CAS 735
(Micro)service-Oriented Archetectures
Fall 2022

Project Report

Hong Li, Shuiliang Wu, Fanping Jiang
McMaster University

Git Repo Here

Contents

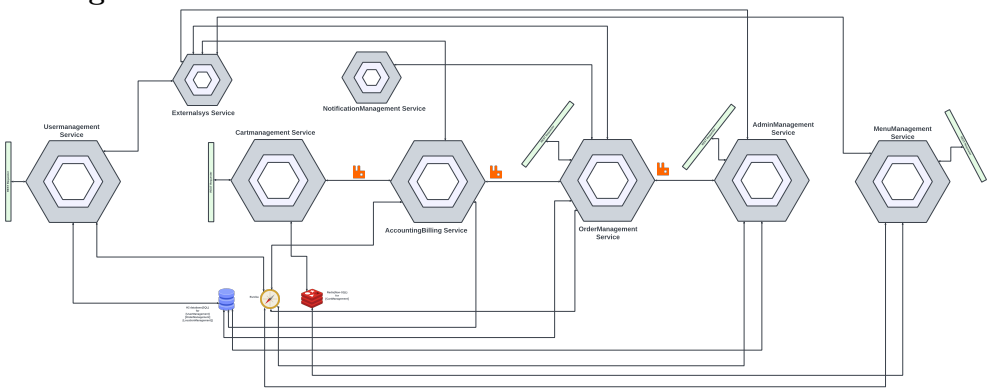
1 Description of Microservice Architecture

1.1 Birdview of Services and Connections

1.1.1 General

Hexagonal Architecture Diagram Below is the thumbnail of the hexagonal service diagram in bird view, the original bird view file is here **Hexagonal Service Diagram in Bird View**.

The detailed version of hexagonal service diagram is here **Hexagonal Service Diagram in Detail**



Brief Introduction for Services There are 8 services and 4 infrastructures.

1. **User management**

   In *User management* service, clients are able to register and get Email of role creation.

2. **External**

   As for *External service*, Email function and ETF are wrapped in it.

3. **Notification**

   This service is to notify users through mobile apps when they get new messages from the other services.

4. **Cart management**

   Since book stores and food providers have similar cart except the products, the *Cart management service* can manage carts for them. The events include get information from a cart, add/delete items and check out.

   After an check-out event, the *Accounting & Billing service* will be called.

5. **Accounting & Billing**

   When receiving a check-out event from *Cart management service*, the service will forward the payment request to *external service*, write the purchase information into repository and call *order management service* to deal with the order.

6. **Order management**

   This service checks and modify the status of orders, update user information, send notification and arrange bikers accordingly.

7. **Admin**

   This service is for the administrator to exercise administrator's responsibilities.

8. **Menu**

   This service is to read and write menus from the repository.

1.1.2 Dataflow for System

First step of all users (including Administrators), is to register an account through *User management service*.

Community Member A community member Avery first browses the menu from *Menu management service*, she may add more items or delete an item from the cart by *Cart management service*.

When she decides to check out and click the button, the order will be forwarded to *Accounting & Billing service*. Now she can make the payment though the ETF by *External service*.

If she has multiple purchase from different food providers (book stores), different orders will be generated separated by food providers (book stores) at back end; Meanwhile, only one total invoice will be ready for Avery to make the payment.

If Avery successfully makes the payment, the *Order management service* will allow her to select a pick-up time and location. If it is too busy at a drop-off location, the food provider can help Avery to relocate and reschedule the delivery by a *Modify location & time event*. After delivery, an notification will be sent by *Notification service*. Also the MDD points will be rewarded automatically at the end of every week.

Food Provider A food provider Kendall can accept and refuse orders through *Order management service*. As for pick-up, Kendall can start a *Modify location & time event* to select pick-up time. The notification will be sent automatically by *Notification service* to bikers.

Once the food is delivered and accepted, the community member can receive a notification. The daily transactions for food will be approved by administrator in *Admin Management Service*.

Biker As a biker, Peyton accesses pending deliveries by phone apps. Peyton will by notified when there is a new food order waitng for pickup.

Also, Peyton can notify the pick-up *Pick-up deliver food order event* and drop-off *Drop-off deliver food order event* to community members.

Bookstore Operator As a bookstore operator, Charlie's dataflow is similar to Peyton, but without delivery-pickup events; also products in bookstore can be purchased by MDD points.

When Avery checks out at a book store, *Order management service* will put the order information into the repository. By *Ask for book store MDD compensation event*, Avery can request the total MDD compensation invoice to McMaster every week.

McMaster's Administrators Morgan, an administrator, he accesses the repository to get information about drop-off locations' workload. Also, the total costs for bikers and bookstore MDD compensation can be calculated through the *Admin Management Service*. Finally, the MDD authentification can be triggered by a *MDD compensation controller*.

1.2 Justification of Choices

1.2.1 Practical & Technically Feasible Design

In this section, we justify the architecture and individual service design from a practical and technically feasible perspective.

1. General: Why we design 8 services

   Before we start, we decide that at least 2 services, one service for user registration (will have deletion and account modification in the future), one for administration. Because these 2 services have very limited correspondence with other services.

   Then we are considering about the business requirements. To purchase from different food providers, we need one cart to gather all items and several orders, one order per food provider. Because food providers and book stores are similar in the purchase process, (i.e. they both have menus for items, they both have carts to checkout and multiple orders for different providers), they can share some services. So we design cart management service, accounting and billing service and order management service.

   Beyond that, our business involves notification, email and ETF payment. So we design external service to implement email and ETF payment, meanwhile, we also design notification service.

   Finally we add a services for menu management, to make it easier for community members to browse and stores to maintain.

Therefore we design 8 services. We will definitely have more if keep building the system, but for now we have 8.

2. *User management*: only talks to H2 repo and external service.

   We find that, for other services, they only need to check the user information when necessary rather than frequently read/write the user information. So user management service only talks to H2 repo and external service.

   This design reduces the coupling and increases the cohesion.

3. *Cart management*: checking menu directly from repo

   Similar to *user management*, *cart management* get menu information from the Redis repository directly, while the menu repo is maintained by a stand alone service *menu management*.

   Then the *cart management service* does not need to take care of the menu any more. This design reduces the size of *cart management* and make the service more robust.

   Also notice that the menu content is frequently changed, a No-SQL repository will have better performance than traditional database. Meanwhile, we want store both product ids and descriptions, a lite-weight No-SQL *Redis* is the best choice.

4. *Accouting & billing*: external service and concurrency

   To deal with a cart checkout, *accounting & billing service* call external *ETF service* and email the community member about the payment invoice.

   After the ETF payment, we can forward the order to *order management service* and processing it. This design can help increase the order processing efficiency and help community members get food earlier.

5. *Order management*: *RabitMQ* for notifications, *Eureka* for *Email*

   In the *order management service*, both *Eureka* and *RabitMQ* are used for communication. However, since external service is a standalone service, it is achieved by *Eureka*.

   Because the pending delivery list is maintained by a message queue in the app, we need to dynamically modify the queue, we use *RabitMQ* to send the delivery information to bikers and the estimated pick-up time to community members. This is also maintained by the front-end app.

6. *Admin management*: MDD approval

   We have 2 requests for communication about MDD compensation, one is through Email to notify administrator, the other one through *RabitMQ* from the repository to send the compensation application

to admin management service. The compensation application service message needs to be verified by an administrator for consistency, and then the MDD compensation can be approved.

To apply 2 ways of communication of MDD compensation between order management service and admin management service, we can minimize the confusing and make it clear and direct for the administrator to verify.

1.2.2 Business Requirement Coverage

The architecture and the individual service are desgined to cover and fulfill all business requirements. We justify that our architecture and design cover the following business requirements.

Community Member

1. Community member registration.

   This is achieved by the *Community member controller* in *User management service*.

2. Community members can order from different providers seam-lessly and pay a single invoice at the end of the ordering process.

   The *order management service* allows community members generate multiple orders in one cart and a total check-out invoice.

3. At the end of the ordering process, community member can select one of the drop-off location available on campus, and an expected delivery time window.

   The *Select Location Time Food Order Controller* helps members select the drop-off location.

4. If the delivery slot is too busy, they will be provided an alternate pick-up location, or to select a less busy delivery window at the same location.

   The alternative location and delivery time are provided by food providers through *Modify Location Time Food Order Controller*.

5. When the order is delivered, community member receives a notification.

   The notification is sent by the mobile app after calling *Notification Management Controller*.

6. *Unloyalty program* & MDD accumulation during the given week

   The MDD calculation with *Unloyalty program* adjustment is implemented in *order management service*.

Food Provider

1. Food provider registration.

   Finished by *User management service*, similar to community members.

2. Food providers can declare their menu

   This is implemented by *Menu management service*.

3. Food provider can refuse the food order and notify community member.

   Rejection is triggered by *Refuse Food Order Controller* and notification is sent automatically by *Notification service*.

4. Food provider can accept the food order and notify the biking centre with an estimated pick-up time.

   Food orders are accepted by *Accept Food Order Controller* and notifications are sent automatically similar to the *Refuse Controller*. Also notice that the pick-up time is calculated by the processor.

Biker

1. Biker registration

   Similar to previous ones.

2. Biker can access on their phone to a list of pending orders that needs to be delivered.

   This is achieved by *Get Waiting For Pick Up Food Orders Controller* and will reflect on the app. Biker can get a list of food orders waiting for pickup.

3. Biker can update order status when pick up the order and notify community member.

   With *Pick Up Deliver Food Order Controller*, the biker notifies the community member that he/she has picked the food.

4. Biker can update order status when drop off the order and notify community member and food provider

   With *Drop Off Deliver Food Order Controller*, the biker notifies both the community member and the food provider that the food is ready.

Bookstore Operator

1. Bookstore operator registration.

   Similar to the previous ones.

2. Bookstore operator can offer a selection of goodies/merch(declare bookstore menu)

   This is implemented by *Menu management service*.

3. Bookstore operator can cash MDDs from member's wallet and offer a selection of goodies/merch.

   This is similar to the food purchase process, the only difference is that the currency is in MDD and the payment is not external but implemented by *Cart management service, Accounting & Billing Service and Order management service* itself.

4. At the end of each week, Bookstore operator can synchronize the list of goods sold with MDD money, and automatically send an invoice to McMaster for compensation.

   Operators can find *Ask For Book Store Mdd Compensation Controller* in *order management* to help generate the goods' list and send invoice to the administrator. Also, the purchase records from the repository will be checked and sent concurrently.

McMaster's Administrators

1. Administrator operator registration

   Similar to the previous registration.

2. Administrator can monitor the usage of the different drop-off locations

   Administrator monitors the workload for each location by */admin/locations/{id},get* controller. The result will be get from the location repository.

3. Administrator can monitor the workload associated to bikers

   Similar to the previous one, except *locations* substituted by *bikers*.

4. The results of the platform (in terms of safety) need to be balanced by its costs (number of hired bikers, MDD reward program goodies) and presented each couple of months to the Board of Governors

   Costs can be obtained through */admin/financial/report/fromTimetotoTime,get* controller in *Admin* service. The time length can be configured freely.

5. At the end of the day, Administrator can approve the transactions during the day and food provider receives a single Interac transfer from McMaster for all the orders delivered during the day, as well as a detailed billing statement summarizing all transaction of the day

   Administrator can get the summary of transactions at */admin/dailytransaction/foodorder/billingstatement/{id}/{date}* for each food provider and send an Interac transfer each by */admin/dailytransaction/foodorder/approval/{id}/{date}*.

2 Description of Interfaces

Interfaces (Business Services) makes the "middle layers" (named ports in this project) which separate outer controller layers (named adapters in this project) from inner business logic layer (named business in this project). It decouples method calls from their code implementation.

2.1 User Management Service

User Management Service provides functions of registration of MacDrop users (ie., Administrator, Biker, Book Store Operator, Community Member and Food Provider) and calls Externalsys Service to send a notification email to the registered user once the registration is completed.

Table 1: List of Interfaces in User Management Service

Interfaces	Methods
AdminRegistrationService	RoleCreationResponse createAdmin(Admin admin);
AdminRepository	/
BikerRegistrationService	RoleCreationResponse createBiker(Biker biker);
BikerRepository	/
BookStoreOperatorRegistration	RoleCreationResponse createBookStoreOperator(Bookstoreoperator bookStoreOperator);
BookStoreOperatorRepository	/
CommunityMemberRegistration	RoleCreationResponse createCommunityMember(Communitymember communitymember);
CommunityMemberRepository	/
FoodProviderRegistration	RoleCreationResponse createFoodProvider(Foodprovider foodprovider);
FoodProviderRepository	/
EmailService	void send(EmailRequest req);

Description The main interfaces in User Management Service are five *RegistrationService* and *EmailService*, together with their corresponding Repository interfaces to add MacDrop user data into its corresponding H2 database. Details on interfaces can be seen in *Table 1.*

Use Administrator as an example, the REST client sends a POST request to */admin*. The body of the request contains the JSON data for an administrator. The REST service will convert JSON data to POJO and add it to admin in-memory H2 database through *AdminRepository* interface. Meanwhile, an *EmailRequest* is built and sent to *EmailClient* through *EmailService* interface which has a *send* method. The response to the client is a Data Transfer Object (DTO) *RoleCreationResponse* which contains a status code and a response. For example, if the registration is processed successfully on Administrator, it will response an status code of 200 and an message "Created administrator successfully!".

The other types of users have a similar description on their interfaces thus their descriptions are not duplicated here.

2.2 Cart Management Service

Cart Management Service is the start point of the whole data flow of the essential food ordering service provided by MacDrop. This service provides functions of performing CRUD operations on the data of food carts and bookstore carts that are stored in Redis database. Once carts are checked out, the data of food carts and bookstore carts is then sent to Accounting Billing Service.

Description The main interfaces in Cart Management Service are *BookStoreCartService*, *FoodCartService* and *AccountingBillingService*, together with their corresponding Repository interfaces to perform CRUD operations on data stored in H2 database. We will use *get*, *addToFoodCart*, *removeFromFoodCart* and *checkoutFoodCart* methods to describe *FoodCartService* and *AccountingBillingService* interfaces for food carts. *BookStoreCartService* interface is similar with *FoodCartService*, thus its description is not duplicated here. Details on interfaces can be seen in *Table 2*.

FoodCart get(String id) To review the items in the Food Cart, the REST client sends a GET request to */foodcart/management/*{*communitymemberid*} with *communitymemberid* as a PathVariable. The request is mapped to *FoodCartController* which calls *get* method in *FoodCartService* interface. *FoodCartService* delegates the call to *FoodCartRepository* interface at the business logic layer and returns *FoodCart*. It is then converted into JSON and sent back to the client.

CartResponse addToFoodCart(String id, FoodCartItem item) When adding an item into food cart, the REST client sends a POST request to */foodcart/management/*{*communitymemberid*} with *communitymemberid* as a PathVariable. The body of the request contains the JSON data of a selected food item. The REST service will convert JSON data to POJO and add it to food cart Redis database through *FoodCartRepository* interface.

Table 2: List of Interfaces in Cart Management Service

Interfaces	Methods
BookStoreCartService	1. BookStoreCart get(String id); 2. CartResponse addToBookStoreCart(String id, BookStoreCartItem item); 3. CartResponse removeFromBookStoreCart(String id, String bookStoreItemId); 4. CartResponse checkoutBookStoreCart(String id);
BookStoreCartRepository	1. String getKey(BookStoreCart bookStoreCart); 2. String getKey(String id); 3. void addToCart(String bookStoreCartKey, Path cartItemsPath, BookStoreCartItem item); 4. void removeFromCart(String bookStoreCartKey, Class¡BookStoreCartItem¿ bookStoreCartItemClass, Path cartItemsPath, Long index);
FoodCartService	1. FoodCart get(String id); 2. CartResponse addToFoodCart(String id, FoodCartItem item); 3. CartResponse removeFromFoodCart(String id, String foodId); 4. CartResponse checkoutFoodCart(String id);
FoodCartRepository	1. String getKey(FoodCart foodCart); 2. String getKey(String id); 3. void addToCart(String foodCartKey, Path cartItemsPath, FoodCartItem item); 4. void removeFromCart(String foodCartKey, Class¡FoodCartItem¿ foodCartItemClass, Path cartItemsPath, Long index);
AccountingBillingService	1. void sendFoodCartToAccountingBilling(FoodCart foodCart); 2. void sendBookStoreCartToAccounting-Billing(BookStoreCart bookStoreCart);

FoodCartRepository interface provides methods to get food cart key using *communitymemberid* and add a selected food item to this food cart through food cart's key and path. The response to the client is a DTO *CartResponse* which contains a status code and a response. For example, if adding a food item is processed successfully, it will response an status code of 200 and an

message "The item has been added successfully!".

CartResponse removeFromFoodCart(String id, String foodId) When removing an item from the food cart, the REST client sends a DELETE request to */foodcart/management/{communitymemberid}* with *communitymemberid* as a PathVariable. The body of the request contains the food Id. The request is mapped to *FoodCartController* which calls the *removeFromFoodCart* method in *FoodCartService* interface. *FoodCartService* then delegates the call to *FoodCartRepository* interface which has a *emoveFromCart* method to delete the food item from the cart through parameters of *foodCartKey*, *foodCartItemClass*, *cartItemsPath* and *index*. The response to the client is a DTO *CartResponse* which contains a status code and a response. For example, if a food item is deleted successfully, it will response an status code of 200 and an message "The item was removed successfully".

CartResponse checkoutFoodCart(String id) and void sendFoodCartToAccountingBilling(FoodCart foodCart) Once food items in food cart are determined, the next step is to check it out. When checking out, the REST client sends a POST request to */foodcart/payment/{communitymemberid}* with *communitymemberid* as a PathVariable. The request is mapped to *FoodCartController* which calls *checkoutFoodCart* method in *FoodCartService* interface. *FoodCartService* first delegates the call to *accountingBillingService* interface which has a *sendFoodCartToAccountingBilling* method to send the food cart to *Accounting Billing Service*. *FoodCartService* interface then delegates the call to *FoodCartRepository* interface which has a *removeFromCart* method to empty the cart. The response to the client is a DTO *CartResponse* which contains a status code and a response. For example, if the food cart is checked out successfully, it will response an status code of 200 and an message "Checking out the cart".

2.3 Accounting Billing Service

Once Accounting Billing Service receives data of food cart and bookstore cart from Cart Management Service, it calls Externalsys Service to send an email and an ETF transfer request to the related community members regarding to the billing. It also generates a food order list and a bookstore order list arranged according to food providers/ bookstore and pass them to Order Management Service.

Description The main interfaces in Accounting Billing Service are *BookStoreAccountingBillingService*, *BookStoreOrderManagementService*, *FoodAccountingBillingService*, *FoodOrderManagementService*, *EmailService* and *EtfTransferService*, together with their corresponding Repository interfaces to get data stored in H2 database. We will use *foodCartAccountingBilling*, *createFoodOrder*, *send* and *etfTransfer* methods to describe *FoodAccountingBillingService*, *FoodOrderManagementService* together with *EmailService*

Table 3: List of Interfaces in Accounting Bill Service

Interfaces	Methods
BookStoreAccountingBillingService	void bookStoreCartAccounting-Billing(BookStoreCart bookStoreCart);
BookStoreOrderManagementService	void createBookStoreOrder(BookStoreOrderList bookStoreOrderList);
FoodAccountingBillingService	void foodCartAccountingBilling(FoodCart foodCart);
FoodOrderManagementService	void createFoodOrder(FoodOrderList foodOrderList);
BookStoreOperatorRepository	/
CommunityMemberRepository	/
EmailService	void send(EmailRequest req);
EtfTransferService	void etfTransfer(EtfTransferRequest req);

and *EtfTransferService* interfaces for food order lists. *BookStoreAccounting-BillingService* and *BookStoreOrderManagementService* interfaces are similar to previously mentioned interfaces for food orders, thus their descriptions are not duplicated here. Details on interfaces can be seen in *Table 3*.

void foodCartAccountingBilling(FoodCart foodCart) After receiving the food cart data at *FoodAccountingBillingController* from Cart Management Service, it calls *foodCartAccountingBilling* method in *FoodAccountingBillingService* interface. *FoodAccountingBillingService* delegates the call to *CommunityMemberRepository* interface to find the community member, in order to get member's email address. *FoodAccountingBillingService* interface then delegates the call to *send* method in *EmailService* interface, in order to send email request to Externalsys Service. After that, *FoodAccountingBillingService* interface delegates the call to *etfTransfer* method in *EtfTransferService* interface, which sends the ETF request to Externalsys Service as well. At the end, *FoodAccountingBillingService* interface delegates the calls to *createFoodOrder* method in *FoodOrderManagementService* interface, which passes the food order list to Order Management Service.

2.4 Order Management Service

In Order Management Service, we receive food order lists and bookstore order lists through their corresponding Controllers from Accounting Billing Service. For food orders, this service provides functions to update food order status and notify the updates to the community member, biker and food providers that are involved. It also calculates MDD rewards for community

members. In terms of bookstore orders, it notifies the bookstore operator and the community member when a book store order is created. Besides, it sends MDD transaction lists and the compensation invoices directly and indirectly (through Externalsys Service) to Admin Management Service for compensation approval.

Description The main interfaces in Order Management Service are *BookStoreOrderManagementService*, *FoodOrderManagementService* and *MddCompensationService*, together with their corresponding Repository interfaces to update data stored in H2 database. We will use *acceptFoodOrder* method as a demonstration to describe its related interfaces for food orders and use *askForBookStoreCompensation* method as a demonstration for bookstore orders. The other POST and GET request methods have a similar description on the interfaces thus their descriptions are not duplicated here. Also, *EmailService* and *NotificationService* have been described before thus they won't be duplicated here either. Details on interfaces can be seen in *Table 4*.

OrderManagementResponse acceptFoodOrder(String foodorderId); When accepting a food order, the REST client sends a POST request to */foodorderoperation/foodprovider/acceptance/{foodorderid}* with *foodorderid* as a PathVariable. It then calls *acceptFoodOrder* method in *FoodOrderManagementService* interface. *FoodOrderManagementService* interface delegates the call to *FoodOrderRepository* interface at the business logic layer to find the food order and update the food order status into "accepted waiting for pickup". After that, it calls *notify* method in *NotificationService* interface which is implemented at *NotificationClient* to send notification command for both community member and biker to Notification Service. The response to the client is a DTO *OrderManagementResponse* which contains a status code and a response. For example, if accepting a food item is processed successfully, it will response an status code of 200 and an message "Accepted the order successfully!".

BookStoreOrderManagementResponse askForBookStoreCompensation(String bookStoreOperatorId, TimeRange timeRange); When bookstore asks for a compensation on its MDD collection, the REST client sends a POST request to */bookstoreorderoperation/bookstoreoperator/mddcompensation/{bookstoreoperatorid}* with *bookstoreoperatorid* as a PathVariable. The body of the request contains the JSON data of a time range indicating the start and end time for MDD calculation. The request is mapped to *BookStoreOrderOperationController* which calls the *askForBookStoreCompensation* method in *BookStoreOrderManagementService* interface. *BookStoreOrderManagementService* then delegates the call to *BookStoreOrderRepository* interface which uses *findByOrderTimeRange* method to select the bookstore orders from its database within the given time pe-

Table 4: List of Major Interfaces in Order Management Service

Interfaces	Methods
BookStoreOrderManagementService	1. void createBookStoreOrder(BookStoreOrderList bookStoreOrderList) 2. BookStoreOrderManagementResponse askForBookStoreCompensation(String bookStoreOperatorId, TimeRange timeRange);
FoodOrderManagementService	1. void createFoodOrder(FoodOrderList foodOrderList); 2. OrderManagementResponse acceptFoodOrder(String foodorderId); 3. OrderManagementResponse refuseFoodOrder(String foodorderId, FoodOrderResponseMessage msg); 4. OrderManagementResponse determineLocationTimeCommunityMember(String foodorderId, FoodOrderLocationTimeSelection foodOrderLocationTimeSelection); 5. OrderManagementResponse modifyLocationTimeFoodProvider(String foodorderId, FoodOrderLocationTimeSelection foodOrderLocationTimeSelection); 6. OrderManagementResponse pickUpFoodOrderBiker(String foodorderId, String bikerId); 7. OrderManagementResponse dropOffCompleteFoodOrderBiker(String foodorderId); 8. void mddCalculation(TimeRange timeRange, String rewardRate); 9. String getWaitingForPickUpOrders(String status);
MddCompensationService	void askForMddCompensation(MddCompensation mddCompensation);
NotificationService	void notify(NotificationCommand notification);
EmailService	void send(EmailRequest req);
Six Repositories	...

riod. It then delegates the call to *BookStoreOperatorRepository* interface to find the book store operator in order to form a *compensationMessage*. After

that, it updates the bookstore operator's MDD and delegates the call to *BookStoreOperatorRepository* interface to update the MDD value. At the end, it delegates the call to *EmailService* interface to send the email request to Externalsys Service, meanwhile delegating the call to *MddCompensationService* interface to send the list of goods and invoice to Admin Management Service. The response to the client is a DTO *BookStoreOrderManagementResponse* which contains a status code and a response. For example, if list of goods and invoice are sent successfully, it will response an status code of 200 and an message "Asking for compensation...".

2.5 Admin Management Service

In Admin Management Service, we monitor workloads on bikers and locations, create financial reports regarding to MacDrop running cost and approve daily transactions made through MacDrop at food providers and MDD compensation requests from book stores. In this project, it is considered as the end point of the whole data flow of the essential food ordering service provided by MacDrop.

Description The main interfaces in Admin Management Service are *MonitoringService*, *FinancialService* and *ApprovalsService*, together with their corresponding Repository interfaces to get data stored in H2 database. We will use one method per each main interface as a representative to provide description. Other interfaces like *EmailService* and *EtfTransferService* have been described before thus won't be duplicated here. Details on interfaces can be seen in *Table 5*.

Map ¡Integer, Integer¿ getCurrentBikerLoads(); *MonitoringService* provides service to get current loads for all bikers. The REST client sends a GET request to */admin/bikers/currentworkloads*. The request is mapped to *AdminController* which calls *getCurrentBikerLoads* method in *MonitoringService* interface. *MonitoringService* then delegates the call to *BikerMonitoringRepository* interface in its business logic layer where *BikerMonitoringRepository* uses *findAll* method to get all the bikers in the database. All the biker loads are obtained through a foreach loop and data is stored in the form of HashMap containing biker id and its corresponding workloads. The response to the client is this HashMap.

Map ¡String, BigDecimal¿ getFinancialReport(String fromTime, String toTime); *FinancialService* provides service to get the financial report in a given time period. The REST client sends a GET request to */admin/financial/report/{fromTime}to{toTime}* with both *fromTime* and *toTime* as PathVariables. The request is mapped to *AdminController* which calls *getFinancialReport* method in *FinancialService* interface. *FinancialService* then delegates the call to *FinancialBookStoreOrderRepository* interface

Table 5: List of Major Interfaces in Admin Management Service

Interfaces	Methods
MonitoringService	1. Map¡Integer, Integer¿ getCurrentBikerLoads(); 2. Map¡Integer, Integer¿ getHistoryBikerLoads(); 3. Map¡Integer, Integer¿ getLocationCurrentLoads(); 4. Map¡Integer, Integer¿ getLocationHistoryLoads();
FinancialService	1. BigDecimal getBikerCenterCost(String fromTime, String toTime); 2. BigDecimal getBookStoreCost(String fromTime, String toTime); 3. Map¡String, BigDecimal¿ getFinancialReport(String fromTime, String toTime);
ApprovalsService	1. String getDailyFoodOrderBillStatement(String id, String date); 2. void approveDailyFoodOrderTransactions(String id, String date); 3. void approveWeeklyBookStoreMddCompensationInvoice(MddCompensation mddCompensation);
EmailService	void send(EmailRequest req);
EtfTransferService	void etfTransfer(EtfTransferRequest req);
Five Repositories	...

in its business logic layer where *FinancialBookStoreOrderRepository* uses *findByOrdertimeBetween* method to find all the food orders and bookstore orders during the giving time period from the database. The Biker Center Cost, MDD Reward Program Cost and Cost In Total are stored in a HashMap which is the response to the client.

void approveWeeklyBookStoreMddCompensationInvoice (MddCompensation mddCompensation); Bookstore's weekly MDD Compensation request is received at *MddCompensationController* from Order Management Service. *MddCompensationController* calls *ApproveWeeklyBookStoreMddCompensationInvoice* method in *ApprovalsService* interface which then delegates the call to *FinancialBookStoreOrderRepository*. *Fi-*

nancialBookStoreOrderRepository then reaches to the database to find all the bookstore orders made during the giving time period. Once the total MDD is calculated, an message is formed based on the comparison between total MDD from MDD compensation request and total MDD obtained from the bookstore order database. *Send* method is then call from *EmailService* interface, in order to send email request to Externalsys Service. If both MDD values are matched, *etfTransfer* method is called from *EtfTransferService* interface which is implemented at *EmailClient* to send the ETF request to Externalsys Service.

2.6 Externalsys Service

Externalsys Service takes care of sending email requests and ETF transfer requests to other service once the requests are received.

Table 6: List of Interfaces in Externalsys Service

Interfaces	Methods
EmailService	EmailSent send(SendEmailCommand request);
EtfTransferService	EtfTransferSent etfSend(EtfTransferCommand request);

Description The interfaces here are *EmailService* and *EtfTransferService*. Their functions are similar thus we only use *EmailService* as a demonstration to avoid duplication. Details on interfaces can be seen in *Table 6*.

EmailSent send(SendEmailCommand request); Email sending command is received at *EmailController*. *EmailController* calls *send* in *EmailService* interface which is implemented at its business logic layer. The response to client is a DTO *EmailSent* which includes a code and string response.

2.7 Notification Management Service

Notification Management Service takes care of notifying the clients.

Table 7: List of Interfaces in Notification Management Service

Interfaces	Methods
NotificationService	NotificationSent notify(NotificationCommand notification);

Description *NotificationService* is the only one interface here. Details on interfaces can be seen in *Table 7*.

Notification command is received at *NotificationManagementController*. *NotificationManagementController* calls *notify* method in *NotificationService* interface which is implemented at its business logic layer. The response to client is a DTO *NotificationSent* which includes a code and string response.

2.8 Menu Management Service

Menu Management Service takes care of updating food menu and bookstore menu (for MDD reward program) then sending email requests to Externalsys Service to notify community members regarding to the updates.

Description The main interfaces in Menu Management Service are *BookStoreMenuService* and *FoodMenuService*, together with their corresponding Repository interfaces to get data stored in Redis database. We will use one method from *FoodMenuService* interface as a demonstration on all CRUD operations in this service. We will also describe *notifyNewMenu* method from *FoodMenuService* interface. Other methods in above two interfaces are similar and *EmailService* has been described before thus they won't be duplicated here. Details on interfaces can be seen in *Table 8*.

MenuResponse addToFoodMenu(String id, FoodMenuItem item); *FoodMenuService* provides service to add a new item in food menu. The REST client sends a POST request to */foodmenu/management/*{*foodproviderid*} with *foodproviderid* as a PathVariable. The body of the request contains the JSON data of a new food item. The request is mapped to *FoodMenuController* which calls *addToFoodMenu* method in *FoodMenuService* interface. *FoodMenuService* then delegates the call to *FoodMenuRepository* interface at the business logic layer where *FoodMenuRepository* uses *findById*, *getKey*, *addToMenu* and *save* methods to have it implemented. The response to client is a DTO *MenuResponse* which contains a status code and a string response.

MenuResponse notifyNewMenu(String id, String foodProviderEmail); *FoodMenuService* provides service to notify all community members when there is an update on food menu. The REST client sends a POST request to */foodmenu/management/newmenunotification/*{*foodproviderid*} with *foodproviderid* as a PathVariable. The body of the request contains the JSON data of food provider's Email address. The request is mapped to *FoodMenuController* which calls *notifyNewMenu* method in *FoodMenuService* interface. *FoodMenuService* then delegates the call to *emailService* interface at the business logic layer where *emailService* uses *send* methods to have it implemented. The response to client is a DTO *MenuResponse* which contains a status code and a string response.

Table 8: List of Interfaces in Menu Management Service

Interfaces	Methods
BookStoreMenuService	1. BookStoreMenu get(String id); 2. MenuResponse addToBookStoreMenu(String id, BookStoreMenuItem item); 3. MenuResponse removeFromBookStoreMenu(String id, String itemId); 4. MenuResponse notifyNewMenu(String id, String bookStoreOperatorEmail);
FoodMenuService	1. FoodMenu get(String id); 2. MenuResponse addToFoodMenu(String id, FoodMenuItem item); 3. MenuResponse removeFromFoodMenu(String id, String itemId); 4. MenuResponse notifyNewMenu(String id, String foodProviderEmail);
BookStoreMenuRepository	1. String getKey(BookStoreMenu bookStoreMenu); 2. String getKey(String id); 3. void addToMenu(String bookStoreMenuKey, Path menuItemsPath, BookStoreMenuItem item); 4. void removeFromMenu(String bookStoreMenuKey, Class¡BookStoreMenuItem¿ bookStoreMenuItemClass, Path menuItemsPath, Long index);
FoodMenuRepository	1. String getKey(FoodMenu foodMenu); 2. String getKey(String id); 3. void addToMenu(String foodMenuKey, Path menuItemsPath, FoodMenuItem item); 4. void removeFromMenu(String foodMenuKey, Class¡FoodMenuItem¿ foodMenuItemClass, Path menuItemsPath, Long index);
EmailService	void send(EmailRequest req);

3 Testing

In order to demonstrate the services cover all business requirements and are technically covered, the following test scenarios are provided.

Before testing the system, please ensure the step of deployment is executed.

Please use Postman or Swagger built in the service to execute REST requests.

The link is document of Postman requests used in the test, the order of request keeps consistent with the order of tests. **Postman request document link**

3.1 Deployment

```
1.Please go to the directory of macdrop_group1/deployment in main
    branch.

2.run docker-compose up --force-recreate to deploy the services.

3.wait all services to be initialized for a few minutes. Then visit
    http://localhost:8761/, All services excluding infrastructure
    services can be seen on Eureka.
```

3.2 Test scenarios for community member

Please execute the following steps to demonstrate community members' business requirements are covered.

- Community member registration

```
1.Visit http://localhost:9092/swagger-ui/index.html#/

2.In createCommunityMember of community-member-controller, with
    POST, send the following JSON payload to
    http://localhost:9092/communitymember
{
  "currentmdd": "0",
  "etfemail": "cmusername1@mcmaster.ca",
  "firstname": "cmfirstname1",
  "lastname": "cmlastname1",
  "password": "******",
  "username": "cmusername1"
}

3.The following response is expected
{
  "statusCode": 200,
  "response": "Created community member successfully!"
```

```
}

4.Visit http://localhost:81, type the JDBC URL of
    jdbc:h2:tcp://localhost:1521/macdrop, then connect to H2
    database.

5.Run SQL of SELECT * FROM COMMUNITYMEMBER, a new community member
    whose username is cmusername1 has been created. There are some
    dummy community members created when initialized the service.

6.In the log of externalsys-srv container, the following message
    indicating that the community member creation email has been
    sent to the community member successfully can be seen.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     cmusername1@mcmaster.ca
* - Object: Community Member Registration on MacDrop
* - Message: Community member registered successfully! Your
    username is cmusername1
*** Ending email request ***
}
also, in the log of usermanagement-srv container, the following
    message indicating that the community member creation email has
    been sent to the community member successfully can be seen.
{
** Preparing sending email
** Using instance: http://172.18.0.10:9091
** Response: EmailResponse(smtpStatusCode=250,
    smtpServerResponse=OK)
** Email Sent
}
```

- Community members can order from different providers seamlessly and pay a single invoice at the end of the ordering process

---

```
1.Visit http://localhost:9093/swagger-ui/index.html#/

2.In addToFoodCart of food-cart-controller, with POST and parameter
    of 10000001, send the following JSON payload to
    http://localhost:9093/foodcart/management/10000001
{
  "foodId": "20000001001",
  "foodName": "bread1",
  "foodProviderId": "20000001",
  "price": 12.5,
  "quantity": 4
}
The following response is expected
{
```

```
  "statusCode": 200,
  "response": "The item has been added successfully"
}
```
which means the community member of 10000001 added the food of 20000001001 from the food provider of 20000001. The quantity and price are 4 and 12.5, respectively.

3.In addToFoodCart of food-cart-controller, with POST and parameter of 10000001, send the following JSON payload to http://localhost:9093/foodcart/management/10000001

```
{
  "foodId": "20000002001",
  "foodName": "Taco1",
  "foodProviderId": "20000002",
  "price": 15.5,
  "quantity": 3
}
```
The following response is expected
```
{
  "statusCode": 200,
  "response": "The item has been added successfully"
}
```
which means the community member of 10000001 added the food of 20000002001 from the food provider of 20000002. The quantity and price are 3 and 15.5, respectively.

4.In addToFoodCart of food-cart-controller, with POST and parameter of 10000001, send the following JSON payload to http://localhost:9093/foodcart/management/10000001

```
{
  "foodId": "20000001002",
  "foodName": "bread2",
  "foodProviderId": "20000001",
  "price": 10,
  "quantity": 2
}
```
The following response is expected
```
{
  "statusCode": 200,
  "response": "The item has been added successfully"
}
```
which means the community member of 10000001 added the food of 20000001002 from the food provider of 20000001. The quantity and price are 2 and 10, respectively.

5.In getFoodCart of food-cart-controller, with parameter of 10000001, send GET request to http://localhost:9093/foodcart/management/10000001. The following response is expected.

```
{
  "id": "10000001",
```

```
  "foodCartItems": [
    {
      "foodProviderId": "20000001",
      "foodName": "bread1",
      "foodId": "20000001001",
      "price": 12.5,
      "quantity": 4
    },
    {
      "foodProviderId": "20000002",
      "foodName": "Taco1",
      "foodId": "20000002001",
      "price": 15.5,
      "quantity": 3
    },
    {
      "foodProviderId": "20000001",
      "foodName": "bread2",
      "foodId": "20000001002",
      "price": 10,
      "quantity": 2
    }
  ],
  "total": 116.5
}
```


which shows the food cart of community member 10000001

6.As food cart shows, the community member of 10000001 selects food from two food providers which are 20000001 and 20000002. In checkoutFoodCart of food-cart-controller, with POST and parameter of 10000001, send REST request to http://localhost:9093/foodcart/payment/10000001. The following response is expected.


```
{
  "statusCode": 200,
  "response": "Checking out the cart"
}
```


7.In the log of externalsys-srv, a billing email has been sent to the community member as follow.

```
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     u1@mcmaster.ca
* - Object: MacDrop Food Billing
* - Message:
*********************************************
Hi, Here is your food billing on MacDrop:
Items details:
Items:[bread2(ID20000001002); bread1(ID20000001001); ] From Food
    Provider:[20000001] Price:[ CAD 70.00]
```

```
Items:[Taco1(ID20000002001); ] From Food Provider:[20000002]
    Price:[ CAD 46.50]


Total price: CAD 116.50

Regards,
MacDrop
****************************************

*** Ending email request ***

8.In the log of externalsys-srv, an ETF transfer message as follow
    shows the community member has paid a single invoice though
    ordered from different food providers.
}

9.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, two orders made by community member of 10000001 are
    shown. However, the status of order is location & time needed,
    which means drop-off location and delivering time need to be be
    determined by the community member. The next scenario will show
    how the drop-off location and delivering time are determined by
    the community member.
```

- At the end of the ordering process, community member can select one of the "drop-off" location available on campus, and an expected delivery time window

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, select the foodorderid of the food order made by
    community member in previous scenario.

2.Visit http://localhost:9095/swagger-ui/index.html#/

3.In selectLocationTimeFoodOrder of
    food-order-operation-controller, with POST and foodorderid from
    last scenrario, send the following payload to
    localhost:9095/foodorderoperation/communitymember/locationtimeselection/{foodorderid}
{
  "location": "80000001",
  "time": "2022-10-30 12:59:54"
}
Please note that the time should be the later then order time when
    test it

4.The following response is expected.
{
```

```
  "statusCode": 200,
  "response": "Determined the location & time successfully!"
}

5.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, the food order's DROPOFFLOCATIONID and
    ESTIMATEDPICKUPTIME have changed accordingly.

6. The status of order has changed to placed, which means the order
    has been officially placed.

7.However, it doesn't mean the food order is accepted. After food
    provider reviews the order and approves the drop-off location
    and pickedup time, it will turn to be accepted and the food
    provider will notify bikers to pick up.
```

- If the delivery slot is too busy, they will be provided an alternate pick-up location, or to select a less busy delivery window at the same location

---

```
1.visit http://localhost:9095/swagger-ui/index.html#/

2.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, pick the foodorderid of the food order in previous
    scenario.

3.In the modifyLocationTimeFoodOrder of
    food-order-operation-controller, with POST and foordorderid in
    previous scenario, send the following payload to
    localhost:9095/foodorderoperation/foodprovider/modification/{foodorderid}
{
  "location": "80000002",
  "time": null
}

4.The following response is expected.
{
  "statusCode": 200,
  "response": "Modified the location & time successfully!"
}
which means an alternated drop-off pick-up location has been
    provided by the food provider.

5.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, the DROPOFFLOCATIONID has changed accordingly.

6.In the modifyLocationTimeFoodOrder of
```

```
    food-order-operation-controller, with POST and foordorderid in
    the previous scenario, send the following payload to
    localhost:9095/foodorderoperation/foodprovider/modification/{foodorderid}
{
  "location": null,
  "time": "2022-10-30 14:59:54"
}
Please note that the time should be the later then order time when
    test it

7.The following response is expected.
{
  "statusCode": 200,
  "response": "Modified the location & time successfully!"
}
which means an alternated time has been provided by the food
    provider.

8.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, the ESTIMATEDPICKUPTIME has changed accordingly.

9.In the modifyLocationTimeFoodOrder of
    food-order-operation-controller, with POST and foordorderid in
    previous step, send the following payload to
    localhost:9095/foodorderoperation/foodprovider/modification/{foodorderid}
{
  "location": "80000003",
  "time": "2022-10-30 15:59:54"
}
which means the alternated drop-off location and time are provided.

10.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, the DROPOFFLOCATIONID and ESTIMATEDPICKUPTIME have
    changed accordingly.

11.Please note that, in the above scenario, the alternated location
    and time are provided by food provider.
```

- When the order is delivered, community member receives a notification

---

```
1.This test must be done after the biker picks up and starts to
    deliver the food order, so it will be mentioned in the test
    scenarios for biker
```

- "Unloyalty" program & MDD accumulation during the given week

1.There are some food orders for tests crated when initialized the service.

2.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM FOODORDER, there are three complete orders whose communitymemberid is 10000005. So, with MDD rate of 7%, community member of 10000005 is expected to get 46.2 MDDs.

3.Visit http://localhost:9095/swagger-ui/index.html#/

5.In mddCalculation of food-order-operation-controller, with POST and parameter of 7, send the following payload to localhost:9095/foodorderoperation/mddcalculation/{rewardrate}

```
{
  "leftday": "24",
  "lefthour": "00",
  "leftmin": "00",
  "leftmonth": "10",
  "leftsec": "00",
  "leftyear": "2022",
  "rightday": "30",
  "righthour": "23",
  "rightmin": "59",
  "rightmonth": "10",
  "rightsec": "59",
  "rightyear": "2022"
}
```

6.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop.
Run SELECT * FROM COMMUNITYMEMBER, the CURRENTMDD of community member whose id is 10000005 has changed from 1000 to 1046.2, which means with MDD rate 7%, The expected MDD has been calculated and added correctly to community member of 10000005.

7.In the above scenario, the community member of 10000005 ordered from three food provider and expected to receive MDD. In the following scenario, the example that community member of 10000006 only ordered in one food provider and shouldn't receive MDD will be demonstrated.

8.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM FOODORDER, there is one only food order made by community member of 10000006.

9.In mddCalculation of food-order-operation-controller, with POST and parameter of 7, send the following payload to localhost:9095/foodorderoperation/mddcalculation/{rewardrate}

```
{
```

```
  "leftday": "17",
  "lefthour": "00",
  "leftmin": "00",
  "leftmonth": "10",
  "leftsec": "00",
  "leftyear": "2022",
  "rightday": "23",
  "righthour": "23",
  "rightmin": "59",
  "rightmonth": "10",
  "rightsec": "59",
  "rightyear": "2022"
}

10.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    COMMUNITYMEMBER, actually the CURRENTMDD of the community
    member of 10000006 is 1000 which doesn't change. It
    demonstrates that the community member will not receive the MDD
    if only ordered food from one provider.
```

3.3 Test scenarios for food provider

- Food provider registration

```
1.Visit http://localhost:9092/swagger-ui/index.html#/

2.In createFoodProvider of food-provider-controller, with POST,
    send the following payload to localhost:9092/foodprovider
{
  "etfemail": "fp1username1@mcmaster.ca",
  "firstname": "fpfirstname1",
  "lastname": "fplastname1",
  "password": "******",
  "username": "fp1username1"
}

3.The following response is expected.
{
  "statusCode": 200,
  "response": "Created food provider successfully!"
}

4.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODPROVIDER, a food provider whose username is fp1username1 is
    shown.

5.In the log of externalsys-srv, the following message is shown to
```

```
    demonstrate that the food provider creation message has been
    sent.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     fp1username1@mcmaster.ca
* - Object: Food Provider Registration on MacDrop
* - Message: Food provider registered successfully! Your username
    is fp1username1
*** Ending email request ***
}

6.In the log of usermanagement-srv, the following message is shown
    to demonstrate that the food provider creation message has been
    sent.
{
** Preparing sending email
** Using instance: http://172.18.0.10:9091
** Response: EmailResponse(smtpStatusCode=250,
    smtpServerResponse=OK)
** Email Sent
}
```

- Food provider can refuse the food order and notify community member

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, select the foodorderid of the food order made by
    community member in previous scenario. Please note that the
    selected order's status must be placed so that the food
    provider can determine whether to refuse it or not.

2.Visit http://localhost:9095/swagger-ui/index.html#/

3.In refuseFoodOrder of food-order-operation-controller, with POST
    and parameter of foodorderid, send the following payload to
    localhost:9095/foodorderoperation/foodprovider/refusion/{foodorderid}.
{
  "message": "sorry, we don't have this food now"
}

4.The following response is expected.
{
  "statusCode": 200,
  "response": "Refused the order successfully!"
}

5.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
```

FOODORDER, the status of the food order in the previous step has changed to refused, which demonstrates that the food provider can refuse the food order.

6.In the log of notificationmanagement-srv, the following message can be seen to demonstrate that the community member has been notified that the food order has been refused.

```
{
*** Starting notification ***
* - from:   OrderManagementService
* - to:     10000001
* - Message: We feel Sorry that your food order has been refused,
    the reason is sorry, we don't have this food now
*** Ending notification ***
}
```

7.We already demonstrate that the food provider can refuse food order and notify the community member in the previous steps. In order to test other scenarios, we will restore the selected order's status from refused to placed. please run UPDATE FOODORDER SET STATUS= 'placed' WHERE FOODORDERID = ; to restoache selected food order's status. The FOODORDERID should be replaced with the actual foodorderid used in the test.

- Food providers can declare their menu

1.Visit http://localhost:9098/swagger-ui/index.html#/

2.In addToFoodMenu of food-menu-controller, with POST and parameter of food provider id, send the following payload to http://localhost:9098/foodmenu/management/20000001 to add item to food menu.

```
{
    "itemDescription": "This is sandwich",
    "itemId": "20000001005",
    "itemName": "Sandwich-5",
    "price": 3.99
}
```

3.The following response is expected,

```
{
  "statusCode": 200,
  "response": "The item has been added successfully"
}
```

4.In getFoodMenu of food-menu-controller, with GET and parameter of food provider id, send request to http://localhost:9098/foodmenu/management/20000001

5.The following response is expected,

```
{
```

```
  "id": "20000001",
  "foodMenuItems": [
    {
      "itemName": "Sandwich-5",
      "itemDescription": "This is sandwich",
      "itemId": "20000001005",
      "price": 3.99
    }
  ]
}
which demonstrates the item has been added successfully.

6.In notifyNewFoodMenu of food-menu-controller, with POST,
    parameter of food provider and request body of food provider's
    email, send request to
    localhost:9098/foodmenu/management/newmenunotification/20000001
    to notify community members that food provider of 20000001 has
    updated menu.

7.In the log of externalsys-srv, the following response is expected
    to demonstrate that the food menus update email has been sent.
{
*** Processing email request ***
* - from:   20000001@mcmaster.ca
* - to:     allcommunitymembers@mcmaster.ca
* - Object: New Food Menu From 20000001
* - Message:
Hey there!

20000001 updated food menu, you are welcome to take a look

Here is the updated menu:
FoodMenuItem(ItemId=20000001005, ItemName=Sandwich-5,



Regards
*** Ending email request ***
}
```

- Food provider can accept the food order and notify the biking centre with an estimated pick-up time

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, pick the foodorderid of the food order made by
    community member in previous scenario. Please note that the
    selected order's status must be placed so that the food
    provider can accept it.
```

```
2.Visit http://localhost:9095/swagger-ui/index.html#/

3.In acceptFoodOrder of food-order-operation-controller, with POST
    and parameter of foodorderid selected in the previous step,
    send request to
    localhost:9095/foodorderoperation/foodprovider/acceptance/{foodorderid}.

4.The following response is expected
{
  "statusCode": 200,
  "response": "Accepted the order successfully!"
}

5.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, the status of the food order in the previous step
    has been changed to accepted & waiting for pickup. Meanwhile,
    the food order comes with ESTIMATEDPICKUPTIME. It demonstrates
    the food order has been accepted by the food provider and is
    waiting for being picked up by the biker.

6.In the log of notificationmanagement-srv, the following message
    can be seen to demonstrate that the bikers are notified there
    is a new food order waited to be picked up.
{
*** Starting notification ***
* - from:   OrderManagementService
* - to:     All bikers
* - Message: Hi bikers, the order 1 is ready to be picked up!
*** Ending notification ***
}
```

3.4 Test scenarios for biker

- Biker registration

```
1.Visit http://localhost:9092/swagger-ui/index.html#/

2.In createBiker of biker-controller, with POST, send the following
    payload to http://localhost:9092/biker
{
  "currentworkload": 0,
  "etfemail": "bikerusername1@mcmaster.ca",
  "firstname": "bikerfirstname1",
  "historyworkload": 0,
  "lastname": "bikerlastname1",
  "password": "******",
  "username": "bikerusername1"
}
```

```
3.The following response is expected.
{
  "statusCode": 200,
  "response": "Created biker successfully!"
}

4.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    BIKER, a biker whose username is bikerusername1 is shown.

5.In the log of externalsys-srv, the following message is shown to
    demonstrate that the biker creation message has been sent.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     bikerusername1@mcmaster.ca
* - Object: Biker Registration on MacDrop
* - Message: Biker registered successfully! Your username is
    bikerusername1
*** Ending email request ***
}

6.In the log of usermanagement-srv, the following message is shown
    to demonstrate that the biker creation message has been sent.
{
** Preparing sending email
** Using instance: http://172.18.0.10:9091
** Response: EmailResponse(smtpStatusCode=250,
    smtpServerResponse=OK)
** Email Sent
}
```

- Biker can access on their phone to a list of pending orders that needs to be delivered

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, there are some food orders created for test when
    initialized the service.

2.Visit getWaitingForPickUpOrders of
    food-order-operation-controller, with GET, send request to
    localhost:9095/foodorderoperation/biker/waitingforpickuporders

3.A list of food orders waiting for pickup is returned. The
    following is an example.
{
The following is a list of food orders waiting for pickup
Foodorder(foodorderid=7, communitymemberid=10000008,
    foodproviderid=20000002, fooditems=food1(ID20000002001);
```

```
    food2(ID20000002002);, foodorderprice=40.00,
    ordertime=2022-10-11 13:01:54, bikerid=30000001,
    dropofflocationid=80000001, estimatedpickuptime=2022-10-11
    14:01:54, status=accepted & waiting for pickup)
Foodorder(foodorderid=8, communitymemberid=10000008,
    foodproviderid=20000003, fooditems=food1(ID20000003001);
    food2(ID20000003002);, foodorderprice=50.00,
    ordertime=2022-10-11 13:01:54, bikerid=30000001,
    dropofflocationid=80000001, estimatedpickuptime=2022-10-11
    14:01:54, status=accepted & waiting for pickup)
Foodorder(foodorderid=9, communitymemberid=10000008,
    foodproviderid=20000004, fooditems=food1(ID20000004001);
    food2(ID20000004002);, foodorderprice=60.00,
    ordertime=2022-10-11 13:01:54, bikerid=30000001,
    dropofflocationid=80000001, estimatedpickuptime=2022-10-11
    14:01:54, status=accepted & waiting for pickup)
}

4.It demonstrates that biker can access to a list of pending orders
    that needs to be delivered.
```

- Biker can update order status when pick up the order and notify community member

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, pick the foodorderid of the food order made by
    community member in previous scenario. Please note that the
    selected order's status must be accepted & waiting for pickup
    so that the biker can pick up the order.

2.Visit
    http://localhost:9095/swagger-ui/index.html#/food-order-operation-controller

3.In pickUpDeliverFoodOrder of food-order-operation-controller,
    with POST and parameter of selected foodorderid, send the
    following payload to
    localhost:9095/foodorderoperation/biker/delivery/{foodorderid}
{
30000002
}

4.The following reponse is expected
{
  "statusCode": 200,
  "response": "Picked up the order successfully!"
}

5.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
```

FOODORDER, the status of the selected food order has changed to delivering.

6.In the log of notificationmanagement-srv, the following message is shown to demonstrate that the community member has been notified that the food order is picked up and being delivered.

```
{
*** Starting notification ***
* - from:   OrderManagementService
* - to:     10000001
* - Message: Hi, your order 1 is being delivered by 30000002
*** Ending notification ***
}
```

7.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM BIKER, the CURRENTWORKLOAD and HISTORYWORKLOAD of biker of 30000002 have changed to 1, which means the biker of 30000002 picked up the food order and the workload of associated biker has been changed accordingly.

8.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM LOCATION, the CURRENTWORKLOAD and HISTORYWORKLOAD of location of 80000002 have changed to 1, which means, after biker pickup up the food, the workload at associated location has changed accordingly.

- Biker can update order status when drop off the order and notify community member and food provider

---

1.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM FOODORDER, select the foodorderid of the food order made by community member in previous scenario. Please note that the selected order's status must be delivering so that the biker can drop off the order.

2.Visit http://localhost:9095/swagger-ui/index.html#/food-order-operation-controller

3.In dropOffCompleteFoodOrder of food-order-operation-controller, with POST and parameter of selected foodorderid, send the request to localhost:9095/foodorderoperation/biker/dropoff/{foodorderid}.

4.The following response is expected,

```
{
  "statusCode": 200,
  "response": "Dropped off the order successfully!"
```

```
}

6.In the log of notificationmanagement-srv, the following message
    is shown to demonstrate that the community member has been
    notified that the food order is dropped off.
{
*** Starting notification ***
* - from:   OrderManagementService
* - to:     10000001
* - Message: Hi, your order 1 has been dropped off, please pick up
    at 80000002
*** Ending notification ***
}

7.In the log of notificationmanagement-srv, the following message
    is shown to demonstrate that the food provider has been
    notified that the food order is dropped off.
{
*** Starting notification ***
* - from:   OrderManagementService
* - to:     20000001
* - Message: Hi, the food order 1 has been dropped off
*** Ending notification ***
}

8.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    BIKER, the CURRENTWORKLOAD and HISTORYWORKLOAD of biker of
    30000002 have changed to 0 fand 1, respectively, which means
    the biker of 30000002 dropped off the food order and the
    workload of associated biker has been changed accordingly.

9.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    LOCATION, the CURRENTWORKLOAD and HISTORYWORKLOAD of location
    of 80000002 have changed to 0 and 1, respectively, which means,
    after biker dropped off the food, the workload at associated
    location has changed accordingly.
```

3.5 Test scenarios for bookstore operator

- Bookstore operator registration

```
1.Visit http://localhost:9092/swagger-ui/index.html#/

2.In createBookStoreOperator of book-store-operator-controller,
    with POST, send the following payload to
    http://localhost:9092/bookstoreoperator
{
```

```
  "currentmdd": "0",
  "etfemail": "bsousername1@mcmaster.ca",
  "firstname": "bsofirstname1",
  "lastname": "bsolastname1",
  "password": "******",
  "username": "bsousername1"
}

3.The following response is expected.
{
  "statusCode": 200,
  "response": "Created bookstoreoperator successfully!"
}

4.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    BOOKSTOREOPERATOR, a bookstore operator whose username is
    bsousername1 is shown.

5.In the log of externalsys-srv, the following message is shown to
    demonstrate that the bookstore operator creation message has
    been sent.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     bsousername1@mcmaster.ca
* - Object: Book Store Operator Registration on MacDrop
* - Message: Book store operator registered successfully! Your
    username is bsousername1
*** Ending email request ***
}

6.In the log of usermanagement-srv, the following message is shown
    to demonstrate that the bookstore operator creation message has
    been sent.
{
** Preparing sending email
** Using instance: http://172.18.0.10:9091
** Response: EmailResponse(smtpStatusCode=250,
    smtpServerResponse=OK)
** Email Sent
}
```

- Bookstore operator can offer a selection of goodies/merch(declare bookstore menu)

```
1.Vist http://localhost:9098/swagger-ui/index.html#/

2.In the addToBookStoreMenu of book-store-menu-controller, with
    POST and parameter of bookstore operator's id, send the
    following payload to
```

```
    http://localhost:9098/bookstoremenu/management/40000001
{
    "itemDescription": "This is Java textbook",
    "itemId": "40000001002",
    "itemName": "Java textbook-2",
    "mddPrice": 5.0
}

3.The following reponse is expected
{
  "statusCode": 200,
  "response": "The item has been added successfully"
}

4.In the getBookStoreMenu of book-store-menu-controller, with GET
    and parameter of bookstore operator's id, send the request to
    http://localhost:9098/bookstoremenu/management/40000001

5.The following response is expected,
{
  "id": "40000001",
  "bookStoreMenuItems": [
    {
      "itemName": "Java textbook-2",
      "itemDescription": "This is Java textbook",
      "mddPrice": 5,
      "itemId": "40000001002"
    }
  ]
}
which demonstrates that the item has been sent successfully.

6.In the notifyNewBookStoreMenu of book-store-menu-controller, with
    POST and parameter of bookstore operator's id, send the request
    to
    http://localhost:9098/bookstoremenu/management/newmenunotification/40000001
    to notify the community members that bookstore operator of
    40000001's menu has updated.

7.In the log of externalsys-srv, the following message is expected
    to demonstarte that the bookstore menu update email has been
    sent to the community members.
{
*** Processing email request ***
* - from:   40000001
* - to:     allcommunitymembers@mcmaster.ca
* - Object: New BookStore Menu From 40000001
* - Message:
Hey there!

40000001 updated bookstore menu, you are welcome to take a look
```

```
Here is the updated menu:
BookStoreMenuItem(ItemId=40000001002, ItemName=Java is is Java
    textbook, MddPrice=5.0)


Regards
*** Ending email request ***
}
```

- Bookstore operator can cash MDDs from member's wallet and offer a selection of goodies/merch

---

```
1.Visit
    http://localhost:9093/swagger-ui/index.html#/book-store-cart-controller

2.In addToBookStoreCart of book-store-cart-controller, with POST
    and parameter of communitymemberid of 10000002, send the
    following payload to
    http://localhost:9093/bookstorecart/management/10000002 to add
    item into bookstore cart.
{
  "bookStoreItemId": "40000002001",
  "bookStoreItemName": "Java textbook1",
  "bookStoreOperatorId": "40000002",
  "mddPrice": 5,
  "quantity": 2
}

3.The following response is expected
{
  "statusCode": 200,
  "response": "The item has been added successfully"
}
which means the community member of 10000002 added the bookstore
    item of 40000002001 from bookstore operator of 40000002. The
    MDD price and the quantity are 5 and 2, respectively.

4.In getBookStoreCart of book-store-cart-controller, with GET and
    parameter of communitymemberid of 10000002, send the request to
    localhost:9093/bookstorecart/management/10000002

5.The following response is expected
{
  "id": "10000002",
  "bookStoreCartItems": [
    {
      "bookStoreOperatorId": "40000002",
      "bookStoreItemName": "Java textbook1",
      "bookStoreItemId": "40000002001",
```

```
      "mddPrice": 5,
      "quantity": 2
    }
  ],
  "total": 10
}
```
which means the item has been added to the cart of community member whose id is 10000002.

6.In checkoutBookStoreCart of book-store-cart-controller, with POST and parameter of communitymemberid of 10000002, send the request to http://localhost:9093/bookstorecart/payment/10000002

7.The following response is expected
```
{
  "statusCode": 200,
  "response": "Checking out the cart"
}
```
which means the order is being placed and community member is paying the order.

8.In the log of externalsys-srv, the following message demonstrates that the communitymemberid of 10000002 has received the billing email for bookstore order placed in this test.
```
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     u2@mcmaster.ca
* - Object: MacDrop Bookstore Billing
* - Message:
*****************************************
Hi, Here is your bookstore billing on MacDrop:
Items details:
Items:[Java textbook1(ID:40000002001); ] From Bookstore:[40000002]
    Price:[MDD 10.00]



Total price: MDD 10.00

Regards,
MacDrop
*****************************************

*** Ending email request ***
}
```

9.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM COMMUNITYMEMBER, the CURRENTMDD(MDD balance) of community member of 10000002 is 990, which has been deducted by 10 from

the original 1000.

10.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM BOOKSTOREOPERATOR, the CURRENTMDD(MDD balance) of bookstore operator of 40000002 has been increased by 10 from 200, which means the bookstore operator of 40000002 earned 10 MMD from community member of 10000002.

- At the end of each week, Bookstore operator can synchronize the list of goods sold with MDD money, and automatically send an invoice to McMaster for compensation

---

1.Connect H2 database with URL of http://localhost:81 and JDBC URL of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM BOOKSTOREORDER, there are some bookstore orders created when initialized service. There are two bookstore orders for bookstore operator of 10000006 in the given week from which 2022-10-17 00:00:00 to 2022-10-23 23:59:00. We are going to ask for MDD compensation for bookstore operator of 10000006 in this given week.

2.Visit http://localhost:9095/swagger-ui/index.html#/

3.In askForMddCompensation of book-store-order-operation-controller, with POST and parameter of bookstoreoperatorid of 40000001, send the payload to http://localhost:9095/bookstoreorderoperation/bookstoreoperator/mddcompensation/40000001

```
{
  "leftday": "17",
  "lefthour": "00",
  "leftmin": "00",
  "leftmonth": "10",
  "leftsec": "00",
  "leftyear": "2022",
  "rightday": "23",
  "righthour": "23",
  "rightmin": "59",
  "rightmonth": "10",
  "rightsec": "59",
  "rightyear": "2022"
}
```

4.The following response is expected

```
{
  "statusCode": 200,
  "response": "Asking for compensation..."
}
```

```
5.In the log of externalsys-srv, the following message indicates
    that an invoice has been sent from bookstore operator to
    administrator
{
*** Processing email request ***
* - from:   bso1@mcmaster.ca
* - to:     admins@mcmaster.ca
* - Object: BookStore MDD Compensation [40000001]
* - Message:
Dear Administrator,

The following are the list and invoice for bookstore MDD
    merchandise between 2022-10-17 00:00:00 and 2022-10-23
    23:59:59. Please compensate me accordingly.
Items sold with MDD:
Bookstoreorder(bookstoreorderid=2, communitymemberid=10000006,
    bookstoreoperatorid=40000001, bookstoreitems=Java
    textbook1(ID:40000001001);, mddprice=20, ordertime=2022-10-19
    10:44:52)
Bookstoreorder(bookstoreorderid=3, communitymemberid=10000006,
    bookstoreoperatorid=40000001, bookstoreitems=Java
    textbook2(ID:40000001002);, mddprice=10, ordertime=2022-10-20
    10:44:52)

Total MDD:
30.0

Regards,
bso1
*** Ending email request ***
}

6.In the log of externalsys-srv, the following message indicates
    that the administrator has transferred the compensation to
    bookstore operator.
{
*** Processing ETF transfer request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     bso1@mcmaster.ca
* - amount: 30.00
* - Message: The Payment From MacDrop Admin For MDD Compensation
*** Ending ETF transfer request ***
}
Please note that, in the system, 1 CAD equals 1 MDD.

7.In the log of externalsys-srv, the following message indicates
    that the administrator has emailed the bookstore operator that
    the compensation has been transferred.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
```

```
* - to:     bso1@mcmaster.ca
* - Object: MacDrop MDD Compensation
* - Message:
*****************************************
Hi, Your Total Amount of MDD Compensation: CAD30.00 Is On The Way.

Regards,
MacDrop
*****************************************

*** Ending email request ***
}

8.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    BOOKSTOREOPERATOR, the CURRENTMDD of bookstoreoperator of
    40000001 has been deducted by 30 MDD from 200 since the 30 MDD
    have been compensated.
```

3.6 Test scenarios for administrator

- Administrator operator registration

```
1.Visit http://localhost:9092/swagger-ui/index.html#/

2.In createAdmin of admin-controller, with POST, send the following
    payload to http://localhost:9092/admin
{
  "etfemail": "adminusername1@mcmaster.ca",
  "firstname": "adminfirstname1",
  "lastname": "adminlastname1",
  "password": "***",
  "username": "adminusername1"
}

3.The following response is expected.
{
  "statusCode": 200,
  "response": "Created administrator successfully!"
}

4.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    ADMIN, a administrator whose username is adminusername1 is
    shown.

5.In the log of externalsys-srv, the following message is shown to
    demonstrate that the administrator creation message has been
    sent.
```

```
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     adminusername1@mcmaster.ca
* - Object: Administrator Registration on MacDrop
* - Message: Administrator registered successfully! Your username
    is adminusername1
*** Ending email request ***
}

6.In the log of usermanagement-srv, the following message is shown
    to demonstrate that the administrator creation message has been
    sent.
{
** Preparing sending email
** Using instance: http://172.18.0.10:9091
** Response: EmailResponse(smtpStatusCode=250,
    smtpServerResponse=OK)
** Email Sent
}
```

- Administrator can monitor the usage of the different drop-off locations

---

```
1.Visit http://localhost:9096/swagger-ui/index.html#/

2.In /admin/locations of admin-controller, with GET, send request
    to http://localhost:9096/admin/locations

3.The following response is expected
{
[
  {
    "locationid": 80000000,
    "name": "location0",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000001,
    "name": "location1",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000002,
    "name": "location2",
    "currentworkload": 0,
    "historyworkload": 1
  },
```

```
  {
    "locationid": 80000003,
    "name": "location3",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000004,
    "name": "location4",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000005,
    "name": "location5",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000006,
    "name": "location6",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000007,
    "name": "location7",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000008,
    "name": "location8",
    "currentworkload": 0,
    "historyworkload": 0
  },
  {
    "locationid": 80000009,
    "name": "location9",
    "currentworkload": 0,
    "historyworkload": 0
  }
]
}
```


which shows the current workload and historical workload in different drop-off location.

4.The response above demonstrates that the administrator can monitor the usage of the different drop-off locations.

- Administrator can monitor the workload associated to bikers

```
1.Visit http://localhost:9096/swagger-ui/index.html#/

2.In getCurrentBikerLoads of admin-controller, with GET, send the
    request to http://localhost:9096/admin/bikers/currentworkloads
    to monitor the current workloads of bikers.

3.The following response showing real-time workload of bikers is
    expected
{
  "30000000": 0,
  "30000001": 0,
  "30000002": 0,
  "30000003": 0,
  "30000004": 0,
  "30000005": 0,
  "30000006": 0,
  "30000007": 0,
  "30000008": 0,
  "30000009": 0
}
Since there is no biker delivering food, so all real-time workload
    are 0.

4.In getHistoryBikerLoads of admin-controller, with GET, send the
    request to http://localhost:9096/admin/bikers/historyworkloads
    to monitor the historical workloads of bikers.

5.The following response showing historical workload of bikers is
    expected
{
  "30000000": 0,
  "30000001": 0,
  "30000002": 1,
  "30000003": 0,
  "30000004": 0,
  "30000005": 0,
  "30000006": 0,
  "30000007": 0,
  "30000008": 0,
  "30000009": 0
}
Since the biker of 30000002 completed 1 food order, so the
    historical workload of biker of 30000002 is 1.

6.The responses above demonstrate that the administrator can
    monitor the real-time workload and historical workloads
    associated with different bikers.
```

- The results of the platform (in terms of safety) need to be

balanced by its costs (number of hired bikers, MDD reward program goodies) and presented each couple of months to the Board of Governors

---

1.Visit http://localhost:9096/swagger-ui/index.html#/

2.In getBikerCenterCost of admin-controller, with GET and parameter of fromTime and toTime, send the request to localhost:9096/admin/financial/bikercentercost/{fromTime}to{toTime}, this is to get the biker center cost during the given time.

3.Please note that the format of the time is YYYY-MM-DD, we are going to get financial report of biker center during October of 2022. So the request URL mentioned above is http://localhost:9096/admin/financial/bikercentercost/2022-10-01 to2022-10-31.

4.The expected response is the figure of biker center cost.

5.In the system, the biker center cost during a given time is
Biker center cost = number of complete food order * 10

6.In getBookStoreCost of admin-controller, with GET and parameter of fromTime and toTime, send the request to
localhost:9096/admin/financial/bookstorecost/{fromTime}to{toTime}, this is to get the bookstore cost during the given time.

7.Please note that the format of the time is YYYY-MM-DD, we are going to get financial report of bookstore during October of 2022. So the request URL mentioned above is http://localhost:9096/admin/financial/bookstorecost/ 2022-10-01to2022-10-31

8.The expected response is the figure of bookstore cost.

9.In the system, the bookstore cost during a given time is Bookstore cost = number of MDD compensated during the given time 1 MDD = 1 CAD

10.In getFinancialReport of admin-controller, with GET and parameter of fromTime and to time, send the request to /admin/financial/report/{fromTime}to{toTime}, this is the get the financial report for both biker center and bookstore during the given time.

11.Please note that the format of the time is YYYY-MM-DD, we are going to get financial report of both biker center and bookstore during October of 2022. So the request URL mentioned above is http://localhost:9096/admin/financial/report/2022-10-01to2022-10-31.

```
12.The following response is expected
{
  "Biker Center Cost, Cnd$": 60,
  "MDD Reward Program Cost, Cnd$": 40,
  "Cost In Total (2022-10-01 to 2022-10-31), Cnd$": 100
}
```

- At the end of the day, Administrator can approve the transactions during the day and food provider receives a single Interac transfer from McMaster for all the orders delivered during the day, as well as a detailed billing statement summarizing all transaction of the day

---

```
1.Connect H2 database with URL of http://localhost:81 and JDBC URL
    of jdbc:h2:tcp://localhost:1521/macdrop. Run SELECT * FROM
    FOODORDER, there are some food orders created for tests when
    initialized the service.

2.Visit http://localhost:9096/swagger-ui/index.html#/

3.In approveDailyFoodOrderTransactions of admin-controller, with
    GET and parameters of id and date, send request to
    localhost:9096/admin/dailytransaction/foodorder/approval/{id}/{date}
    to approve the daily transaction for the specific food provider.

4.Please note that the format of the time is YYYY-MM-DD, the
    parameter of date must follow the format or it will fail to
    execute.

5.We are going to approve the complete transactions for food
    provider of 20000005 on October 30, 2022. So the requet URL is
    http://localhost:9096/admin/dailytransaction/foodorder/approval/
    20000005/2022-10-30

6.In the log of externalsys-srv, the following message shows that
    the food provider receives a detailed billing statement
    summarizing all transactions of the day through email.
{
*** Processing email request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     fp5@mcmaster.ca
* - Object: MacDrop Food Billing Statement
* - Message:
*********************************************
Hi, Here is the daily transaction summary of your food store on
    MacDrop today by 2022-10-30 23:59:59:
Billing Details:
Transaction Time: [2022-10-30 10:46:54] Food Item:
    [food1(ID20000005001); food2(ID20000005002);] Purchased by
```

```
    Community Member: [10000004] Price: [CAD 120.00]
Transaction Time: [2022-10-30 10:59:54] Food Item:
    [food1(ID20000005001); food2(ID20000005002);] Purchased by
    Community Member: [10000005] Price: [CAD 100.00]


Your Total Amount On The Way: CAD 220.00

Regards,
MacDrop
*****************************************

*** Ending email request ***
}

7.In the log of externalsys-srv, the following message shows that
    the food provider receives an ETF payment from administrator
    for all transactions of the day.
{
*** Processing ETF transfer request ***
* - from:   macdropadminoffice@mcmaster.ca
* - to:     fp5@mcmaster.ca
* - amount: 220.00
* - Message: The Payment From MacDrop Admin
*** Ending ETF transfer request ***
}
```
