# IsRun : Mobile Application for Beginning Runners*

Seungmok Kang[2017313821], Junseok Kim[2017311133], and Hyewon Lee[2018314348]

Sungkyunkwan University, 2066,
Seobu-ro, Jangan-gu, Suwon-si, Gyeonggi-do, Republic of Korea
{twoskysm,factory22,pyan999960}@g.skku.edu

**Abstract.** Running is a simple workout one can perform. However, it is difficult to begin for who not used to. With the lack of motivation, there are many who quit before beginning. Currently, there are services and mobile applications that lead exercise and track one's fitness activities. In a different perspective, it results that they do not include entertaining features to attract beginners. To overcome the limitation, IsRun is designed as a fitness mobile application that includes gaming elements. A combination of running with virtual character simulation game lets users to relate themselves to the characters and visualize the accomplishments. The service will motivate, encourage and bring interest in running to newcomers.

**Keywords:** Running · Virtual Character · Mobile Application

## 1 Introduction

The COVID-19 pandemic has led to a dramatic loss of human life worldwide. One of the losses is outdoor activities which has effect on both physical and mental health. Many locked up themselves home; exercises and workouts outside have stopped. At the same time, interests in healthy lifestyles have increased. Now the world tends to live with the virus and people are searching for social and physical activities. Running is probably what most people can easily think of in the category mentioned. It is a simple and easy but a popular physical activity which individuals can enjoy daily. In addition, it includes benefits such as potential weight loss, improved cardiovascular and respiratory health, as well as psychological well-being.

Running is mostly the start of any cardiovascular exercise. For people who exercise every day, it will be easy to start an easy workout. However, for others who are not familiar with exercise, it would not be easy. There are many services and applications that focus on fitness. Most of them provide exercise analysis features based on distance, time, calories burned, etc. These features are mostly

* Supported by Sungkyunkwan University.

set for people who perform in their daily routine. They provide feedback, but do not provide motivation to continue for newcomers. This brings to a question 'then what would give newcomers motivation in running?'

In the project, the proposed solution is to introduce a mobile application that includes health features combined with gaming elements, providing running assistance and entertainment simultaneously. The application will give motivation by providing entertaining rewards in every running. The rewards include interactions with virtual characters to let people check their current state and achievements.

## 2 Design

In this chapter, we will explain the overall system architecture, which techniques were used, the reasons for using these designs, what challenges we faced and how they were resolved.

### 2.1 Level Design[6][7]

*Distance Recommendation.* We use age, gender, height, and weight data collected as 'UserData' for distance recommendation. When the 'Recommendation' button is turned on, the application automatically calculates recommended distance and time ratio. For 'Recommendation' function, we referred to the Korean army physical fitness test measurement table[10], which contains distance and time according to age and gender. As our 'IsRun' is a running app for new runners, the pace recommendation is based on the lowest fitness level. It was difficult to find criteria for recommending running distance and time related to height and weight data[14]. However, if you are overweight, you can find a study that excessive running can give stress on the knee joint. Accordingly, in the case of users classified as highly obese by height and weight data, it is recommended to set the distance and time according to their capabilities without using the Recommendation function.

*Differing rewards based on levels.* The types of goods used in the game we designed are gold and food. And the elements that can be targeted include the collection of characters and posters, the character's experience, and the character's love value. When designing the level of the game, we designed the level with this in mind because any currency must have its own use.

1. You can get gold+, food+, char.love+, char.full- by RUNNING.
2. You can get char.love+, char.full- through the PLAY interaction, which can be executed once every 3 hours. At this time, the amount of increase of char.love is given a bonus proportional to the char.level.

3. If char.full is not the MAX value, you can FEED it to fill char.full and increase char.exp at the same time. A FEED action consumes food. In this case, the char.exp increment is given a bonus increment proportional to the char.love value.
4. You can acquire new posters and characters by consuming gold.

Target values include exp and collection factors. Means of value include gold, food, and love. Through this design, we were able to design so that all the instrumental values are used. And after the first RUN, two PLAY interactions, and FEEDing them, the reward values were adjusted so that the character of LV1, which is given by default, can become LV2. While leveling up right away like this, we aimed to get users to use the app as an instant feedback on how to use it and the fun of nurturing it.

### 2.2 Frontend

The frontend of this application is divided into two parts. First is running pages and second is gaming pages. Running pages have all the run related pages, such as running records, running start page, etc. So Running pages need GPS coordinates and map views. On the other hand gaming pages focus more on communication between virtual characters and users. Because the characteristics of each pages differ so much, we categorised each with 4 sub-pages.

Running pages consists of following 4 pages.

1. Running Home
2. Records
3. Achievements
4. Locations

Gaming pages consists of following 4 pages.

1. Gaming Home
2. Character Selection
3. Background Selection
4. Gacha

**UI/UX** For all 8 pages, we tried to apply two big concepts, the thumb zone and MVVM design pattern. And for implementing map views, we used Kakao API.

*The thumb zone.* The thumb zone[8] was introduced in Designing Mobile Interfaces, written by Steven Hoober. Along with Hoober, Josh Clark has recorded in-depth information on how people hold their devices in his book Designing for Touch. Hoober's research shows that 49% of people hold their smartphones with one hand, relying on thumbs to do the heavy lifting. Clark took this even further and determined that 75% of interactions are thumb-driven. With this understanding of hand placement, we can conclude that certain zones for thumb movement apply to most smartphones. We'll define them as easy-to-reach(natural), hard-to-reach and in-between(streching) areas(Fig 1).

According to those researches we tried to minimalize number of buttons and play in natural areas as possible. With this design we expected to have more comfortable experience, while using IsRun.

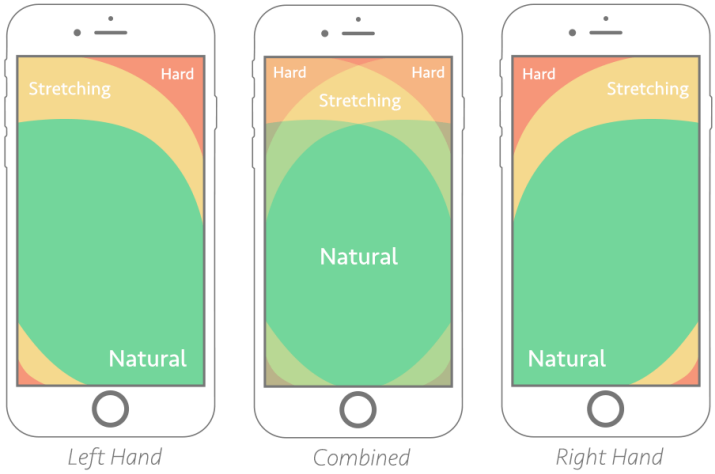



Fig. 1: The thumb zone

*MVVM Design Pattern.* Software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. In other terms this is a loose layout or suggestion for building a software. Among the design patterns, we selected MVVM design pattern. Model–view–viewmodel(MVVM)[9] consists of View, ViewModel, and Model(Fig 2).

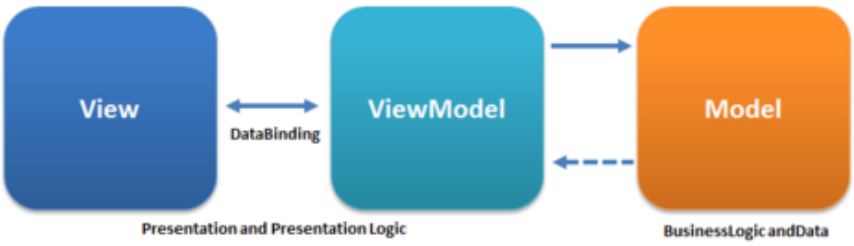



Fig. 2: MVVM Design Pattern

This design pattern facilitates the separation of the development of the view(the graphical user interface) from the development of the model(the busi-

ness logic or back-end logic) so that the view is not dependent on any specific model platform. The viewmodel of MVVM is a value converter, meaning the viewmodel is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented.

For easy understanding, view is showing UI based on the data on viewnodel, while the viewmodel conveys some data updates to model based on actions on view. Applying this to IsRun, which is an android app, views are views(usually .xml file) and the activities(could be kotlin or java file) it consists of. Viewmodel is newly created expanding viewmodel class. Model is the backend server we will make.

MVVM pattern is suitable for IsRun, because the is run is only based on one user data and there are no interaction between each users. This means that the synchronization between client-side data and server-side data can wait, and we can focus on faster response of characters with data on viewmodel. Then we can simply update data on server and get the state of success on data update. This will improve the speed of response from characters.

*Kakao API.* Kakao API[16] is used for maps in running views. Kakao supports Map API for developer users registered on kakao Developers site. The steps to start map API are kindly described on the site. Here is a summary of those steps.

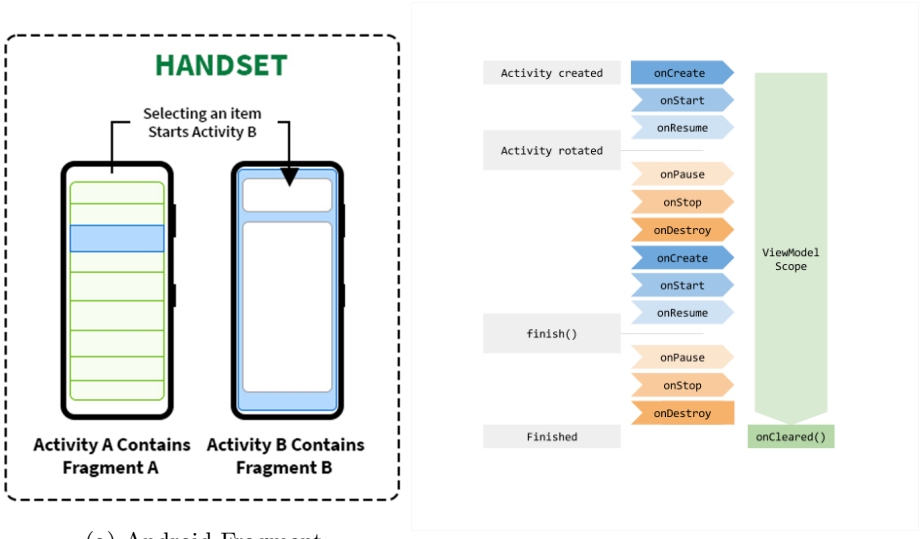
1. Register on kakao Developer site & Generate App
2. Select Platfrom(Android for IsRun)
3. Register Hash key
4. Download SDK & Add to library
5. Set AndroidManifest file
6. Add map view on activity

The most confusing part was registering hash key. hash key is a key to register the application on API for checking. If this app is registered on a store, the store gives hash key and we can publish application with that hash key. Which means we could register this hash code only. However, as IsRun is currently on developing application level, we needed to add all the hash keys generated in devices that IsRun ran on(Home desktop, Junseok's laptop, and Hyewon's laptop).

The reason we decided to use kakao api is because it provides a map in vector image and has the highest number of free inquiries about the "Dynamic map casting" service we want to use.

**Challenges** In this section, we explain about the challenges we encountered while implementing the service and how we overcame them.

*MVVM Design Pattern.* We implemented MVVM with android fragments[12] and navigation[13] with viewmodel. Fragments and navigation is library provided by android studio. Fragments helps user to fragment each screen factors and




(a) Android Fragment

(b) Life cycle of ViewModel

Fig. 3: Fragmets and ViewModel

bind viewmodel for each fragment(Fig 3a). However data of each viewmodel per fragments were not shared. So in order to share data between fragments, we used "MuatbleLiveData" variable type.

MuatbleLiveData[9] is LiveData with Mutable option. Mutable simply means that this data type can be changed somewhat time after, and LiveData means this data is live for a lifecycle of application or activity that this viewmodel is binded to(Fig 3b). The simple picture of how LiveData is used in MVVM is shown in Fig 4. So as long as IsRun is not shut down, all the data in viewmodel are shared with every fragements in IsRun application.

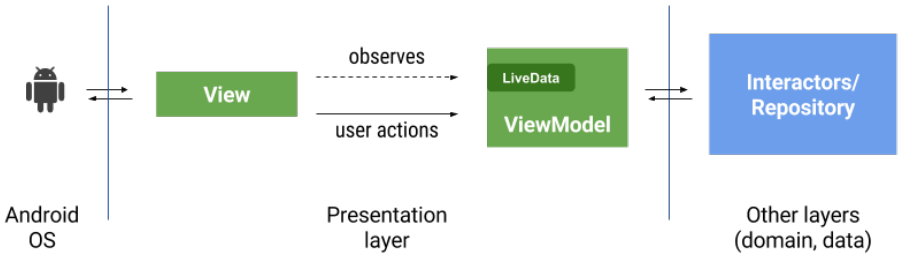



Fig. 4: LiveData in MVVM

Navigation[13] helps navigate through fragments. Android studio provides design page for easy development. As seen in Fig 5, each fragments can be included in navigation path and linked together with action. Of course, the code page is also provided, but we thought using navigation tool would help us visualize how IsRun works.

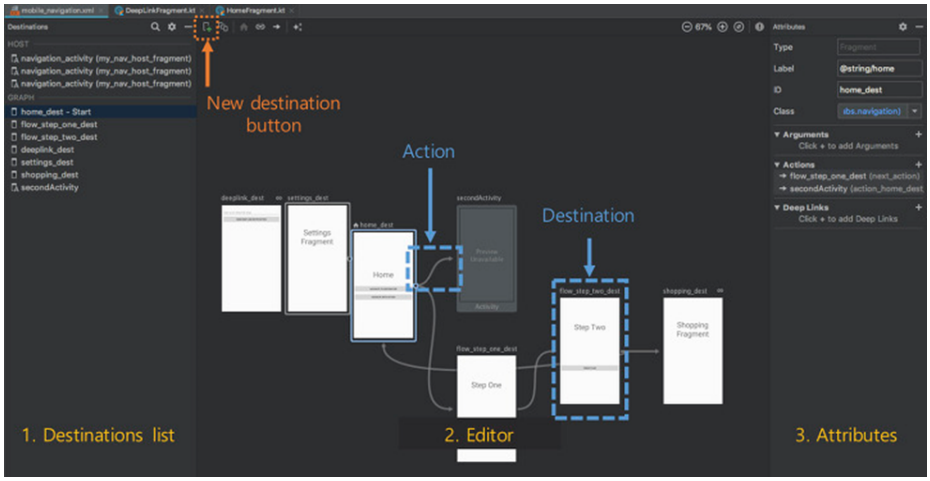



Fig. 5: Navigation example page

*Android Multi-Thread synchronization.* Before looking at challenges we encountered, let's first understand how thread[11] works. Thread is the smallest sequence of programmed instructions that can be managed independently by a scheduler. The implementation of threads and processes differs between operating systems, but in most cases a thread is a component of a process. A process is a sequence of activities, they can be single or multi activity processes. But this doesn't mean activity is thread. Activities can be broken down into multiple threads.

Android supports multi-threading. Although there can be multiple threads, every threads can't start running on same time. There has to be a prior thread, which can start or make other threads. So, by following up the thread, one thread at the beginning is needed, and it is called the main tread.

Main thread is important because any other thread starts from it. Not only that, for android, main thread is the only thread that can update the UI. Now let's think about livedata. Using MVVM pattern, UI is changed based on the data change of viewmodel, which is a livedata form. So, livedata must change in main thread to apply data change on UIs, which android systemically block livedata updates on other thread.

However, when networking with server, IsRun used other thread to wait for responses. So in order to synchronize data in viewmodel and update UIs, we used

two different methods. First, update data on main thread. When the response was successfully achieved, we called main thread and updated data incoming to view model. Second, wait for response in main thread while showing loading page on UI. The second method was usually used in gaming page, such as gacha pages and character select page.

*GPS & KaKao API.* Android needs GPS permissions for using Kakao map API[16] and getting current GPS coordinates. As android versions upgrade, they upgraded level of security, so it was hard to access GPS permissions. Android versions below Android 11 only needed to add permissions to manifest file. However, android versions equal to or greater Android 11 need to add permission check in code level. This resulted in complicated code for every activity of using GPS data.

As a result, we used fused location provider API[15] provided by google-play service. With help of fused location provider, we could easily get GPS permissions and GPS data.

We still had problem with using KaKao maps API. KaKao maps API didn't support using multiple map views in one activity. So we managed this problem with resetting views before moving to another view using map views.

### 2.3 Backend

We designed the server with following architecture.

1. MQTT protocol[4] for data transmission.
2. Using influxDB(TSDB)[3] and mysql(RDB) for storing data.
3. AWS EC2(Electric Compute) for global public access[2].
4. Time interval based cheat prevention.

#### Server model

*Multi threading.* The server is implemented using Multi Threading. In the case of Multi Threading, it is because it is easy to share data between threads and it is possible to prevent delays in using the server between users through thread separation(Fig 6). In the case of MQTT[4], a Pub Sub structure[5] is used that uses a central broker. Due to the nature of these protocols, it is easy to host multiple users and, like a running app, the user's location constantly changes, so it is easy to maintain a connection to the server even when the network environment changes frequently. And due to the nature of the Pub Sub structure and the python3 paho mqtt library we used, if you subscribe to a specific topic and register the processing function in the callback, it has the advantage of automatically executing the function when the data corresponding to the topic comes in. also fit well.

| Threading | Multiprocessing |
|---|---|
| I/O Bound | CPU Bound |
| Network(http, downloa, etc..)<br>File System(file read/wirte, etc..)<br>... | Calcucalculator<br>Image Processing<br>... |

Fig. 6: Use case for thread and process

*Performance and user experience.* Since the supported QoS, cost, and performance are different depending on the MQTT broker, We decided to use the Mosquitto broker after checking the performance comparison. Because our goal was to make the response speed possible within 1 second when 10000 users use the service at the same time. For this performance test, we tested the performance of the server by creating a virtual client code that simulates virtual requests. Although the collection of running data was slow, it did not affect the user experience much, but the communication delay that occurred when using the app had a significant impact on the user experience. Accordingly, we tried to improve performance by separating the running data collection thread, which collects a lot of data, and the user data collection thread.

*Data Base design.* We used two databases management system, MySQL and InfluxDB. For relational database, MySQL was used to simply store user data like fitness data and game related data. (Fig 7) And a time series database, InfluxDB was used to store 'Running data'. The 'Running data' has GPS coordinates and timestamps. The distance between the GPS points and the time interval between the points were needed to calculate the pace (speed) and save today's running route.

## Challenges

*Anti Cheat.[1]* Basically, it is known that all data sent by the client should be verified by the server based on the possibility that it has been tampered with. We thought that this game is a game with little user-to-user interaction, so using cheats in the game will have a relatively small negative impact on the user experience of other users. Therefore, the server performs simple verification only for important calls, and the rest of the operation is designed on the client side. After the server stores the theoretical maximum and minimum values per time interval, if the value sent by the user is out of this range, it is assumed to be cheat(Fig 8). For example, if data that runs faster than the 25km/h record of Usain Bolt(who is known as the fastest human being) comes in, the data is recognized as falsified data and is not accepted. Another example relates to the PLAY interaction, which can only happen once every 3 hours. You can increase

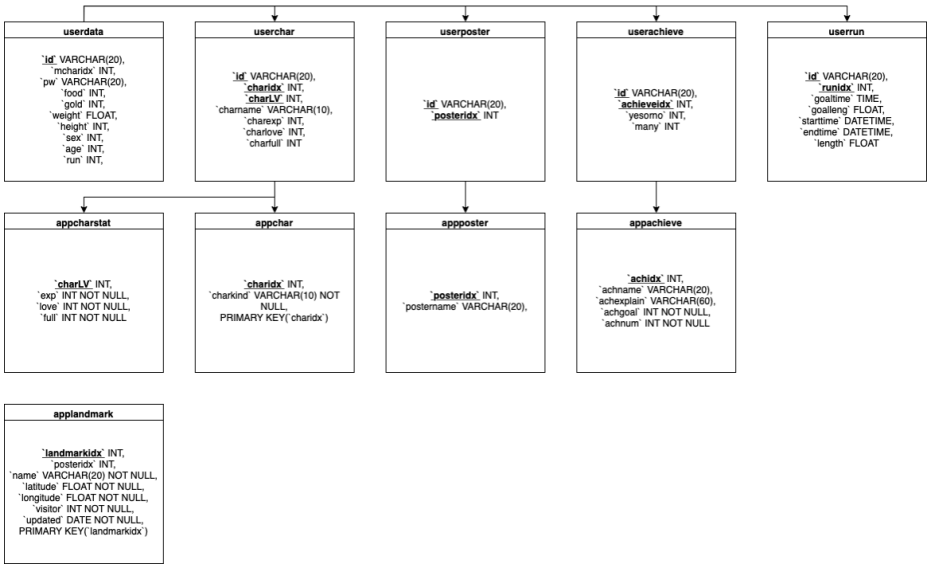



Fig. 7: data table

char.love and char.exp by executing PLAY an abnormal number of times by manipulating the local time of the device in the client. In order to prevent this behavior, when executing a limited action on the server on the client, the verification for this should be executed based on the centralized Server Time.

*QoS and server load.* Originally, when sending Running Data, data was transmitted through QoS(0), which allows loss at short intervals. This method showed the side effect that the MQTT Broker's performance degradation due to call overflow was more severe than the performance improvement through the lightness of the protocol of QoS(0) compared to QoS(1). To solve this problem, when sending GPS data, QoS(2), which guarantees data delivery through ack, was used, and instead of sending GPS data every time it was created, it was possible to improve the performance of the server by collecting and sending a certain number of data(Fig 9).

## 3 Implementation

### 3.1 Frontend

Implementing screens of each pages, we have considered the thumb zone. The navigation bar and run & game mode switch is at the bottom, which is natural area for most of the users. Also we tyied to not put as much buttons or interacting figures on the top of the screen.

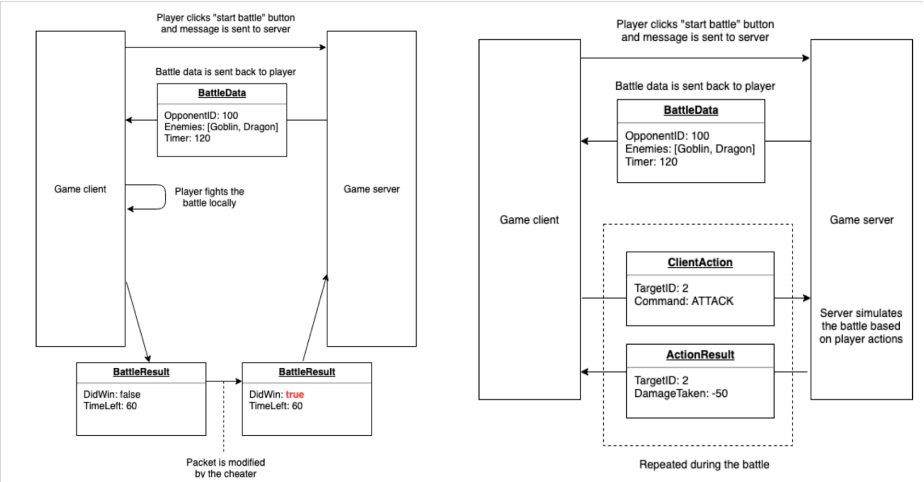



Fig. 8: Server Client model anti cheat example [1]

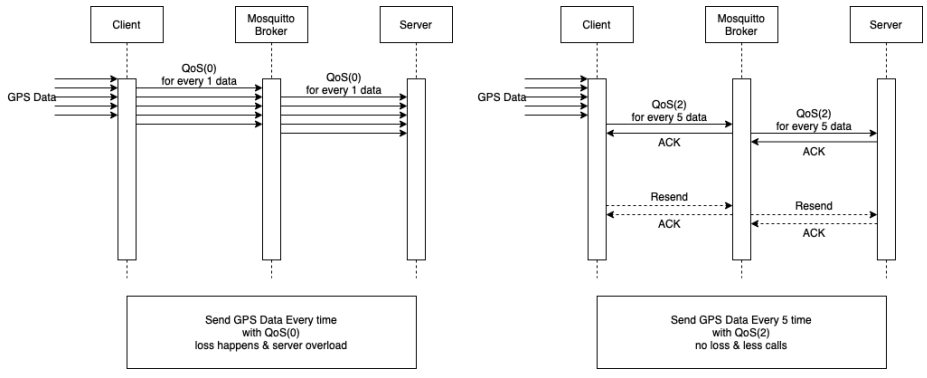



Fig. 9: server load with QoS

**ViewModel** This is a code we tried to implement viewmodel in kotlin(Fig 10).

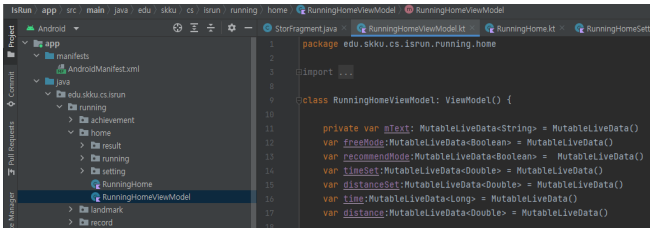
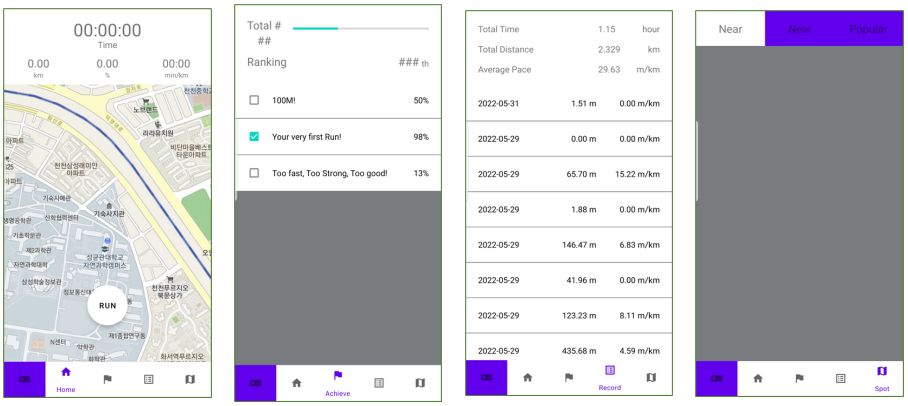



Fig. 10: ViewModel implemented with MutableLiveData[9]

**Running Page** 4 Running pages are implemented as follows.

1. Running Home(Fig 11a)
2. Running Achievements(Fig 11b)
3. Running Records(Fig 11c)
4. Location Recommendation(Fig 11d)

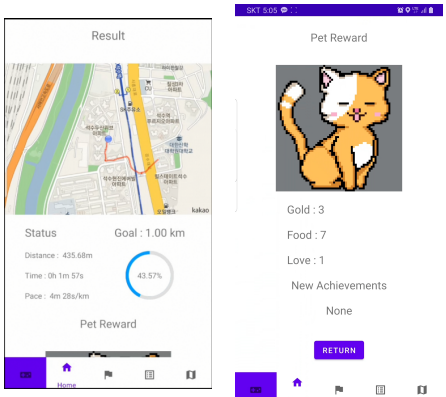



(a) Run Home (b) Achievement (c) Record (d) Location

Fig. 11: Running Pages

Below shows the result page with running route(Fig 12a) with rewards(Fig 12b).

**Gaming Page** 4 Gaming pages are implemented as follows.

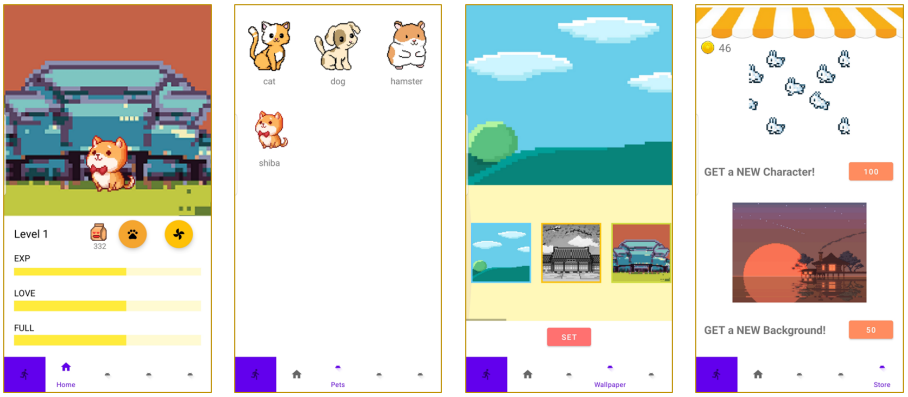



(a) Run Result with Route

(b) Pets

Fig. 12: Run Result with Rewards

1. Game Home(Fig 13a)
2. Character Selection(Fig 13b)
3. Poster Selection(Fig 13c)
4. Store(Fig 13d)




(a) Game Home (b) Characters (c) Posters (d) Store

Fig. 13: Gaming Pages

### 3.2 Backend

**Server implementation** The following description specifically describe how the server calls were implemented.

*Backend and frontend collaboration.* For the connection between the backend and the frontend, the following table(Fig 14) is used to organize which request to send to access specific data from the client. Certain data (mostly tables marked with app*) cannot be accessed with User rights and must be added with Admin rights to edit. This is because the data stored in the app* table is data used throughout the service.

| | | SELECT | UPDATE | INSERT |
|---|---|---|---|---|
| userdata | `id` VARCHAR(20), | SignIn | | SignUp |
| | `mcharidx`, `mposteridx` INT, | GetUserData | UpdateUserData | |
| | `pw` VARCHAR(20), | SignIn | | SignUp |
| | `food` INT, | GetUserData | UpdateUserData | |
| | `gold` INT, | GetUserData | UpdateUserData | |
| | `weight` FLOAT, | GetUserData | UpdateUserData | |
| | `height` INT, | GetUserData | UpdateUserData | |
| | `sex` INT, | GetUserData | UpdateUserData | |
| | `age` INT, | GetUserData | UpdateUserData | |
| | `run` INT, | GetUserData | UpdateUserData | |
| userchar | `id` VARCHAR(20), | GetUserChars | UpdateUserChar | NewUserChar |
| | `charidx` INT, | GetUserChars | UpdateUserChar | NewUserChar |
| | `charLV` INT, | GetUserChars | UpdateUserChar | NewUserChar(Default) |
| | `charname` VARCHAR(10), | GetUserChars | UpdateUserChar | |
| | `charexp` INT, | GetUserChars | UpdateUserChar | NewUserChar(Default) |
| | `charlove` INT, | GetUserChars | UpdateUserChar | NewUserChar(Default) |
| | `charfull` INT | GetUserChars | UpdateUserChar | NewUserChar(Default) |
| userposter | `id` VARCHAR(20), | GetUserPosters | | RunStart, RunEnd, NewPoster |
| | `posteridx` INT | GetUserPosters | | RunStart, RunEnd, NewPoster |
| userachieve | `id` VARCHAR(20), | GetUserAchives | RunEnd | SignUp |
| | `achieveidx` INT, | GetUserAchives | RunEnd | |
| | `yesorno` INT, | GetUserAchives | RunEnd | |
| | `many` INT | GetUserAchives | RunEnd | |
| userrun | `id` VARCHAR(20), | GetUserRuns | | RunStart |
| | `runidx` INT, | GetUserRuns | | RunStart |
| | `goaltime` TIME, | GetUserRuns | | RunStart |
| | `goalleng` FLOAT, | GetUserRuns | | RunStart |
| | `starttime` DATETIME, | GetUserRuns | | RunStart |
| | `endtime` DATETIME, | GetUserRuns | RunEnd | |
| | `length` FLOAT | GetUserRuns | RunEnd | |
| | | **SELECT** | **UPDATE** | **INSERT** |
| appcharstat | `charLV` INT, | GetUserChars | Admin | |
| | `exp` INT NOT NULL, | | | |
| | `love` INT NOT NULL, | | | |
| | `full` INT NOT NULL | | | |
| appchar | `charidx` INT, | GetUserChars | | |
| | `charkind` VARCHAR(10) NOT NULL, | | | |
| appposter | `posteridx` INT, | GetUserPosters | | |
| | `postername` VARCHAR(20), | | | |
| appachieve | `achidx` INT, | GetUserAchives | | |
| | `achname` VARCHAR(20), | | | |
| | `achexplain` VARCHAR(60), | | | |
| | `achgoal` INT NOT NULL, | | | |
| | `achnum` INT NOT NULL | | | |
| | | **SELECT** | **UPDATE** | **INSERT** |
| applandmark | `landmarkidx` INT, | RunStart, RunEnd, GetRecommar | RunStart, RunEnd | Admin |
| | `posteridx` INT, | | | |
| | `name` VARCHAR(20) NOT NULL, | | | |
| | `latitude` FLOAT NOT NULL, | | | |
| | `longitude` FLOAT NOT NULL, | | | |
| | `visitor` INT NOT NULL, | | | |
| | `updated` DATE NOT NULL, | | | |
| | | **SELECT** | **UPDATE** | **INSERT** |
| Running/UserId/RunIdx | `time` TIME, | GetRun | | RunningData |
| | Lat, FLOAT, | | | |
| | Lon, FLOAT | | | |

Fig. 14: Data table with access calls

*An MQTT server that conveniently implements mapping using the tree structure of topics.* In MQTT, you can use "/" to split parent-child structures between topics(Fig 15). This tree-structured topic helps the broker to map messages to subs more efficiently and quickly. Subs to too many topics or subs to topics that are not structured in a tree can cause the broker's performance to deteriorate. Therefore, it is good to group general topics used when a client connects to the server in the same superstructure.

```
# sub to UserData/#
def UserData_Thread():
    UserData_Client = mqtt.Client()
    # Default
    UserData_Client.on_connect = on_connect
    UserData_Client.on_disconnect = on_disconnect
    UserData_Client.on_subscribe = on_subscribe

    # SignIn Message Handler
    UserData_Client.message_callback_add("UserData/SignIn", SignIn_message)
    UserData_Client.message_callback_add("UserData/SignUp", SignUp_message)

    UserData_Client.message_callback_add("UserData/GetUserData", GetUserData_message)
    UserData_Client.message_callback_add("UserData/GetUserChars", GetUserChars_message)
    UserData_Client.message_callback_add("UserData/GetUserPosters", GetUserPosters_message)
    UserData_Client.message_callback_add("UserData/GetUserAchives", GetUserAchives_message)
    UserData_Client.message_callback_add("UserData/GetUserRuns", GetUserRuns_message)

    UserData_Client.message_callback_add("UserData/UpdateUserData", UpdateUserData_message)
    UserData_Client.message_callback_add("UserData/UpdateUserChar", UpdateUserChar_message)

    UserData_Client.message_callback_add("UserData/NewUserChar", NewUserChar_message)
    UserData_Client.message_callback_add("UserData/NewUserPoster", NewUserPoster_message)

    # HiveMQ를 사용합니다.
    UserData_Client.connect('ec2-52-79-242-94.ap-northeast-2.compute.amazonaws.com', 1883)
    # 해당 토픽에 구독해서 메시지를 확인합니다.
    UserData_Client.subscribe("UserData/#", 2)

    UserData_Client.loop_forever()
```

Fig. 15: Sub by "/" Topic tree Hierarchy.

*Receiving running data(GPS Data) from client.* Subscriptions to many topics can lead to degradation of MQTT broker performance, so when a client starts running, it starts a subscription to "RunningData/ClientID/RunIdx" and cancels the subscription when a RunEnd signal is received. Sub to topic is prevented(Fig 16).

*Avoid using global variables if possible.* Using global variables in multi-threaded situations can cause synchronization issues with racing situations. In the case of the current system that uses a database, the database automatically handles

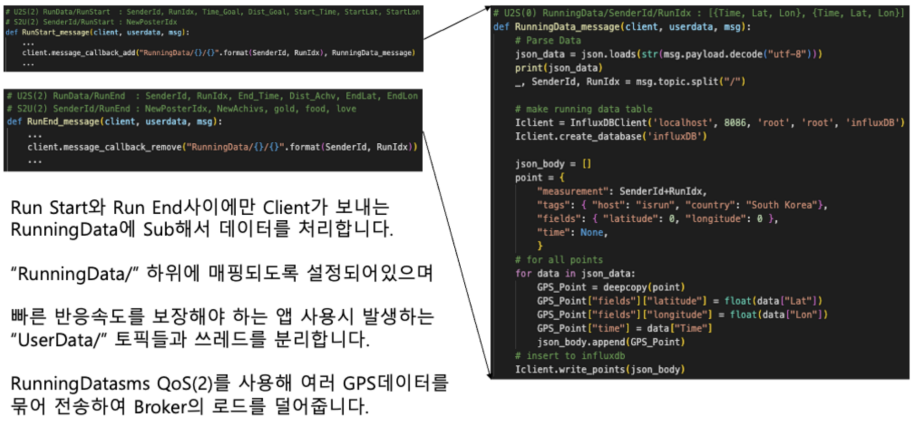



Fig. 16: How to receive Running data from client

these problems, so we refrained from using global variables as much as possible and called and used data from the database.

## 4 Limitation and Discussions

### 4.1 Frontend

**KaKao API** Although we managed to handle some problems caused by KaKao API[16], there are still unsolved problems. Most disturbing problem was that KaKao API doesn't work on some devices with android version over Android 12. So currently IsRun will properly work on devices under Android 12. In addition, as explained above, developers can't use multiple KaKao maps. As a result, it was hard to show run route records on screen in record pages.

**Gesture Sensitivity** The first plan was to change between running and gaming modes with swipe action, but the touch action sensitivity wasn't good enough. While doing swipe gesture, we needed to determine if the input gesture is swipe or not. Not only that, but also swipe animation needed more resource in main thread. Thus the mode changing method was implemented in button touch.

**GIF hardcoding** The character interaction was intended to be animated with gif image. However, the gif images could not be played with general image resource id because the default 'ImageView' provided by android doesn't support gif images. Instead, the gif image name should be hardcoded into the view in the source code level. As a result, the character interaction animations were all changed to just non-animation images.

### 4.2 Backend

**Trade-off between Anti-Cheat and Server Load** Anti-cheat[1] requires the server to verify the client's data. Such data validation causes an increase in the number of operations executed on the server and leads to degradation of the server's performance. If the server's performance is good, this load is not a problem, but if the server's performance is not good, it becomes a big problem.

For games where the client interacts with the client, such as the pvp element, the use of cheats directly ruins the user experience of other users. Therefore, in this case, the client sends only information and data, and after all operations are performed on the server, the result of the operation is sent back to the client.

But since our game is far from client to client interaction we planed to use "Time interval min max method". This method is not exactly validating each and every data from user. Still, it checks users data by each time interval and checks value increase is within possible range. This method is widely used in online rpg games for checking exp increase of character. However, in order to use this method, it is necessary to logically determine the minimum and maximum values. If it is difficult to establish such a clear standard, it is possible to closely examine whether cheats are being used by verifying every packet for clients that send data values out of a certain deviation through the normal distribution graph.

**MQTT Protocol security issues** The MQTT protocol[4] has many potential vulnerabilities. For example, open ports can be used to initiate denial of service (DDoS) attacks as well as buffer overflow attacks on networks and devices. Depending on the number of connected IoT devices and supported use cases, the complexity of the "Topic" structure can increase significantly and cause scalability issues. MQTT message payloads are binary encoded, and those application/device types must be able to interoperate. Another problem is that MQTT messages are sent in clear text with username and password.
Although not specifically part of the MQTT specification, it is common for IoT platform intermediaries to support client authentication with SSL/TLS client-side certificates. Encryption in transit using SSL and TLS can protect data when implemented correctly. Sensitive data such as user IDs, passwords, and other types of credentials should always be encrypted to protect against threats.

## 5 Related work

### 5.1 Prior Works

There were several works trying to connect athletics with games. However, their approaches were either only light enough for temporary interest or for heavy users. For instance, there are running apps like Nike Running App[**?**], whose

users are heavy runners, without game parts included. For cycling YAFIT Cycle is famous. But, YAFIT only tracks for cycles that works with particular device.

What about games? One of the famous athletic games is the Ring Fit Adventure[20]. Ring Fit is more of an rpg game. You work out to defeat some mosters in game and run through maps. Because it is still indoor activity, it is not suitable for running. Also it needs particular device called nintendo Switch to run this game.

Next example is Wii Sports[**?**]. Will Sports supports various athletics(e.g. boxing, bowling, and tennis) for multi-user, however running is not included. Moreover, Wii sports are considered as game for family or friends more than athletic effect. Meaning people won't keep interest for long. Last and least Wii sports also need it's own device nitendo wii.

### 5.2 Differentiation

Two big differences compared to former works. First, IsRun doesn't need special device to run with. We only need our phone to go out running and interact with our characters. Second, there were no application that connected running and game. And for applications which works with outdoor activity usually works with rpg type games. In contrast, IsRun's game type is virtual character simulation game, which links user's development with characters easily. As a result, we expect IsRun to be great application for beginners in running.

## 6 Conclusion

We have developed a running app 'IsRun', a running assistance application that motivates new runners to run daily. There are two modes, running and gaming. We combined virtual character game with running so that beginners who are just starting out can enjoy running. The results of running and becoming healthy is not visible, but the interaction and growth of the character is visible. With every day running, users can see their characters grow and check their daily running records. Since wearing a mask is no longer compulsory outdoors, let's go out to run, find a nearby landmark and collect posters and characters.

The frontend UI is implemented considering the interaction between two modes. Also, the thumb zone and MVVM design pattern were considered. Backend server is built with python and connection between frontend is confirmed by mqtt protocol. Although anti-cheat and security might be a little vulnerable, we've selected mqtt to give lower load to the server because 'IsRun' is not a pvp type game.

## References

1. Samuli, Lehtonen. "Comparative Study of Anti-cheat Methods in Video Games" University of Helsinki, Department of Computer Science, 7 march 2020.

2. Julie, Solon. "amazon-ec2-user-guide". Github, Awsdocs, 27 Jan 2022, https://github.com/awsdocs/amazon-ec2-user-guide/blob/master/doc_source/concepts.md. Last accessed 25 Mar 2022.
3. "InfluxDB 1.7 documentation". Influxdata, https://docs.influxdata.com/influxdb/v1.7/. Last accessed 27 Mar 2022.
4. "MQTT Version 5.0". OASIS, OASIS Standard, 7 Mar 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Last accessed 23 Mar 2022.
5. "발행-구독 모델". Wikipedia, 7 Feb 2022, https://ko.wikipedia.org/wiki/\%EB\%B0\%9C\%ED\%96\%89-\%EA\%B5\%AC\%EB\%8F\%85_\%EB\%AA\%A8\%EB\%8D\%B8. Last accessed 27 Mar 2022.
6. https://www.nuclino.com/articles/level-design
7. https://gamedevacademy.org/level-design-open-world-tutorial/
8. "The Thumb Zone: Designing For Mobile Users", 19, Sep 2016. https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/
9. "Android ViewModel & LiveData", 30, Sep 2019. https://megazonedsg.github.io/Android-ViewModel-and-LiveData/. Last accessed 25 Mar 2022.
10. "2019년 육군부사관 체력검정안내", 14, Jul 2019 https://blog.daum.net/goonmusic/144. Last accessed 25 Mar 2022.
11. "Android Thread", 5, Oct 2018. https://recipes4dev.tistory.com/143
12. "Fragment". Android developers, 22, Nov 2021. https://developer.android.com/guide/fragments. Last accessed 25 Mar 2022.
13. "Navigation". Android developers, 18, Apr 2022. https://developer.android.com/guide/navigation/navigation-getting-started. Last accessed 25 Mar 2022.
14. "BMI (body mass index) calculator". Running Training Plan, 18, Apr 2022. https://runningtrainingplan.com/bmi-calculator.php
15. "Fused Location Provider API". Google Developers, 3, Mar 2021. https://developers.google.com/location-context/fused-location-provider. Last accessed 25 Mar 2022.
16. "KaKao Maps API". KaKao Maps API, 22, Sep 2019. https://apis.map.kakao.com/android/guide/. Last accessed 25 Mar 2022.
17. "Glide". Sam Judd. https://github.com/bumptech/glide. Last accessed 5 Jun 2022.
18. Jessica, Jonsen. "Nike Run App Review". Fitrated, 11 Nov 2021,https://www.fitrated.com/gear/workout-apps/nike-run-club-app-review/. Last accessed 27 Mar 2022.
19. Casamassina, Matt. "Wii Sports Review". IGN, 14 Nov 2006, https://www.ign.com/articles/2006/11/14/wii-sports-review. Last accessed 23 Mar 2022.
20. Jenas, Sitzes. "Ring Fit Adventure Review: One Year Later". Gamespot, 18 Oct 2019, https://www.gamespot.com/reviews/ring-fit-adventure-review-nintendo-switch/1900-6417586/. Last accessed 23 Mar 2022.