Architecture Report: ACME Run

Group 2 - Yi Qiao, Xiaoran Xie, Cynthia(Yingxue) Liu

1. Bounded Contexts & Services Boundaries Justification

1.1 Bounded Contexts

In our service, we have defined **five** bounded contexts to separately address specific aspects of our system's domain. In the following section, we provide a comprehensive introduction to each of them, along with the justifications for their existence.

The first one is the **User Center** which manages the user information, including create, read, update, and delete. It is also the place where the user chooses to launch the application. By isolating these functionalities, we ensure that user information is consistently managed, and their data is secure and accessible only where needed.

The second one is the **Trail Center**, which plays a pivotal role in allotting the trails. After users create their accounts, the trail center will allot trails according to their geographic locations. Isolating trail allocation could optimize the allocation process, for example, when the user enters a new area, it could help find the trails nearby more efficiently.

The third one is the **Game Center**, which offers users the available options according to their previous records. This is the most critical part of our service, as it includes remembering and analyzing the users' decisions and adapting their training plan to provide the best fitness experience. With the game center as a bounded context, we could adjust the algorithm of generating the fitness plan more flexibly without needing to modify other code.

The fourth one is the **Operation Center** which tells players what to do after they make a decision. Isolating this part enhances the modularity of the code, ensuring the user interaction and service response are developed separately so they can be modified and improved more effectively.

The last one is **Challenge Center**, which is the center that manages the global challenges and generates the badges. Isolating these functions ensures a clear area of responsibility, that is, we could give this section to the domain experts to analyze what challenges to put into play next, without any conflict or ambiguity.

1.2 Services Boundaries Justification

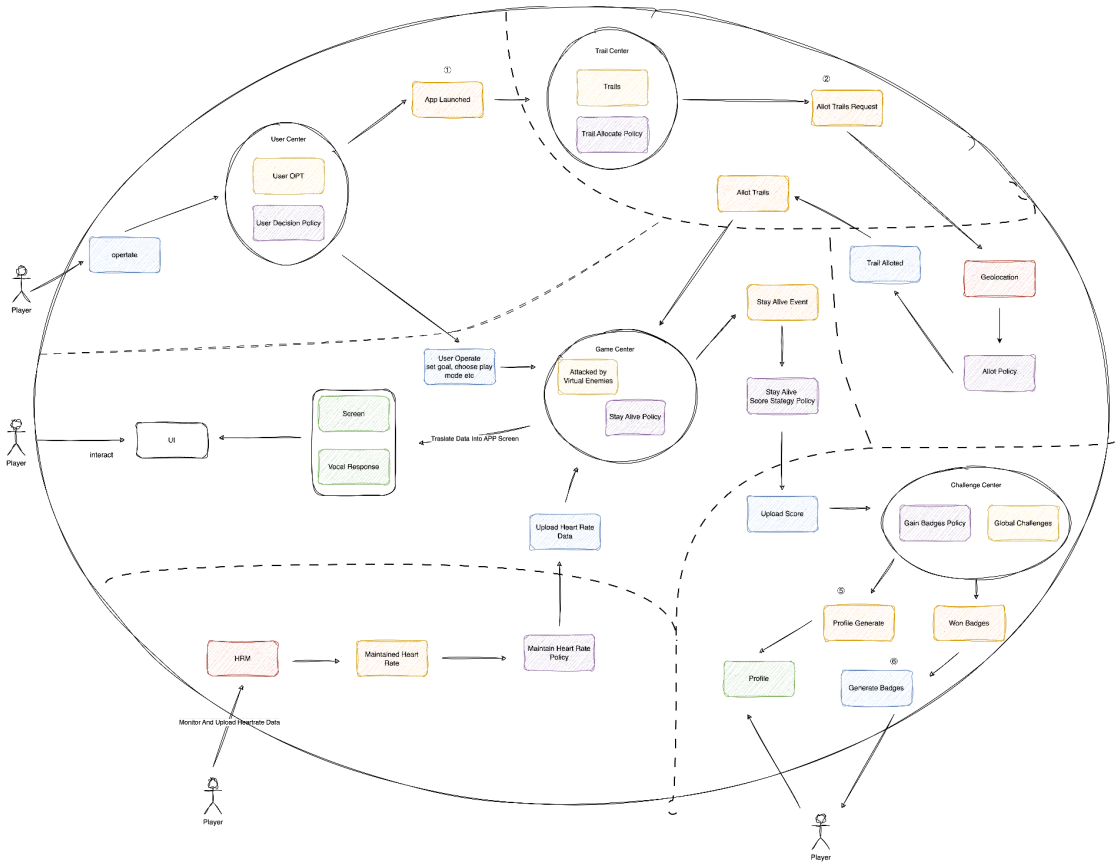
For the service boundaries, we have divided our service into five parts. In the following section, we introduce each of them focusing on how they interact and communicate with each other.

We have established three of our distinct service boundaries based on the bounded context, which are **trail management**, **game management**, and **challenge management**. Then, we combine the user center and operation center into a module called **user management**. We also have the **HRM** as an external system. In our project, they are called **trail-center**,

game-center, **challenge-center**, **user-opt**, and **heartrate-monitor** correspondingly. We also use the eureka service and the folder name for it is **discovery-server** in the code.

After the user registers and starts one workout, their location information is shared with the trail management to allot the available trails near them. Then when the users start running, the trail management intelligently shows the shelter option based on the user's real time location. The external HRM, functioning as a heart rate and motion recorder, plays an important role in data acquisition by detecting user actions and transmitting this information to the workout management. After the users finish running, their performance score will be calculated and passed to the challenge management, which measures the challenge process and generates badges correspondingly.

1.3 Event Storming Diagram



This is our event storming diagram, you can find the process of developing the diagram and the zoomable version in the following link:
https://app.diagrams.net/#HCAS735-F23%2Fmacrun-yi_xiaoran_xue%2Fyi_test%2F735.drawio#%7B%22pageId%22%3A%22UdRP8n3MG-tA0bp-XavI%22%7D

2. APIS

2.1 Asynchronous API

Using asynchronous APIs in a microservices architecture offers several advantages. Here are some primary reasons for adopting asynchronous communication in such setups:

A. Decoupling of Services
B. Improved Response Times
C. Fault Tolerance
D. Scalability
E. Load Leveling

We plan to use 6 asynchronous interfaces in our project(We have labeled the DDD field design diagrams with a circle + a number), which are:

Interface Number	Application	From … To …	Primary Reason
①	To broadcast a user successfully launch the app	From【User Center】to【Trail Center】	A D
②	To assign paths to the geographic system	From【Trail Center】to【Game Center】	A
③	HRM system departure events	From【HRM】to【Game Cenrter】	A
④	Transmits the user score to the Challenge Center	From【Game Cenrter】 to【Challenge Center】	B
⑤	User Generate Profile	From【Challenge Center】to【User】	A B D
⑥	User Generate Badge	From【Challenge Center】to【User】	A B D

Justification:

① Broadcasting user registration success messages is asynchronous to ensure that the user registration process is not interrupted by the messaging system. This falls under the categories of "service decoupling" and "scalability" as it allows the registration service to run independently of the messaging service and to handle large numbers of registrations without delay.

② The assignment of paths to the geographic system is asynchronous to accommodate processing times without delaying the user's interaction with the system. This is a case of "improved response time" because the user does not have to wait for the path assignment to complete.

③ Allowing heart rate monitoring data to be transmitted independently without affecting the user experience improves system fault tolerance and response speed while realizing the decoupling of services.

④ Transmission of user scores to the Challenge Center can be asynchronous to "improve response times" for users who complete the challenge so they don't have to wait for scores to be transmitted and recorded.

⑤ User-generated profiles may be asynchronous to "load-balance" the profile creation

process and ensure that the system is not overwhelmed by multiple simultaneous profile creation requests.

⑥ The "User Generated Badge" interface is asynchronous and contributes to "Service Decoupling" as the badge generation system can be operated independently of the user and "Load Balancing" is achieved by efficiently managing badge generation requests during high traffic.

2.2 Synchronous API

1. The user launches the application
POST http://localhost:8762/user/starter
This interface is used for launching the MacRun where the user provides personal information to start the application. It is synchronized because immediate feedback is critical for the user to know if their launch is successful.

2. The user gets user details after launching the app
GET http://localhost:8762/user/details/?userID={userID}
This interface is used for fetching the user's information when the user provides the userID. It is synchronized because the user can immediately access the personal details.

3. The user gets trail allocation results
GET http://localhost:8762/trail/details/?userID={userID}
This interface is used for fetching the trail allocation results when the user provides the userID. The reason for this synchronization is that the user can immediately get the assigned path.

4. The user sets customized geographic information to get trail allocation
POST http://localhost:8762/trail/localization
This interface is used to get the path assignment result when the user provides geolocation. By default, the system automatically gets the user's location and assigns a path to the game. Still, we support the user to customize the location in case they think it is inaccurate. The reason for this synchronization is that the user can immediately re-supply the geolocation and get the assignment result.

5. The user sets actions and interacts with the game
POST http://localhost:8762/game/action
The interface handles user interaction within the game environment. The synchronized nature of this interface is critical to responding to user actions in real-time and ensuring a seamless interactive gaming experience.

6. The user interacts with HRM and starts a new workout
POST http://localhost:8762/hrm/workout/starter
The interface is used for starting a new workout session and sending heart rates to the game service. By default, once this app is launched, the default heart rates are sent to the game service. However, once the user starts a new workout, the corresponding heart rate replaces the default heart rate and is sent to the game service. The reason for this synchronization is that our system needs to keep track of the user's heart rate instantly when they start a workout.

7. The user interacts with HRM and gets one workout detail by userID
GET http://localhost:8762/hrm/workout/details/?userID={userID}
This interface can fetch the user's workout details when the user provides the userID. The synchronization of this interface is very important, which allows the user to confirm the status of the current workout session in real-time.

3. Service Pattern

An essential aspect of microservices is the ability to effortlessly scale our services up or down to handle peak load periods. A key concept in achieving this goal is the effective use of load balancing. Load balancing evenly distributes traffic across multiple instances of the same service, preventing overload on a single instance from impacting our system. Therefore, in our project's service pattern, we have addressed the design needs of our microservice architecture by focusing on service discovery, gateway, and load balancing. This approach ensures a robust and scalable system capable of handling varying demands efficiently.

We first implemented **Service Discovery** using Eureka, registering the services in our project with Eureka's server. This approach allows us to avoid hardcoding the locations of services, enabling them to discover each other dynamically. Eureka's dynamic discovery mechanism facilitates seamless communication between different services, enhancing the system's overall resilience and flexibility. We can access the Eureka interface by using http://localhost:8761/ to check out our registered services. It simplifies service management and ensures that new instances or changes in service locations are automatically updated and propagated throughout the system.

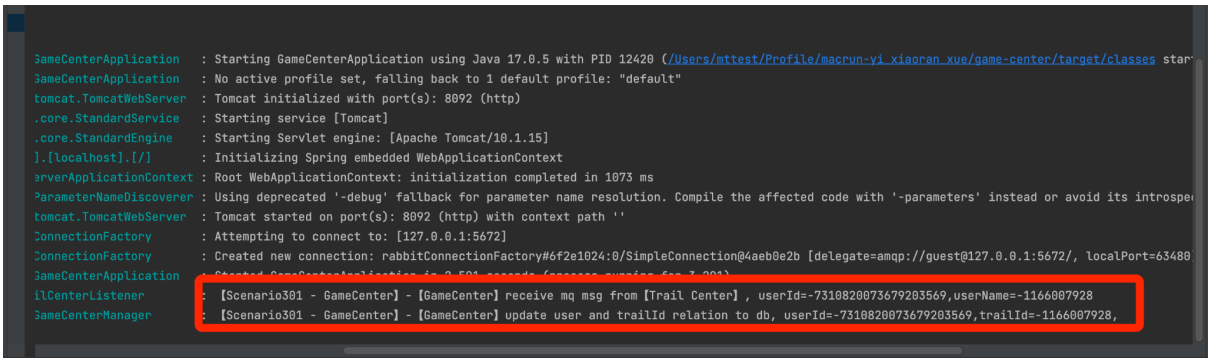
Then we used Spring **Gateway** to distribute synchronous requests. We assigned port 8762 to the Gateway and configured it using the IDs, URIs, and predicates of different services in our system. With this setup, all synchronous requests can be directed through the same port, enhancing the system's organization and efficiency. For instance, when the launching app request is like POST http://localhost:8762/user/starter, the Gateway pattern will help us to direct to UserOpt service by our configuration. This centralized management simplifies routing logic, security, and monitoring. Also, the Gateway's configuration allows for dynamic routing based on service IDs and URIs. This flexibility enables us to easily modify routing rules and service locations without disrupting the overall system.

For another example, when multiple users create workout sessions by using POST http://localhost:8762/hrm/workout/starter, by leveraging the Gateway's built-in **Load balancing** features, we can start multiple HRM service instances and Gateway will help us to distribute request traffic across various service instances evenly. The integration of a Gateway with load-balancing capabilities is a strategic choice in our microservices architecture, contributing to a more scalable, maintainable, and efficient system.

Altogether, our microservice architecture leverages the combined strengths of service discovery, gateway and load balancing pattern, which together create a dynamic, resilient, and scalable system that is well-equipped to handle the ebb and flow of service demands.

4. Scenarios

We use scenario ID to number the scenarios, and the scenario numbers can be found in the code in the corresponding business logs, for example



You can find Scenario301 in our code in this format:
【ScerarionId - Domain】-【From Domain】actions【To Domain】, log detail info

Scenario 101 Launch Application

Before the game starts the user needs to complete the registration of the personal account, the user needs to use the email address and user name for the registration process. Once registered, the user is automatically assigned a userId which is used to uniquely identify each user in the system and to track the user's behavior and actions. In addition, the system obtains the latitude and longitude of the user's location, which will be useful in subsequent path assignment scenarios. After that, the user can choose to launch the application immediately and start running, but to simplify the process, in our project, the users always do so. This is the core foundation of the application and has a **high** priority on the technical side. This ensures that the user can create an account and start using the application smoothly and securely. User information also has a **high** business value and is essential for understanding user needs and personalized marketing.

Scenario 102 User Operation

During gameplay, the user is required to choose from several behavioral options based on what they encounter on the map. These behaviors include not fighting, sheltering, escaping, and fighting back. This scenario has **medium** business value in our service as it is the base gameplay of the game. The user operation only provides support for subsequent services and is not directly involved in commercial behavior. In addition, this part has **low** priority on a technical level as there is not much logic to implement.

Scenario 201 Allocate Trail

After a user has registered, the system obtains the user's geographic location, including latitude and longitude. The user center then sends the user's information and geographic location to the Trail Center, where the geographic location is sent back to an external system to request a specific trail allocation. When the allocation is complete, the Trail Center receives a response from the external system, which will contain a specific trail that appears on the map. This trail will be used for the rest of the game, and the different attributes in the trail correspond to the different situations encountered in the game, including shelters, enemies, and so on. This part has a **low** business value because the trial allocation is more of a basic part of the game and is not involved in commercial activities. In addition, although

this part largely supports the basic gameplay, the allocation algorithms originate from an external system, so trail allocation has only **medium** priority on a technical level.

Scenario 301 Update the relation of trailId and userId

This part of the priority is **low** considering both business value and technical effort, its main responsibility is the accumulation of data in the game center, that is, the relationship between the user and the distribution path. Through this allocation relationship, we can know the path where the user is located.

Scenario 302 Update HRM data and generate play mode(sheltering, escaping, or fighting back)

This is the most important part of our non-commercialization module, the process is through the user's choice of game mode and the heart rate detected by the HRM system, through the calculation algorithm to generate the user's play mode, such as A, B, C. Then through the way of MQ, the play mode will be displayed to the UI in the form of a written model, and then the user can interact with the game through the UI. This is of **high** priority in terms of technical effort, as it is a core piece of our system because it serves as a link in the gameplay generation model, without which the game will not be able to proceed. This is of **moderate** business value as it is a foundational element without which the game cannot proceed, ensuring a seamless and tailored gaming experience for our users.

Scenario 303 The user operates and sends the score to the challenge center

This part is a very important part of the game center, which calculates the score obtained by the user through the combination of the user's actions and the user's heart rate and other data. In a real-world scenario, the algorithm could be more complicated as it should also take into account of distance they run, the achievement they get, etc. Another approach of delivering the information is to send everything such as the running distance and running time to the challenge center to generate corresponding badges, but for simplification, we only send the score to the challenge center in the form of an MQ. This has **high** priority for technical effort, as it is the key part that generates the result and presents it to the user. It also has **high** business value because the score could act as the biggest motivation and give users a sense of accomplishment as it visualizes their workout results, which could improve user loyalty.

Scenario 401 Integration with External HRM

Users may use external heart rate monitors to collect and transmit their real-time heart rate to the game center every five seconds. This scenario is of **high** business value, as it enables us to track and record users' health information, increasing the precision of fitness tracking and personalization. Ultimately, this leads to enhanced user experience and overall product value. The technical effort needed is **moderate**, our service needs to listen to and temporarily store the heart rate data for use within the game center.

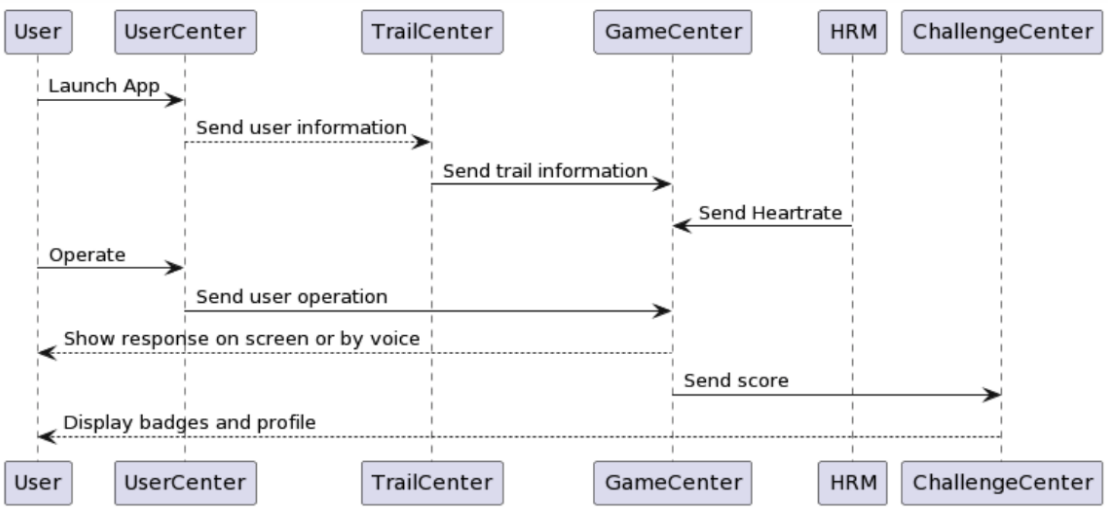
Scenario 501 Generate Profile & Badge

After completing a trial, the game center sends the players' scores to the challenge center, where they are awarded virtual badges as recognition and achievement based on their scores. These badges can be displayed on the users' profiles, which they can access at any time to manage the displayed badges. This scenario is of **high** business value in our service,

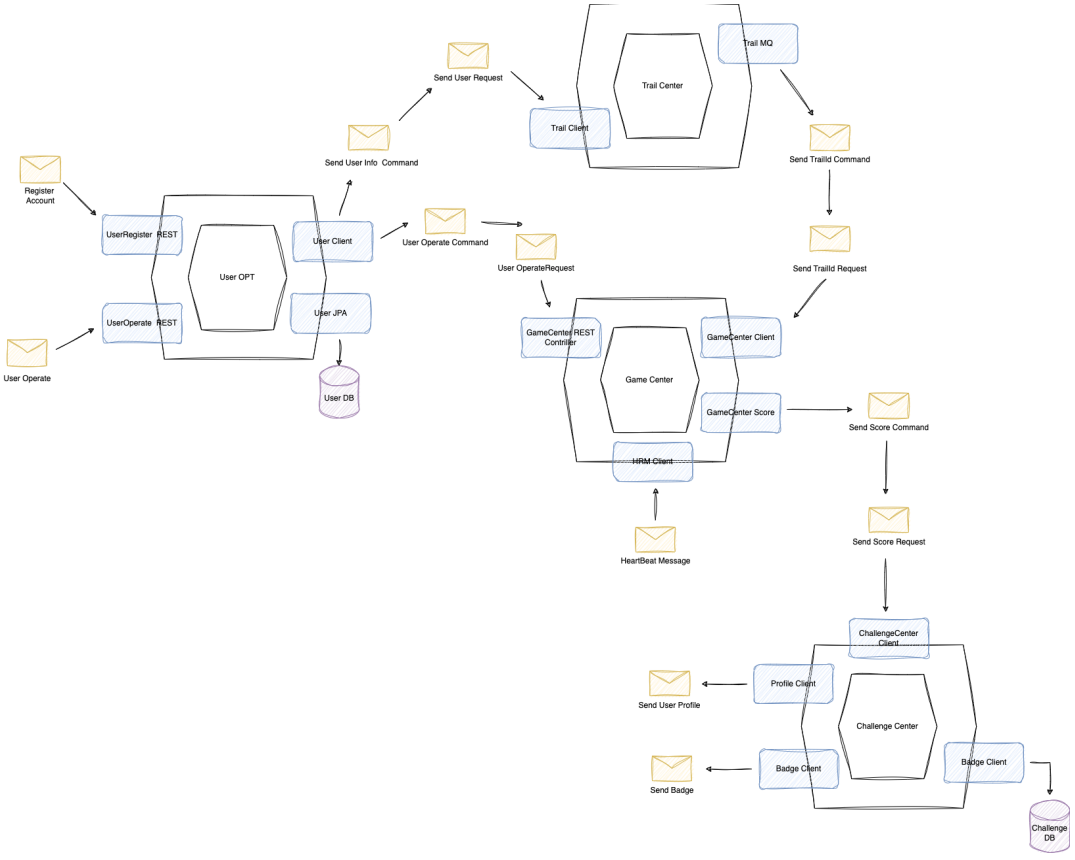
as the management of these challenges and badges serves to motivate and engage users. Implementation of this gamification element may act as a critical driver for user retention and satisfaction. The level of technical effort required is **moderate** and is heavily contingent on the algorithm's complexity utilized in generating badges.

5. Graphical Representation

5.1 Sequence Diagram



5.2 Hexagons



6. Executable Script

The executable script of the postman can be found in the following link. Simply click on the link to open it and fork the collections to run on the local machine.
https://www.postman.com/xiaoranxie/workspace/cas735-project/collection/30263408-35a84955-db56-458b-8094-cb62825e07e3?action=share&creator=30263408

7. How to run the scenario script and test

To run our project, simply download the zip file from our GitHub repo. Then, open up the terminal cd into the deployment folder, and run the services by "*docker-compose up*". The service should start running automatically. Kindly wait for a few minutes(eg. 5 mins) as running might take some time. The following section will show the complete process of a game.
Another way to run this service is to use the "*docker-compose -f docker-compose-prod.yml up*" to pull the image from the docker hub.
The unit test and integration test could also be run by "*mvn test*" in the main project folder.

Firstly, we can create a new user account with Postman using the method in 6.1. Then, the terminal will print the following log message indicating the success of user creation, which is scenario 101 as mentioned in section 4.

```
user       | 2023-12-06T01:50:12.454Z  INFO 1 --- [nio-8090-exec-1] c.c.p.useropt.adapters.UserController    :
【Scenario101 - UserCenter】-【UserCenter】launch application and send user info to【TrailCenter】, userName=yx
l
user       | 2023-12-06T01:50:12.458Z  INFO 1 --- [nio-8090-exec-1] c.c.p.useropt.business.UserCrudManager   :
【Scenario101 - UserCenter】-【UserCenter】user launching msg sent to mq (Destination: 【TrailCenter】), userId
=4103110152070936864,userName=yx l
```

Then the user center could send the user ID to the trail center, where the trails will be allocated to the user according to the user's geography location. After that, the user information with the allocated trail will be sent together to the game center, as in **scenario 201**.

```
trail      | 2023-12-03T15:40:35.633Z  INFO 1 --- [ntContainer#0-1] c.c.p.t.d.listener.TrailCenterListener   :
【Scenario2 - TrailCenter】-【TrailCenter】receive mq msg from【UserCenter】, userId=-3268886005374161139,userNa
me=yx l
trail      | 2023-12-03T15:40:35.703Z  INFO 1 --- [ntContainer#0-1] c.c.p.t.business.TrailCenterManager      :
【Scenario201 - TrailCenter】-【TrailCenter】allocate trail, userId=-3268886005374161139,trailId=1358570058,trai
lName=The 5th Campus Trail
trail      | 2023-12-03T15:40:35.706Z  INFO 1 --- [ntContainer#0-1] c.c.p.t.business.TrailCenterManager      :
【Scenario201 - TrailCenter】-【TrailCenter】send trail msg to【GameCenter】, userId=-3268886005374161139,trailI
d=1358570058,trailName=The 5th Campus Trail
```

Although this is not a critical part of the whole service, the trail, and user information can also be updated to the database, as **scenario 301**.

```
game       | 2023-12-03T10:21:54.796Z  INFO 1 --- [ntContainer#0-1] c.c.p.g.d.e.l.TrailCenterListener        :
【Scenario301 - GameCenter】-【GameCenter】receive mq msg from【Trail Center】, userId=-4814753499671256197,user
Name=-404496387
game       | 2023-12-03T10:21:54.821Z  INFO 1 --- [ntContainer#0-1] c.c.p.g.business.GameCenterManager       :
【Scenario301 - GameCenter】-【GameCenter】update user and trailId relation to db, userId=-4814753499671256197,t
railId=-404496387,
```

In **scenario 302**, the game center can receive heart rate from the heart rate monitor, and generate the play mode. To simulate the user action, here we randomized the user action at a rate of every time the game center receives heart rate.

```
game       | 2023-12-03T10:22:01.025Z  INFO 1 --- [ntContainer#2-1] c.c.p.g.d.e.listener.UserHRMListener     :
【Scenario302 - GameCenter】-【GameCenter】receive mq msg from【HRM】, userId=4950311478150564572,heartRate=103
game       | 2023-12-03T10:22:01.026Z  INFO 1 --- [ntContainer#2-1] c.c.p.g.business.GameCenterManager       :
【Scenario302 - GameCenter】-【GameCenter】generate play mode, userId=4950311478150564572,playMode=ESCAPING
game       | 2023-12-03T10:22:01.032Z  INFO 1 --- [ntContainer#2-1] c.c.p.g.business.GameCenterManager       :
【Scenario302 - GameCenter】-【GameCenter】send mq to user device and display in it, userId=4950311478150564572,
playMode=ESCAPING
```

In **scenario 303**, the user action can also be simulated manually by sending a Postman post request as in section 6.2.

```
game       | 2023-12-03T10:23:17.700Z  INFO 1 --- [nio-8092-exec-2] c.c.p.g.adapters.GameCenterController    :
【Scenario303 - GameCenter】-【GameCenter】User start play, userId=123,attackId=123
game       | 2023-12-03T10:23:18.311Z  INFO 1 --- [nio-8092-exec-2] c.c.p.g.business.GameCenterManager       :
【Scenario303 - GameCenter】-【GameCenter】send user score to 【Challenge Center】, userId=123,score=9
game       | 2023-12-03T10:23:18.320Z  INFO 1 --- [nio-8092-exec-2] c.c.p.g.business.GameCenterManager       :
【Scenario303 - GameCenter】-【GameCenter】User start play end and calculate score, userId=123,score=9
```

In **scenario 401**, the heart rate is sent from the external heart rate monitor to the game center every 10 seconds. To better visualize our log message and prevent it from logging too frequently, the rate is set to 10000.

```
heartrate  | 2023-12-03T10:23:00.875Z  INFO 1 --- [   scheduling-1] c.c.p.h.b.HeartRateMonitorManager
【Scenario401 - HRM】-【HRM】sending heart rate to 【GameCenter】: 106bpm
```

In **scenario 501**, the user ID and score are sent from the game center to the challenge center, and the challenge center will generate the user profile and badges if the user achieves one.

```
challenge  | 2023-12-03T10:23:19.173Z  INFO 1 --- [ntContainer#0-1] c.c.p.c.adapter.ScoreListener            :
【Scenario501 - ChallengeCenter】-【ChallengeCenter】Receive message from【GameCenter】. Receive userId = 123,tr
ailId = 9
challenge  | 2023-12-03T10:23:19.348Z  INFO 1 --- [ntContainer#0-1] c.c.p.c.adapter.ScoreListener            : P
rofile: user = 123, highest score = 9, badges = null
```

We sincerely appreciate your time in reviewing our report for the 735 final project. Should you have any inquiries or encounter any challenges while running our application, please do not hesitate to reach out to us through Teams, we had a Team group named CAS 735 Group 2.