User does not have enough points / other internal errors  Example Value 1  {  "code": 500,  "message": "You do not have enough points to purchase everything in your  cart!"  }  Example Value 2  {  "code": 500,  "message": "an error occurred updating user inventory: <error>"  } |