POLYTECHNIC OF TURIN

Faculty of Engineering

Master's Degree

in Computer Engineering

Master Thesis

Android Wear: Usability Guidelines, Features and Development of a Prototype

Supervisors

Prof. Maurizio Morisio

.....

Candidate

Andres Camilo Jimenez Vargas

.....

October 2016

Index

Figure index

Table Index

1. Introduction

The Joint Open Lab (JOL), a research group of Telecom Italia (TIM), developed an application named Table Me for Android and IOS mobile devices, to manage a game of table foosball. The goal of the application is to trace a match between four people in a mobile device, accounting the result and the performance of the players. Moreover, after the user logged in, the application compiles all the information of the player's games, showing their profile information with the statistics, achievements and match history. Additionally, organizing their results in a leader board between all the players.

Thanks, to the functionalities of the application, it allows easily to follow and record a table foosball match and to share this information with the community. However, as the application is hosted locally in a single mobile device, it brings more complexity to the user, either if the user is being part of the teams involved in a match or not, must pay attention to the other player's movements and goals to count them correctly.

To be able to solve this problem, it is desirable to give the capability to the players to be involved in a game, in a more comfortable way without the necessity of having their mobile devices on hand all time. With the use of a smart watch, the user can have the commodity through the game, of having a functional instance of the game in a device paired with a mobile device on the wrist. Based on that, it is possible to give that functionality to the players taking away the dependency of using their phone constantly during the match. In addition, this will decouple the creation of the match in a single device, and make the match distributed with the devices involved in the match.

This project is focused on the development of an Android Mobile application and an Android Wear application, that allow the usage of the same features of the previous Table Me application in a smart watch. This will improve the players experience in the game, involving it to give its results during the match in a more precise and comfortable way, from a click away in his smart watch. Additionally, to access the user's basic profile data in the smart watch without the necessity of using the mobile device. To be able to develop this solution, it is necessary to develop a solution (that is integrated to the Table Me application), to manage the communication between the smart watch and the mobile device. Moreover, to create the match distributed between multiple devices, is necessary to develop a server able to handle the match and the communication to the devices.

From this point on, knowing the main objectives of the project, the it is necessary to include an additional item to develop to create the multiplayer environment. Due to that, is necessary to create a Server that follows the basic architecture of an interactive game online with multiple players. Based on the Table Me application where commands of goals in a match will be notified to the server, the implementation that approach the best solution is a Peer to Peer architecture with Lockstep used in online strategy games such as Star Craft, Age of Empires and Civilizations. This allows a real time gameplay with a non-heavy load server that just recognize commands from users and deliver them to the participants of the match without congesting their network resources. Additionally, the architecture recommends the usage of real time technologies to facilitate the communication of these commands. To solve this issue, it has been selected the usage of Web Sockets via the library Socket IO that supports multiple platforms for the communication including Android where the project was developed.

The developed solutions bring to the users the Table Me application for Android devices and smartwatches, supporting a multiplayer interactive environment where a user as a referee (being part of the match or not), can create a match and control the global score. The Android Wear application provide an easy to use interface (developed based on the lineaments proposed by Android), that allow to the users to interact with their own score (adding and dismissing goals or auto goals). This solution allows to control in a more accurate way the score of a match and follow it in real time by a referee that is hosting the game in the phone. Additionally, providing the previous functionalities that the Table Me application have in the smartwatch to consult the profile information, the leader board and the match history of a player.

1.1. Description of the problematic

The application Table Me can just manage a table foosball match locally in one device, restricting the players to be in touch with the game. In addition, this adds more difficulty to the match's host to keep track of each of the players moves, even if the host is a participant in the game or not. Moreover, all the data of the players and the matches are stored in a server hosted by the JOL, and all the player's information each time by application when the user open it and log in. Additionally, does not have an extension to the Android Wear technology, if any user has one.

Based on the previous statements, here are the most important points that comes from this problem:

- Difficult to use by the match's host (being a participant player or not), because require to have the mobile device on hand and keep track of the score of each four players during the game (limited user experience).
- The results of a match are followed by just the person using the application.
- Local dependency on a device to be able to keep track of the match.
- If a player has an Android Wear, he/she cannot use it with anything related to the Table Me App [2].

1.1.1. Formulation of the problem

Provided the problematic description, the project search to resolve the following question:

How to develop an Android Wear to help managing a table foosball match with four players, and provide the player's information in the Android Wear device, integrated with the Table Me mobile application?

1.1.2. Justification of the project

The development and completion of this project seeks to show how is constituted and how to use the recent technology of Android Wear with the smart watches. Based on that, the project presents a guideline of implementation of the communication with the smart watch, the design and implementation of the user interface. All this, to show how to solve the problems presented from the Table Me application.

Within the Android Wear application solution, the project search to give a user that has a smart watch, an instance of the application Table Me on it, to be able to use the main features with no need of mobile device use. Additionally, be part of a match (being the host or not), providing his results in the smart watch during the game. Because of that, will decouple the dependence of managing a match in a single device, and finally as well give the capability to a player (or players), of keeping track simultaneously of the game and their results in the smart watch.

To be able to develop this solution is necessary to create an Android Wear application, as well as two more components: An Android application to communicate the mobile with the smart watch, and a server to be able to host the game communicating all the players that participate in the match simultaneously.

Finally, the project seeks to develop a prototype that will be integrated with the already working Table Me application.

1.1.3. Expected Impact

The expected impact of the project with the documented guideline to the usage of the Android Wear technology is to help to facilitate the development of Wear applications (mainly the communication between the mobile and the smart watch), providing a guide to the developers that interested on the possible solutions that this implementation could take.

Additionally, with the prototype integration it will expect to give a more comfortable and easy to access way of using the Table Me application in a smart watch, facilitating the usage of the main features, without using the phone. Moreover, give a more interactive and easy way to the users to manage a game by the capability of a player of being part a match simultaneously with the other players that may or may not use the application in their smart watch. With this on future, is expected to give more functional features to the Table Me application to give to the users that may have an Android Wear device, opening the doors of the Android Wear market where the application can be used.

1.2. Description of the project

1.2.1. General Objective

Develop an Android Wear application, integrated with the Table Me mobile application, that implements its main features and offer to the user to be part of a distributed match with a smart watch, simultaneously with the other players of the match.

1.2.2. Specific Objectives

- Perform the characterization of the system by means of a process of requirement engineering, identifying the general requirements for the server, mobile and Android Wear applications, that allow to implement the main features of the Table Me application, and create a distributed match with other devices.
- Perform the design and architecture of the server, the mobile and the wearable application that allow to implement the main features of the Table Me application, and to be able to create a distributed match with other devices.
- Develop a prototype of the mobile and the wearable application that allow to implement the main features of the Table Me application, and to be able to create a distributed match with other devices.
- Integrate the developed prototype with the Table Me application, coupling its features with the already working one.
- Validate the integrated prototype in a testing environment of the domain of the Table Me users.

1.3. Expected deliverables

The deliverables proposed and planned to this project are the following:

- A software requirement specification document, that will contain:
    - Use cases model.
    - Definition of system's scope.
    - Functional and nonfunctional requirements.
    - Requirement prioritization.

- A functional prototype integrated to the Table Me application, that will contain:
  - User manual.
  - Description of the main features of the prototype.

1.4. Methodology

The methodology that will be used to develop the project will be based on the spiral lifecycle model from Barry W. Boehm [1]. This model is an incremental model that based on a risk analysis of each step of the project done frequently, will allow to prioritize the development of certain components of the project. The model proposes that in an iterative development of four activities the software will be develop in phases that accumulate prototypes or parts of it in the development of the system. The following figure illustrate the main structure of the spiral model.

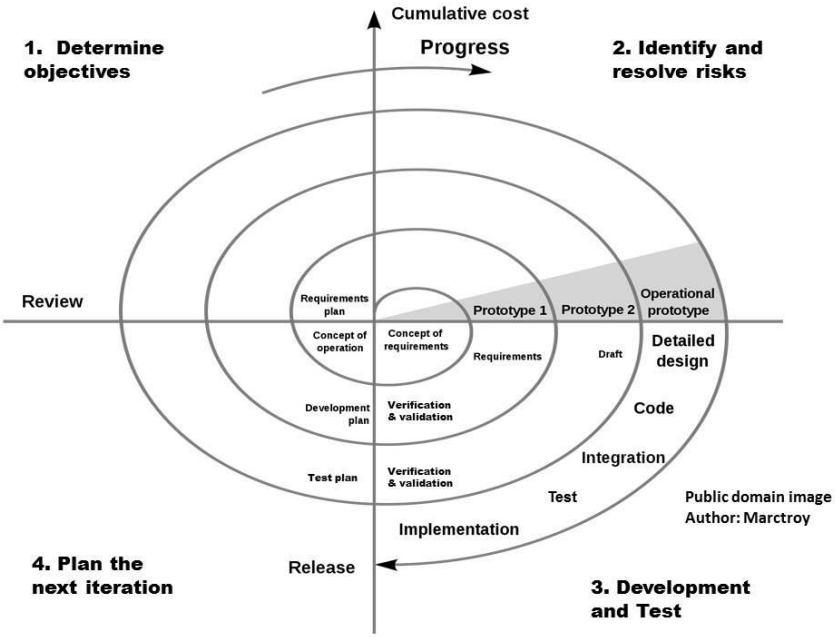


Figure 1: Spiral model from Barry W. Boehm [1]

The model is divided in four important phases. Firstly, determine objectives, where the requirements are collected from the user's necessities of the software and the development of an initial design. Then, comes the identification of risks, where based on the previous progress will be evaluated the risk that takes implementing the requirements capture in the iteration. After that, the development phase initiates based on the previous design and during the development multiple tests are done. Finally, the last iteration is for the user evaluation of the software testing it until release, during this phase new tests are added to be done in the following iterations.

For the purpose of this project the main four phases are described in the following:

- **Diagnostic**, research of the state of art and requirement collection based on the user's necessities and the research.
- **Design and architecture**, design of the software components and use cases to be implemented. During this phase a requirement prioritization will be done to

identify the most complex requirements of the iteration for the development of a use case.

- **Development**, implementation of the use cases in the JOL laboratory using the smartwatches to perform tests.
- **Testing and evaluation**, after implementing the use cases planted for each month, there will be a presentation of the prototype to the user showing the implemented features, taking in account the comments and possible changes.

2. Theoretical frame

2.1. Basis concepts about Table Me application

The application Table Me was design to trace a table foosball match, in a single mobile device. In the first instance, the application uses an authentication feature done through Facebook API or Google plus API, registering the user in the server, and retrieving the user information. After this authentication, the application shows a summary of the profile of the player and the functionality to create a match with a button.

Based on this introduction, the application's features are divided in the player's decisions of consulting in deep his profile information of starting a match. For this reason, here are presented the following features that the application offers.

2.1.1. Profile features

The application shows a dashboard screen divided in two sections:

- **Profile summary** [2], this section has the player's picture from Facebook or Google plus, where can expand by clicking on it, and this will show in detail the statistics, composed by:
    - Victories.
    - Losses.
    - Total goals and average goals per match.
    - Total auto goals.
    - Best position.
    - Best score.
    - Preferred team mate.
    - Nemesis team mate.

  After this section located on the left of the main view, there are three rows at the right side. The first row has the list of the latest achievements of the player; this row has a plus button to see in detail the list of all the achievements of the player in another screen. The second row has the ELO score of the player and his position on the leader board displayed in a badge, clicking the image of the position the application shows the full leader board in another screen.

  Finally, the third row has two bars that represent the number of victories and the number of lost matches. Each of the bars grows at the contrary of the other.

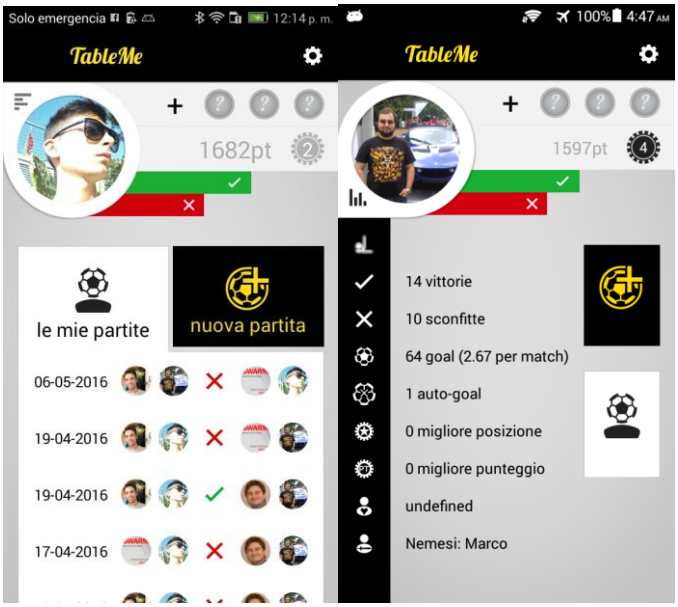


Figure 2: Profile screens of Table Me

- **Match history** [2], in the rest of the screen below the profile section as in the Figure 2, there is a list of the matches where the player participated. Each row represents a match, including the date and the images of the profiles of each player. Finally, in the middle, it shows a green check if the player won that match, or a red cross if not. By clicking the row with a match, the application will show another screen with the complete information of the match in a screen similar to the screen on game, showing the results of each team. This screen contains the total goals scored by each team, the goals that each player did individually and the auto goals too. The team that lost selected the match, will have a badge under the team's total score that says "rivincita" to play a revenge game, by pressing this the user will start a new game with the same players of the match.

Figure 3: Singular Match History Table Me

- **Main leader board** [2], the user can enter to this screen by tapping the position badge, this will show the general leader board. Moreover, if the user tap over one of the players in the leader board, then the application will show the correspondent

profile information of the selected player. The profile information displayed have the same attributes as the user's profile section.

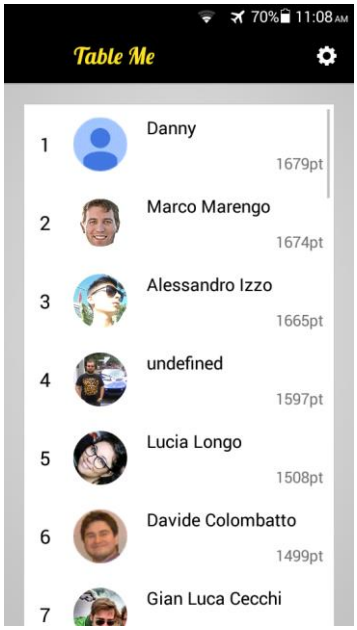


Figure 4: Main leader Board Table Me

2.1.2. In game features

- **Player selection** [2], after clicking on create a new match, the application displays a new screen with a layout similar to a foosball course. On each side are two buttons to add the participants of each team, by clicking on them it displays a list with the players of the game, and the user is able to select a player to all the four positions. After that, the application enables a button to start the game.

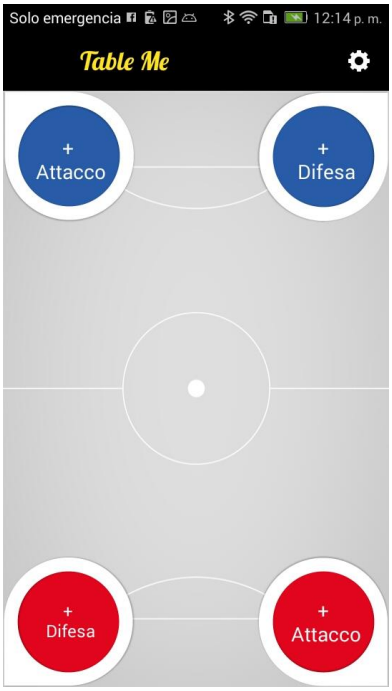


Figure 5: Player Selection Table Me.

- **In game** [2], after clicking the start game button, the application displays a layout similar to a foosball course and the players of each team, identified by their

photos, over their respective sides. Over the sides of each of the player are three buttons, the first for scoring a goal adding a point to the team who score it, second to score an auto goal, and third to dismiss a goal. The application allows to keep the score of a match until a team reach six points, and when the match ends it display the results in a similar way to the match history screen.

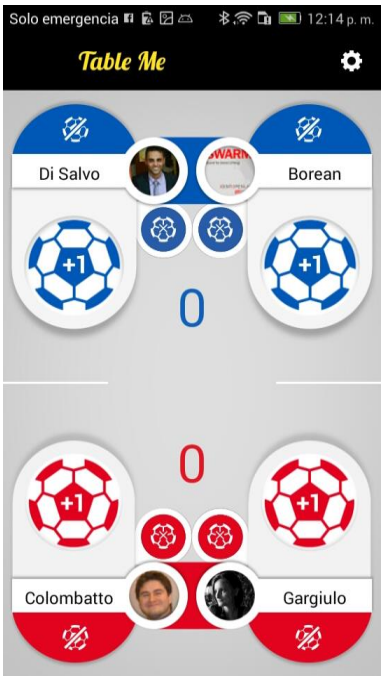


Figure 6: In Game Screen Table Me

- **End game** [2], after one of the team's scored six goals, the application asks to confirm if the score is correct or not. Selecting the incorrect option, the last goal will be dismissed and the game continue. In the other hand, by selecting the correct score, the application will change the background color with the color of the winner team with an animation of the score. Finally, a pop up window is displayed showing the final score, and giving two options to the user: play a revenge game or go back to the dashboard.

Figure 7: End Game Table Me

- **Revenge game**, after the user select to play a revenge game as shown in Figure 7, the application shows the match all set up with the participants of the previous match or the match selected from the history, enabling the start button to start the game.

2.2. Conceptual frame

The conceptual frame of the project is target to the following important aspects to develop the distributed game of foosball for the Table Me application. Firstly, is necessary to know the main and important ways of communication between the mobile devices and their correspondent smart watch. Secondly, an architecture for multiplayer games that allow supporting the games correctly and its communication methodology. Finally, the user interface to be implemented in the smartwatch, due that its reduced size, it is required to know the guidelines to design and implement it.

2.2.1. Android Wear

The smartwatches that uses Android Wear are very recent, the first one was released the 25th June of 2014, being able to develop applications in the same manner as with a mobile device. This release was using the android distribution Kit Kat 4.4.1 now a day the last release is the version 1.4 launched 4th February 2016 using the distribution Lollipop 5.0 giving multiple features in addition with the previous updates such as Wi fi connection directly from the watch, Bluetooth complementation, longer battery life management and support to send multimedia data from the phone to the smartwatch.

The main features of Android Wear are that in the home page of the device it manages a stream of cards displayed vertically in a Context Stream [55]that are stacked in a prioritize way with the information sent from the phone. The stream cards allow to display notifications and applications features that simplify the usage of the application in the smartwatch and in the mobile device.

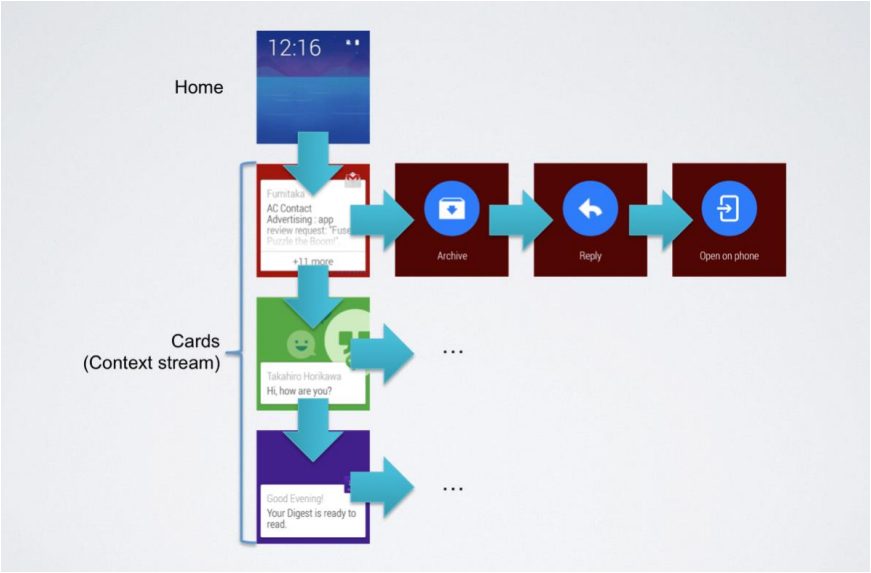


Figure 8: Android Wear Context Stream[3]

The context stream allows to access to applications features or notification information by swiping to left displaying concatenated cards, and it can be dismissing by swiping to the right.

Additionally, to this, Android Wear allow to create applications that use the Context Stream to place the activities of the application. As said before the advantage of this technology is the usage of the same implementation as in a mobile device development, this allow an easy management of the application's code.

In the following points there will be a description to the different ways to communicate the smartwatch with the mobile device using the Message API [9], the Data API [13] and the Channel API [32].

2.3.1. Communication between devices

In the communication between devices (mobile and smartwatch) is relevant to note that it is necessary to take care of the compatibility between devices to be able to use them correctly. Android offer the versioning of their releases for the mobile device that allow to using different services depending of the device and the android distribution that it use, in the same way the smartwatches that use Android Wear. About this, the technologies developed for the smartwatches have evolved from the simple pair connection with a single device through to Bluetooth, to a direct Wi Fi connection and have multiple smartwatches connected to a single mobile device. The following architecture shows the normal usage of this technology.

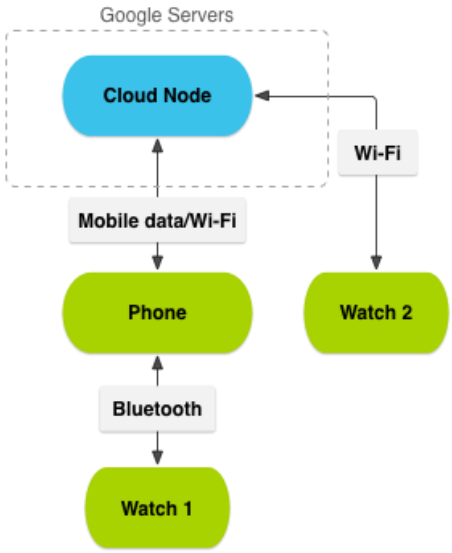


Figure 9: Physical Architecture of multiple devices[4].

Due to this, to be able to be compatible to the smartwatches that use only the Bluetooth connection, it is necessary to know the communication methods between the mobile and the smartwatch using the Wearable package[5]. This package implements all the different ways to transfer and synchronize data between devices.

Additionally, is necessary to know a trustable way to communicate between mobile devices using the Google Cloud Messaging service to notify asynchronously about events.

Moreover, this implementation helps to the development of a simple base of a multiplayer architecture to be able to support a distributed game for the Table Me application.

2.3.1.1. Notifications using Google Cloud Messaging

The notifications are one-way asynchronous communication to send messages to a mobile device to another. This can be done using Google Cloud Messaging service, that allow to push notifications between devices sending a JSON message in an HTTP request to the service with the information of the receiver and the message. To be able to use the service, every device that use the application must register itself in the Google Cloud Messaging service, and the result of that will obtain a token that will identify the device and through it will be able to send and receive messages [6].

However, a device cannot know all the other devices that will send an asynchronous message, as such in a game environment. For this it is needed a server that store this information and help a device to easily send the message to the desired mobile. The documentation of Google Cloud Messaging [6] advises to use this architectural implementation in the following mode:

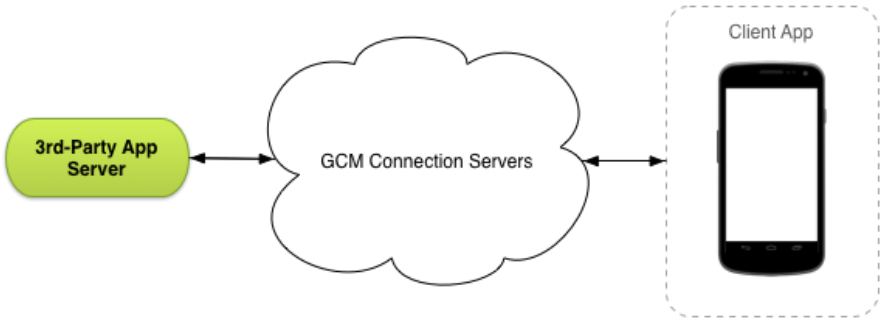


Figure 10: GCM Architecture[7]

Following the advice given by the source, the third party app server will be in charge of store the information of all the clients that use the application saving the information of each one for further requests. After this point, the server can act as a database of user's tokens so the client application can send the notification by itself, or the server can be more active and manage to send the notifications by itself by demand of the users.

After a message is sent using the Google Cloud Messaging service, the client uses a listener service already implemented by Google (the GCM listener service [39]), that is instantiated every time the device receive a message from Google Cloud. From that point on, the device is in charge of raising the visual alert of the notification setting its visual properties (e.g. icon of the notification, background, text, title, actions with pending intents to open activities or services, etc.). It raises and display the notification drawer on top of the user interface, and when the user slides it down it shows the full content. If it is clicked it will execute the action with the pending intent that was set before, and will execute the activity or service attached in the intent [7], [8] .

In the case of when the device paired to a smart watch, by default all the notifications raised from the phone will be also in the smart watch. However, either there is an option to raise the notification locally, so each device either the mobile or the smartwatch display it. In the smartwatch the notifications are raised in a stack of cards that is managed by the operative systems, where all notifications are organized by priority. the cards that display

a notification have multiple ways of use allowing to set the same properties as in the mobile device [8]. Using the custom layout of a notification, the user can create a custom view with a certain style of the card and its text; additionally, it is possible to add an extra page that can be access by sliding to the left to put more information.

Finally, to open equally an action as in the mobile device, the notifications in the smartwatch manage actions in an additional page per pending intent, and can be access by the same way as the additional pages of the notification. A set of dots in the inferior part of the interface below the notification represented the quantity of pages of the notification.

2.3.1.2. Message API

The messages are one-way communications that are suitable in short life applications of components of it. this method of communication was designed mainly for remote procedure calls (RPC [9]), to invoke functionalities between devices by sending a message. A Data Layer managed the data to send and it is in charge of sending and receiving the messages across the Bluetooth connection between devices. Additionally, the Data Layer is in charge of raising the services that are waiting for a response through this channel.

Moreover, the service is offered by the Google Play Services for Wearable devices, by using the API implemented by Message API [5] inside the Wearable package [5], that is able to send and receive messages between the mobile and the smartwatch, and used to send data in byte format identified with a path id.

There is two main ways to use this way of communication. The first one, used just to receive information, is done by using a Wearable Listener Service [10] that is activated by the Data Layer when any kind of data event that is sent between devices (Device connection, receiving messages, communication errors), will be notified in a specific method. To capture messages sent by the Message API the service uses the implementation on Message ()[10]. With this the data will arrive in a message event that contains the bytes of the message, its identifier path, the id of the request of the message and the source node that sent the data. Additionally, the service handled all the connections automatically.

The second way of communication is guided to be implemented by using a Google API Client applying the Wear API [5] that acts as a channel that allows to send data and attach listeners to receive the data through the Message API listener implementations[9]. Additionally, the listeners give more information to control the actions to take. Because of this, they allow to know when the devices (either a mobile or a smartwatch) are connected or disconnected or when the communication between the devices suffer of an abeyance. Finally, and the most important aspect is that the connection can be controlled in the activity or service that will interact with the devices by the usage of the Google API Client [11], using the same implementation with the method on Message() who connects and disconnect the channel it's needed programmatically. It is important to note that, to be able to send a message using this implementation, it is necessary to send it in a different thread with the user interface or in the same thread where a service is running.

The following architecture shows how a message is send from the mobile device to the smartwatch through the methods necessary to send a message as explained before.

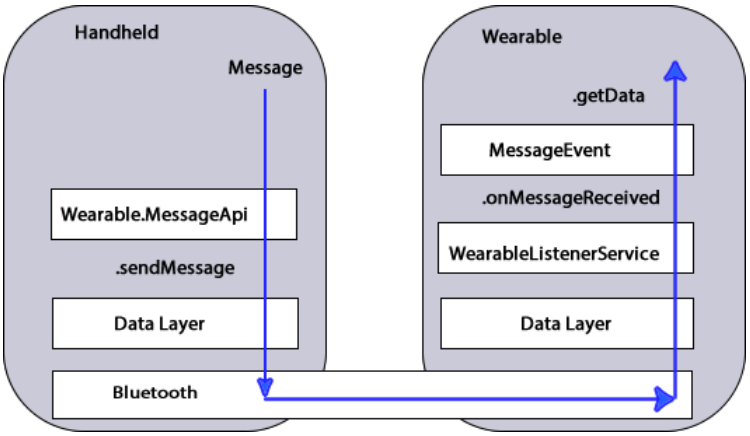


Figure 11: Message API Architecture of the implementation[12]

With the latest releases of the Google Play Services, it is possible to pair via Bluetooth multiple smartwatches with one mobile device, and there are identified as a node, with a unique identifier. The nodes can be retrieved by using a node listener attached to a Google API Client[11], and it will allow to have control over the devices when are connected or disconnected to the mobile device and vice versa.

2.3.1.3. Data API

The Data API services [13] provided by the Wearable API [5], is a one-way communication between devices paired via a Bluetooth connection, and was designed to be used as a data synchronization method. As well as the Message API is used for RPC (however does not allow to send more than a set of bytes); the Data API allows to send objects in a Data Map. This act as a hash map storing any object with a string identifier, allowing sending more heavy loaded data between devices without the necessity of serializing the data. A Data Map allows storing the simple data types. Additionally, it can use a bundle of data from an intent, and for images, audio or video, can be used Assets that allow to store and serialize this kind of multimedia data[14].

The Data API access to the data layer, not as directly as the Message API. The data must be stored in a request that is sent to the Data Layer, settled in a priority queue that is organized depending of the urgency of the data to be sent (Between more urgency the Data Map is sent without delay, the default configuration can take over 30 second to send a Data Map[15]). Then, automatically sends the data to all the devices connected through an instance of the Google API Client implementing the Wear API[5]. This can be a disadvantage if a user use multiple smartwatches connected to the mobile.

The usage of the implementations of the Data API is used in the same way that the Message API. One implementation using a Wearable Listener Service [10] and the other one controlling the connection using a Google API Client implementing the Wear API [5], attaching to it a Data API Listener that implements the method on Data Changed ()[10].

For sending the data, it is necessary to use a Data Map as said before, store the information into it and put it into a request, this request allow to set its urgency. Then, this

request is sent to the Data Layer through the Data API implementation in the Wearable package[5]. Moreover, to be able to send the request, it must be done a thread different form the thread where is running the user interface or the service.

After this, for receiving the Data Map sent, can be use the Wearable Listener Service implemented by the Wearable API using the method onDataChanged() [10]. This method works as a data synchronization method, receiving the Data Map issued by a device. However, it is necessary to note that this listener will be activated only when the data sent is not the same as it was received before, denying duplicated data.

The following architecture shows how a Data Map is sent from a mobile device to a smartwatch using the implementations explained before.

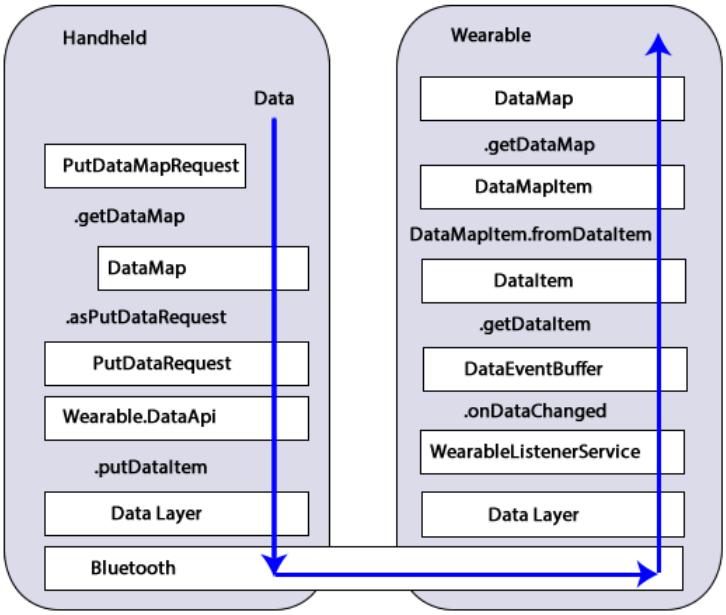


Figure 12: Data API Architecture of the implementation[15]

2.3.1.4. Channel API

The Channel API is a service provided by the Wearable package[5], that send large amounts of data, streaming it directly to an specific device. The Channel API create a bidirectional connection with a node sending objects such as with a Data Map [15] and also use Assets for multimedia data [14]. In this case, the Channel API is different to the Data API due to that Data API uses data synchronization and it does not send repeated information. In the other side, the Channel API sends the data independently if is redundant or not. Additionally, it relays on the Message API to send in case of a heavy loaded file and the Channel API streams the file through this way. Moreover, it facilitates the streaming of data to and from the network, such as multimedia data. Finally, the Channel API creates a copy of all the data received in local memory of each device. This can be a disadvantage if a smartwatch or a mobile device does not have enough memory to support copies of large files.

The implementation of the Channel API uses a streaming method to send the data through a Google API Client implementing the Wear API [5], and it creates an input and an output stream that is open until it is closed by the Google API Client. As before, the

implementation can be done using a Wearable Listener Service, but implementing the methods on Channel Opened(), Channel Closed() when the channel is created and onInputClosed() and onOutputClosed() [10], when the stream of data is closed by the sender. Otherwise, controlling the connection with the Google API Client, attaching a listener of Channel API implementing the same methods explained before.

2.3.2. Architecture for a multiplayer game

The architecture of a distributed game helps to use and organize the server correctly. In the communication between the mobile devices and the game's hosting server, is necessary to use a technology that support the game without long delay response, and the way how the data in the server and in the client is managed.

2.3.2.1. Peer to Peer

This architecture of a network where every device connected exchange information between them apart if someone is a server or a client [56]. Additionally, for Real Time Strategic games this architecture uses a deterministic lockstep that helps to simulate a player behavior and to synchronize the communication between devices only sending individual commands such as clicks or points over a game.

However, the architecture with the lockstep was thought only of local network games, this could bring a high latency to the players. Firstly, due to the connection status, and finally cause of the simulation of the game.

Moreover, the architecture allows to a component in the middle to run a soft controlled instance of the input based on the previous state sent. This soft instance can be managed with the games rules whenever a command triggers an event or is not permitted. The component will notify the participants of the game only the commands form the other players; this will allow a faster way of respond to results. Thanks to this, the architecture allows to create real time communication between the devices, allowing to decouple the dependency of a stable network connection.

The following figure illustrate the basic architecture model based on the Napster sharing network design:

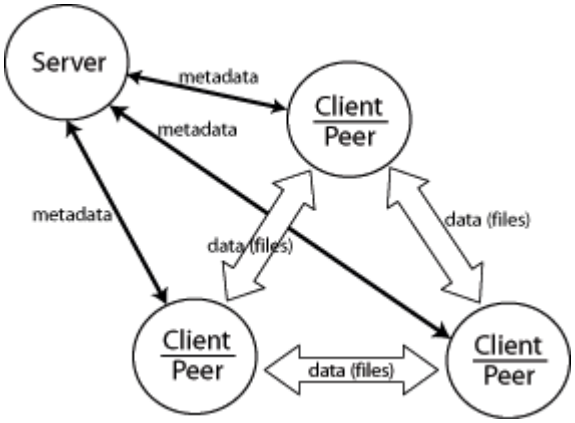


Figure 13: Peer to Peer architecture from Napster[16]

The architecture purpose is to use a light server component that allow to the clients to communicate in a peer to peer way helping to synchronize the lighter data as an intermediate cooperator between them.

2.3.2.1. Client/Server

The Client/Server architecture for multiplayer videogames is based in a soft client and a heavy load server, where every client that is playing does not execute locally in the client, but a soft instance of the game is paired with the information of the server. With this, the client must work in order to update with the server's multiple responses done from other players. This allow to have a great performance in the client side, but it depends on the state of the connection of the players with the server, for this case this can bring a lot of latency in to the game, due that the clients will receive the update until the slowest client send information to the player.

The implementation of a distributed architecture for interactive multiplayer games by Ashwin R. Bharambe, Jeff Pang and Srinivasan Seshan[17], gives an explanation of the development of a First Person Shooter named Colyseus. In their research, the authors propose an architecture for the nodes involved in the game that help to organize the data of the game and the interaction between the nodes. This can help to develop the architecture of the server of the game and to adapt the client structure to the structure developed in the Table Me application.

The following architecture (Fig. 5) shows the architecture of the nodes in the Colyseus game. This can be used as a guide of the main elements for the server and for the client. In first instance, this architecture is based on the Model View Controller (MVC) [57], with the game application as the view, the local object store as the model and the object placer and locator as the controller.

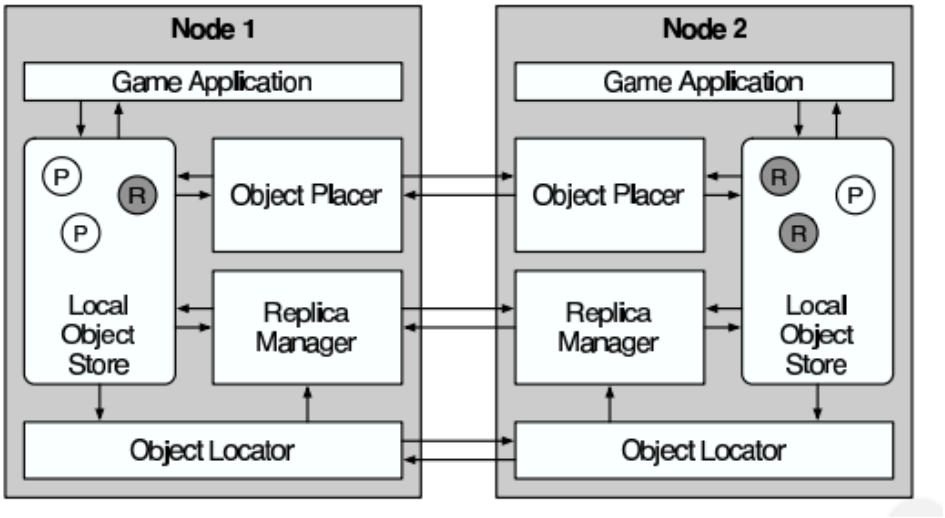


Figure 14: Architecture of components of Colyseus, (Figure 4[17, p8])

In the architecture referred, the nodes represent a set of important elements to have in the client and in the server. In this case, the node 1 acts as a client and the node 2 as the server. The elements are structured by the following way:

- As said before, the game application act as a view for the client, using the model to represent the game engine or application, displaying the information of the game stored in the model.

- The game application act as part of the controller in the server, using and administrating the primary data of the game.
- The object locator helps to a node to discover the objects that are involved in a primary interest. In the case of the Table Me application a new game, challenging a set of players.
- The replica manager is in charge of have a loosely-synchronization[17] of the replicas of the nodes and objects involved in the primary interest. In the case of the Table Me application, it would be managing an instance of the replica of a node involved in a game in an instance created on run time.
- Object Store, where is managed the storage of the node's information and additionally releasing memory of data not needed.

2.3.3. Communication technologies

For a multiplayer game is necessary to have short time response communications. For this reason, it would be necessary to use real time communication to be able to ensure a minimal time of response between Client and the Server. In addition, it is necessary to guide the solution to implement a real time system.

A real time system responds to stimuli of an environment in an established time, as well as ensuring the integrity of the data transmitted. Among real time systems, we find different categories: the hard real time systems are those where response times must be always respected, because can they can be involved in life threatening environments that need a fast answer. Firm real time systems can miss some of the deadline times, but it does not affect the environment where the system is. However, the performance will eventually degrade if too many responses are missed, leading to degrading the system as well: so different time limits must be set regularly, such as seismic or temperature sensors.

With a real time, system, which also is distributed, the communication becomes more complex, because now it needs to consider the network infrastructure upon which the system is running, and (as the data is sent) is assured that the communication is still in real time.

To address the above, it is necessary to check the real time protocols: these protocols allow us to transmit data within a stipulated time. There exist various implementations, suitable for the different scenarios, each of them has advantages and disadvantages. Likewise, many different technologies help us to achieve the above. The most important one of these are Long Polling and Web sockets.

With respect to these technologies, we know that these three are used to achieve the improvement needed based on a web application or a library on a mobile application.

2.3.3.1. Long Polling

The Long Polling technology is based on the usage for single purposed software for low level hardware communications. This technology simulates a real time communication through continued use of the http requests to a server using AJAX for web development of

interactive applications. To achieve this, the client establishes the connection with the server asking if there is any update. Then the server replies with the updates (if any) [41] [33]. Upon receive the response the client establishes the connection again and requests next possible updates wait for any new information from the server. All the connections managed by the server are stored in a data base where the server needs to query the client's connection each time a new request is received. This base protocol has the disadvantage of high consumption of resources processing and I/O operations between the data base and the clients, that must be used whether or not there are updates for clients. In the Long polling implementation, when the server receives a request it waits some time to send the response, although, the disadvantage is that it uses network resources constantly. Additionally, if multiple clients are connected, the server will have a high load of work just checking the state of the clients taking too long to answer. The following figure illustrate the behavior of the Long Polling technology.

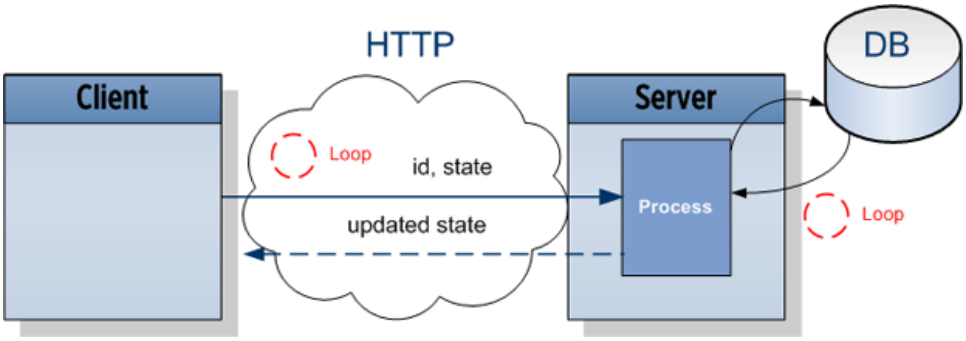


Figure 15: Long Polling [58]

To be able to manage the technology in android, it is necessary to use it with normal HTTP request. However, the management of the response time must be done programmatically to be able to repeat the responses in case of failure.

2.3.3.1.1. AJAX Long polling

The Asynchronous JavaScript and XML (AJAX) technique is used for applications that are highly interactive with multiple users. This technology operates by only sending additional data that trigger different actions in the client, without sending multimedia content. This method of communication uses the long polling thanks that keeping the connection alive in background, updating the user interface whenever a new state is received from the server [49]. The technology uses XML HTTP Requests as standard to communicate between clients, so it can be used in different platforms that use Java as minimum.

AJAX can be use with long polling where, as said before, the client keeps the connection to wait for the server response, when the client receives an update it immediately asks for another long poll request to keep the connection. The following figure illustrate the AJAX Long Polling behavior.

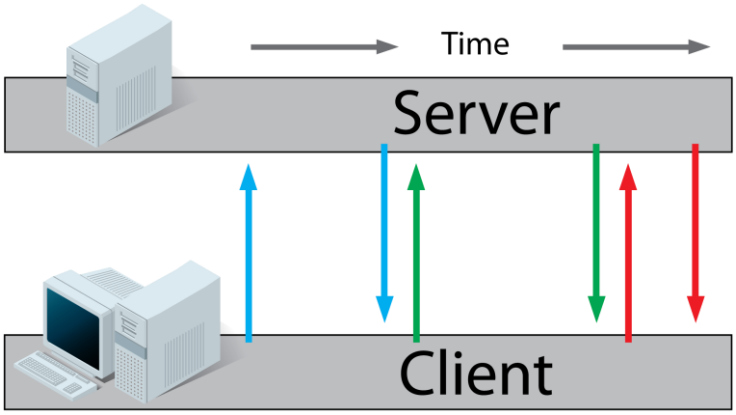


Figure 16: AJAX Long Polling behavior[19]

However, the drawbacks of this mode of use is the constant use of resources as said before, and the client needs to initiate always the communication to be able to receive and send data (not event based). Additionally, the server needs to be sending data often, or the clients could have a memory overflow with the channel opened all the time consuming memory and processing.

2.3.3.2. Web sockets

Web sockets are the technology that closely resembles the real time communication, without the resource consumption that the Long polling uses. The web sockets open a bidirectional channel between the server and the client on a single TCP connection, with that the server will send when necessary any update to the client without overloading the network of multiple requests. In first hand, this technology was used for browser use only, however with the latest development on this technology it is able to use in any type of server and client across multiple platforms, offering not only a great performance, but security and scalability attributes for the developing. In this manner this technology allows to implement massive platforms such as multiplayer interactive videogames.

The following figure show the behave of the Web Sockets technology.

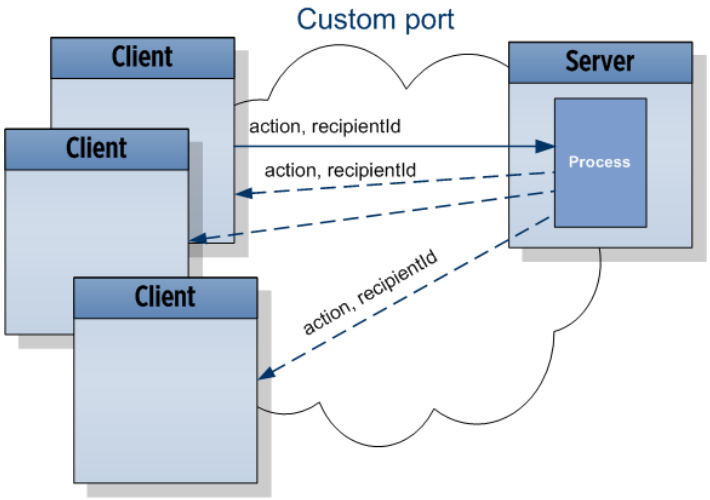


Figure 17: Web Socket Architecture [58]

This technology offers a wide portfolio of solutions that easy up the communications through different quality attributes. In the security area, the Web Sockets can be used in encrypted connections between points, allowing secure communication using a TLS connection. This implementation is named Web Socket Secure. In the reliability part, the

technology offers a transparent connection, that use automatic reconnection to the clients and a trustable message sender that is aware to send the right messages to the clients in the correct time. In top of this, to improve the usage of memory the Web Socket server manage an array of the connected nodes in the application's memory, allowing to reduce I/O operations, differently than as it is used in Long Polling.

As said before, the web sockets are developed multiplatform, and there are multiple implementations that can be used in different environments, then the best solutions for a multiplatform system are the Web Socket Node, Stream Socket and Socket IO. However, the Web Socket Node just use an implementation for Node JS client or server, but not support for multiplatform. In the other hand, Socket IO allow to be used in the mobile development environments for Android and IOS, and in various languages for the standalone clients or servers in using JavaScript. In addition, Stream Socket can be used by sending normal HTTP requests

Eric Terpstar had develop a prototype of a multiplayer game using web sockets, implemented in a library named Socket IO [48]. This implementation is done in node JS and express JS allowing to play the game through a browser, either from a phone a smart tv or a computer. The game is called anagrammatix, and is a real time system for a multiplayer game, where the players compete in a short time to find an anagram word in a list of words. The players must find an anagram in the list based on an initial word with the same letters given at the start of the game. This game shows a simple and complete implementation of how to use Web Sockets in a library able to support the technology handling the main features for the communication letting the user have an easy to use implementation.

2.3.3.2.1 Socket IO

Socket IO is a Java Script framework cross platform that allow real time communication, and open source. Since with the 1.4.5 release available it allows to deal with compatibilities between browsers and devices [48]. Additionally, Socket IO manage a library for android where can be used as easy as in the Java Script implementation. This library communicates through events associated to a socket that are executed in a new process each time a new event is received. This allows to handle concurrency issues more easily. To the scalability of the server, Socket IO does not store and replicate data about the connected clients, it just manages the events that are received and associate them with the right nodes. This part is managed by the emitters that accept the connections and are in charge of send and receive the right messages to the clients and the server. The multiple implementation of the emitters in different environments, allow to an easy integration for a cross platform application.

The implementation of the Socket IO uses a single TCP connection that is kept alive using a small heartbeat message that allow to know through events when a client is connected, reconnected or disconnected that allows in a more assertive way to know when a connection is terminated. Additionally, to this, Socket IO, have a more refined implementation of interconnection of clients independently the network error due to the disconnection.

Socket IO have been managing an upgrade/fallback algorithm in the latest releases that supports the type of connection that is needed depending of the client's compatibility. If the user has compatibility problems to use Web Sockets, Socket IO uses the algorithm, in the other case, the connection is totally done through Web Sockets. In the initial

connection a Long Polling implementation is used, applying the improved memory management approached explained before. After a connection is completed, the algorithm tries to upgrade the connection to a Web Socket connection, keeping the channel alive and closing the Long Polling session. If the upgrade is not possible the communication is done through Long Polling behaving as the Web Sockets.

2.3.3.3. Technology benchmark

Finally, to be able to select correctly Socket IO for the development of the Back End of this project, there are the following benchmarks done by Cubrid Apps & Tools, where they explain and show the best solution to use for a server using real time communication. This first benchmark shows quantity of messages sent in a certain rate of microseconds, it used a server using AJAX that a similar implementation as with Long Polling with a persistent data base of connections, a server using Socket IO and a server using Socket IO but using persistent data base for the connections. These are the results:

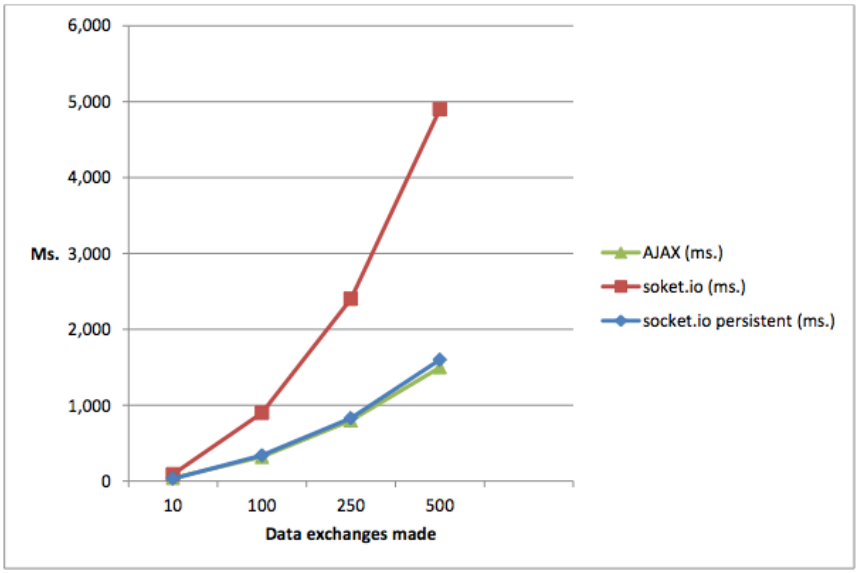


Figure 18: Speed Benchmark for AJAX persistent server, Socket IO server and Socket IO persistent server [49]

This shows that for small traffic the three implementations can be used in the same manner, however when the load increases the Socket IO implementation improves due to the lack of I/O operations with any connections database.

Now knowing the performance in the speed sending messages, now is necessary to know the capability for having concurrent clients using a server that is very relevant for this project to be able to host multiple matches without losing performance. The following benchmark done by Drew Harry, who perform a set of test showing the concurrency capacity of the Socket IO implementation, receiving and sending messages with different loads, in a certain rate of concurrent connection from 25 to 1000 connections, within a roundtrip time. These are the following results.

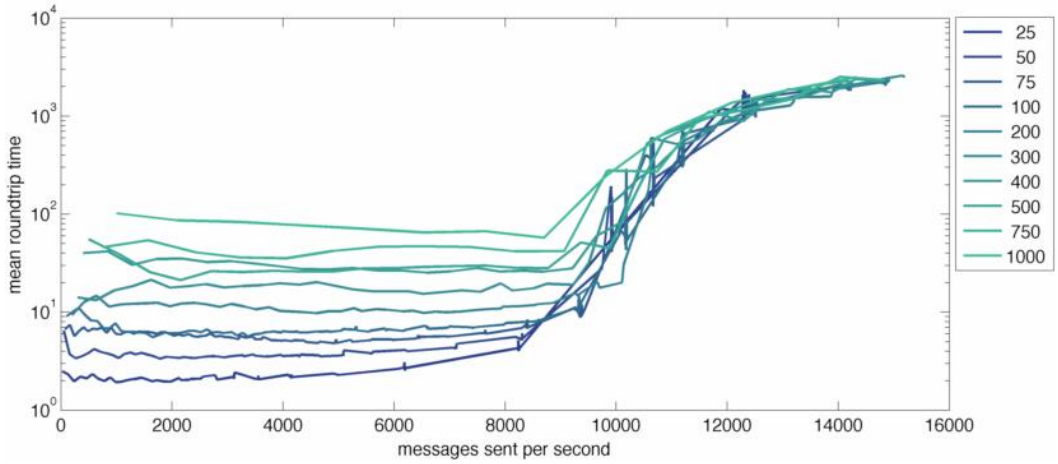


Figure 19: Concurrent Benchmark on Socket IO server, number of messages sent by roundtrip time in different concurrent rates[20]

The main focus of this benchmark shows that by the higher concurrency level, the load takes a longer around trip but it is not related with the message load, however, the response times maintain the same distribution regardless of the load of the data sent, but it depends on the concurrency level. Showing that this time difference between the higher concurrent cases adds 100 ms that is manageable and not very costly for the application.

Finally, in conclusion the Socket IO solution would help to support a multiplayer game, with a low delay of sending messages and a high support of multiple connections with different load of data sent and received from the clients. Additionally, the Socket IO implements multiple features that complement a cross platform solution with backwards compatibility support in real time communication, reliable connections and security features.

2.3.4. Graphic User Interface Design for Android Wear

For the development in the smartwatches using Android wear, google have developed a set of design principles for the user interface design and performance in the following way:

- "Focus on not stopping the user and all else will follow" [21], this explains that as a smartwatch it has to be a device that allow to the user to do different tasks without stopping him/her from the normal flow of actions.
- "Design for big gestures" [21] or big thumb principle design, that explain that is necessary to be careful about the user interaction in different situations where the interface has to be proportionate and easy to use. For this, Google advise to use large interactive targets such as buttons or lists so the user can use easily.
- "Think about stream cards first" [21] As said before in 2.2.1. Android Wear, the smart watch uses a steam cards that shows information and give features from applications of the mobile device. For this is necessary to know when the design of the application must use cards to show events from applications, cloud services or sensors.
- "Do one thing, really fast" [21] An application will be used for small periods of time, but is used multiple times a day, the application to be developed must show

potential but short information with few action buttons, allowing the user to use the basic actions such as swipe left or right.

- "Design for the corner of the eye" [21] The application must not pull out the user of his/her normal flow, the application must be design to be used quickly so the user can return to do what it was doing.
- "Don't be a constant shoulder tapper" [21] Do not constantly notify the user using vibration actions to alert him. Notify him when is necessary.

This applies to single applications and stream card applications. Additionally, Google use it for design principles for notifications in the stream card of the watch, for this the notification can manage a style that is customizable that allow to use different actions in concatenated cards using intents for a specific activity or to trigger a next card concatenation.

3. Analysis

3.1. Proposed model

The proposed solution of the system using the user's smartwatches to involve features of the Table Me application will require existing components and new ones for its development. Due to the previous implementation of the Table Me application, the JOL (Joint Open Lab) manage a server in which all the system data is stored and request through HTTP. Thanks to the server implementation, the solution counts with the user information such as profile data, match history and leader board as it was used before. To complete the solution, it would be necessary to complement the server with an additional component that will be in charge of hosting the multiplayer platform. For this, is necessary to note that is needed an internet connection to be able to use the previous server such as the new one.

The following model show the principal elements needed for the solution.

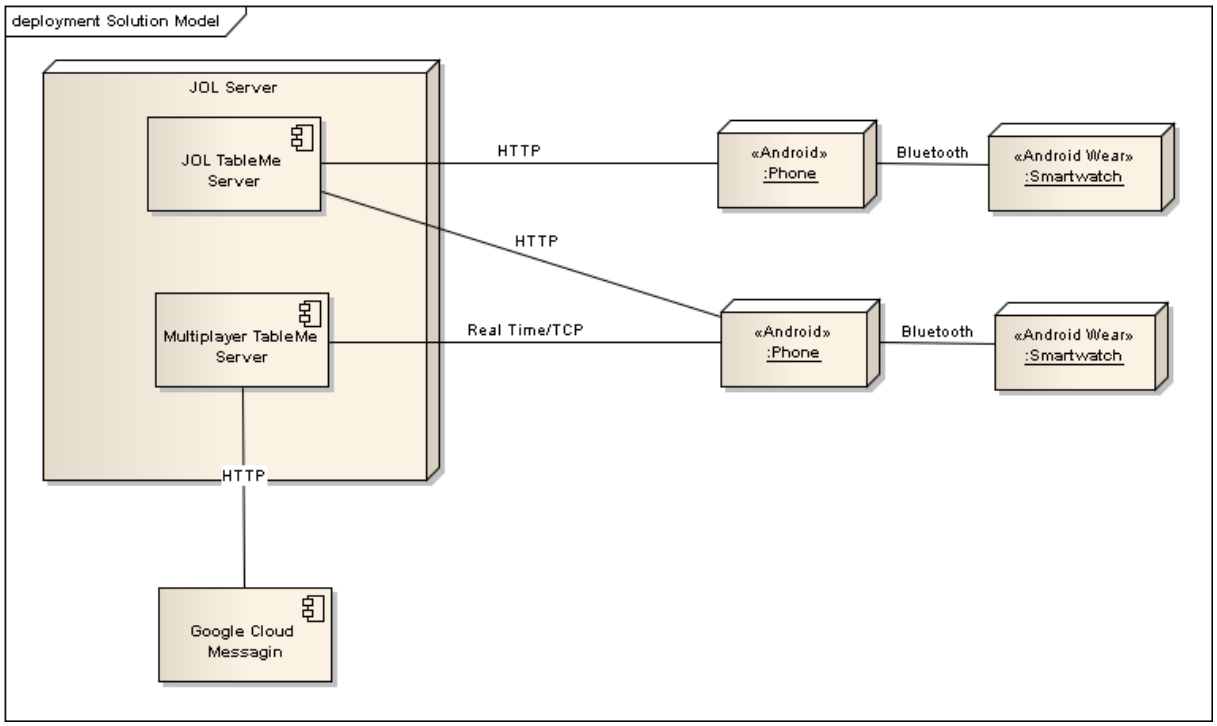


Figure 20: Proposed Model

Moreover, the multiplayer component will require the usage of the Google Cloud Messaging to push notification to the devices to alert a user when is asked to participate or not in a game.

In addition, the communication between the smartwatches and the mobile devices will be through Bluetooth connection, this to support the previous smartwatches that only have this channel to connect with the phone.

3.1.1. Distributed gameplay

For the implementation of the distributed gameplay it will use a Peer to Peer architecture approach, based on the lockstep addition to this architecture, instead of using it for the

prediction of the users moves, it will be used to control a match allowing a user that use the application in the mobile device to host the game and communicate to the participant players. Additionally, it is taken in account the architectural implementation for managing the game's data done in the game Colyseus shown in the Figure 14.

The final system will use a real time technology for the communication with the client's mobile devices, and will require the usage of the services from Google Cloud Messaging to push notification to trigger the events in the clients to be able to start or join a distributed game. It is important to note that it will be necessary an internet connection to be able to use this feature, if not the user still would be able to host a game in a single mobile device as the previous implementation.

For the gameplay experience, the user that create a game has total control of the match as a referee, in his/her mobile device. Meanwhile, the players with smartwatches that join the game, can manage their own scores from the Android Wear device. However, if the referee doesn't allow a goal it can dismiss it from the phone. Finally, at the end of the game, if the match creator scores the winning goal and decided to accept it, the players will be notified and the game will end. In case that a player scores the winning goal from the smartwatch, it will have the sale behavior as before and all the players and the match creator will be notified of the winner.

3.1.2. User information management

The management of the user information is done through the server hosted by the JOL (Joint Open Lab), and all the communication is done by a component named Communication Manager, used in the Table Me application. For the implementation of this section it is necessary to integrate and reuse the functionalities of the Communication Manager when the user request it from the smartwatch. For this is necessary to use the Wearable Listener Service [10] to receive the events from the smartwatch and return the correct information of the user.

3.2. Software prototype

Through the design phase and the correspondent evaluation, it had been able to recollect and represent the use cases and its respective requirements to describe the prototype's features and behaviors to be develop. To develop this, it had been done a Software Requirement Specification, that show how was the process to define the use cases, the respective requirements and the prioritization of them to identify their relevance to give a precedence order to the most complex cases to implement. The document is in the following annex: Software Requirement Specification.

The next sections will show the analysis and design of the integrated prototype of the Table Me application before being programmed.

3.2.1. Use cases

The uses cases were developed using a process defined in a Software Requirement Specification document (Annex: Software Requirement Specification) where had been organized how will the use cases and the requirement derived from them were obtained and refined to create the final product.

The recollection of the use cases and the requirements was done based on the description of the game given by Marco Marengo, Cecchi Gian Luca and Alessandro Izzo knowledgeable people that through a series of reunions had explain application functionality and the desired features to be implemented with the Android Wear technology. Additionally, the usage of the actual working application in the testing version on the Google Play Store. From this was developed a description of the main features and their principal elements.

3.2.1.1. Actors

The stake holders involved in the system are the following:

Actor name	Description
Player	A user of the Table Me application that is challenged to play a game. This user has a smartwatch and the Table Me application installed on it. This player is involved as a client in the interactive multiplayer game managing his score in the game.
Match Creator	A user of the Table Me application that create and host a game, managing the team's scores from the mobile device. The host as a player too, is able to use the smartwatch if participate in the game, as well using the application in the phone (noting that in the smartwatch will be only the personal score).
GCM	Google Cloud Messaging a service from Google that provide push notifications feature to the server and the player's mobile devices.
JOL Table Me Server	The server of the previous implementation of the Table Me application where all the data is stored and requested of the users and their games.

Table 1:Actors

3.2.1.2. Use case description

After the recollection and refinement of the user's information and necessities, the use cases were classified in the following categories depending on the phase that are developed:

Phase	Description
Menu options	Requirements about displaying the profile information, match history and leader board.
Game preparation	Requirements about the process to prepare a game and notify the players that will be involved in the game.
Connection	Requirements about the connection between the server and the mobile client, and the mobile client and the smartwatch.

In game	Requirements about a current match with or without multiple players, hosted by a user of the Table Me application.
Game end	Requirements about the finishing phases of a game.

Table 2: Phases of the Table Me application

Based on the previous information the use cases stated for the implementation of the system are the following:

ID	Use case name	Phase
UC-001	Synchronize wear data	Initialization
UC-002	Register device to the server	Connection
UC-003	Create a match	Connection
UC-004	Send notification to a device	Game preparation
UC-005	Select red team	Game preparation
UC-006	Select blue team	Game preparation
UC-007	Use GCM service for a token	Connection
UC-008	Receive challenge notification	Game preparation
UC-009	Show challenge notification	Game preparation
UC-010	Accept challenge	In Game
UC-011	Start match	Game preparation
UC-015	Add goal	In game
UC-018	Update score	In game
UC-019	Display score	In game
UC-020	Select winner	Game end
UC-021	Display winner or loser	Game end
UC-024	Show profile data	Menu options
UC-025	Show leader board	Menu options
UC-026	Show matches history	Menu options
UC-027	Notify player ready	Game preparation
UC-028	Register player ready	In game
UC-030	Register player decline	In game
UC-031	Create match	Game preparation
UC-032	Cancel match	Game preparation
UC-035	Reconnect wear	Connection
UC-036	Reconnect player	Connection
UC-037	Add auto goal	In Game
UC-038	Request user information	Menu options
UC-039	Request leader board	Menu options
UC-040	Request match history	Menu options

Table 3: Use cases

- The use case diagram is in the following diagram: (Annex: Use case diagram)
- The total description of each use case is in the following file: (Annex: Use case specification).

3.2.2. Requirements description

After defining the use cases, the requirements for the development of the scenarios planned where collected and evaluated to check their correctness and priority. This section has been done in the Software Requirement Specification (Annex: Software Requirement Specification), where is a specification of each requirement and its dependencies.

From that, it was produced the requirement classification based in the FURPS+ model [22] to organize their types. The main two groups are the functional requirements and nonfunctional requirements, the correspondent specification of each one is done in the following files:

- Functional requirements: (Annex: Functional requirement specification)
- Nonfunctional requirements: (Annex: Nonfunctional requirement specification)

3.2.3. Data model

The system already manages a data model that uses the server and the Table Me application to share information. The relevant issue is how to use this model to represent the data that will be sent to the smartwatch when the user requests his profile information, his match history and the general leader board. Based on the previous model, it has been design the following data model that will be used to organize and structure the responses from the server to send the correspondent information to the smartwatch.

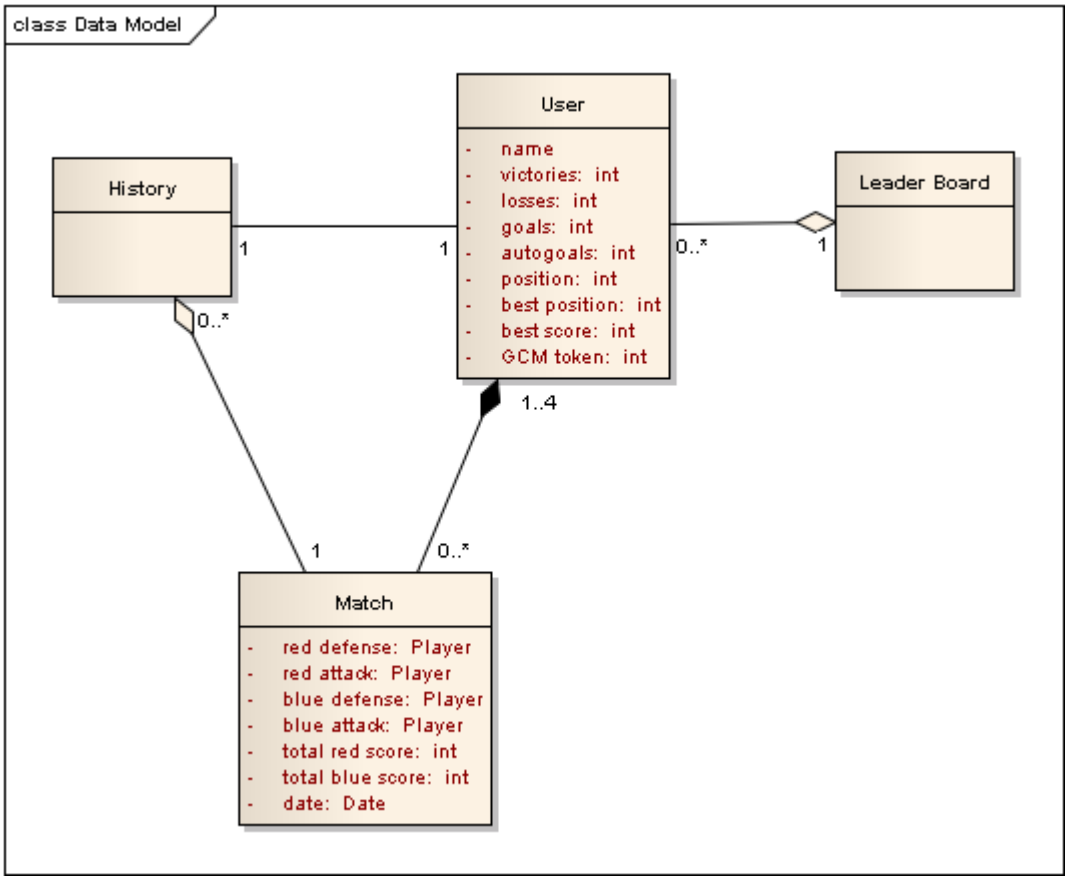


Figure 21: Data Model

3.2.4. Programming language for the server side

For the implementation of this project's server, it had been decided to be developed in Node JS for the easy use of the Java Script language that allow thanks to the callback based concurrency [49], the event based model used in server side and the ability to encode and decode data in JSON format. Additionally, the technology of Socket IO [33] and Stream Socket can be managed in this environment. Moreover, it will use the library of Socket IO due that its multiplatform facilities with libraries for Android and Java Script. Thanks to the benchmarks between the Long Polling technology using AJAX and the Socket IO implementation, it has brought to a clear window of which technology can give to the architecture of the system a better performance through the gameplay experience.

3.2.5. Architecture

The logical architecture of the system follows the structure of the proposed model shown in the Figure 20. Based on that, the deployment of the system will be done in the following way shown in the physical architecture:

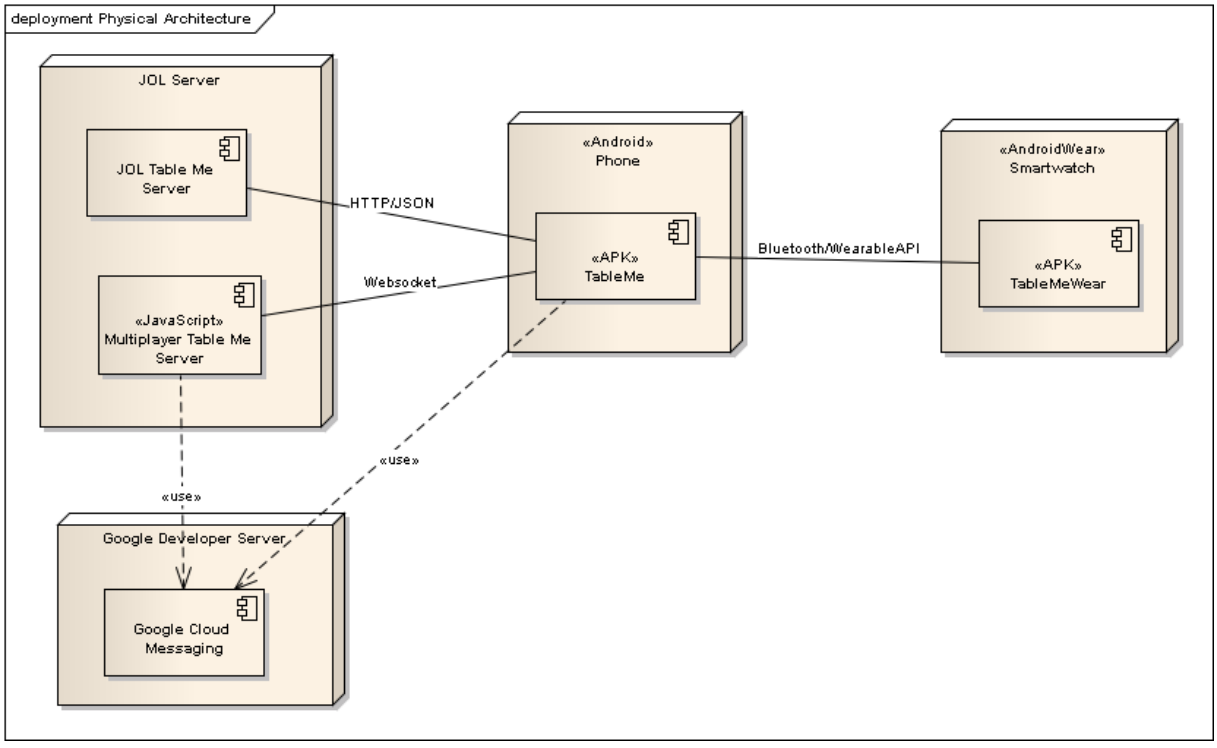


Figure 22: Physical Architecture

The multiplayer Table Me Server will be deployed in Node JS. The deployment for the mobile devices will be in an Android Application Package (APK) for the phones and for the smartwatches.

Concerning the communication of the system it is important to remark that as said before, the usage of the previous implementation of the Table Me application and the server developed for it will be used, all the communication is managed by a Communication

Manager via HTTP request to the JOL Table Me Server, this is used for the storage of permanent data such as matches results and retrieving profile information of any player.

For the multiplayer Table Me server the communication will be done using web sockets with the implementation of Socket IO using the libraries for Node JS in the server side and Android for the client side. More over the usage of the Google Cloud Messaging service to push notifications will be managed only by the server due that is the component that knows all the players registered in the system, then the Table Me application will receive the notification using the library provided in Google Play Services to receive a notification as explained in 2.3.1.1. Notifications using Google Cloud Messaging.

Finally, the communication that will be used between the phone and the smartwatch will be through the Bluetooth connection using the Wearable API for the communication. The usage of this library is due that allow an effective communication for remote procedure calls using the Message API, for functionalities in the mobile device; and for the synchronization of heavy loads data with the Data API. The Channel API won't be use due that can generate memory overflow due that reserve a part of the smartwatch memory to save a copy of the data sent from the phone, and is dedicatedly mostly for the transition of files.

3.2.6. Features and technical aspects

The deployment and installation of the server will require for recommendation the Node JS environment from version 4.4.4 onwards. After the installation it is required the following libraries:

- **Socket IO**, for installation is necessary to type the following command "npm install socket.io".
- **Google Cloud Messaging,** for installation is necessary to type the following command "npm install node-gcm"

For the deployment of and installation of the libraries for the Table Me application in Android and Android Wear are done in the gradle build of the application adding the following dependencies:

- **Wearable API**, compile 'com.google.android.gms:play-services-wearable:8.3.0'
- **Socket IO**, for this library the application contains the version 0.7.0 for Android that is compatible with the version 1.0 onwards of the JavaScript implementation. This library was developed by Naoyuki Kanezawa, if a new version is needed can be downloaded from Naoyuki repository:
    - https://github.com/socketio/socket.io-client-java

  And would require the following dependency in the gradle build, where "x" is the version of the library:
    - compile('io.socket:socket.io-client:x) {
      exclude group: 'org.json', module: 'json'
      }

Finally, for the interfaces developed for the smartwatch application has two layouts for the different types of watches such as the square or the round ones. For this, it has been used

watch stub views that allow to set which layout will be displayed if is used in a device with a round or square screen.

3.2.6.1. Profile information on the smartwatch

After opening the application, the user can use a list of option to select about the user's profile information and the game. The profile option shows in a similar way as in the Table Me application the basic information of the user. Firstly, the photo, the position in the leader board inside a badge, two bars that indicate proportionally the number of victories (green bar) and number of lost games (red bar). The at the end the name of the user and his/her ELO score.

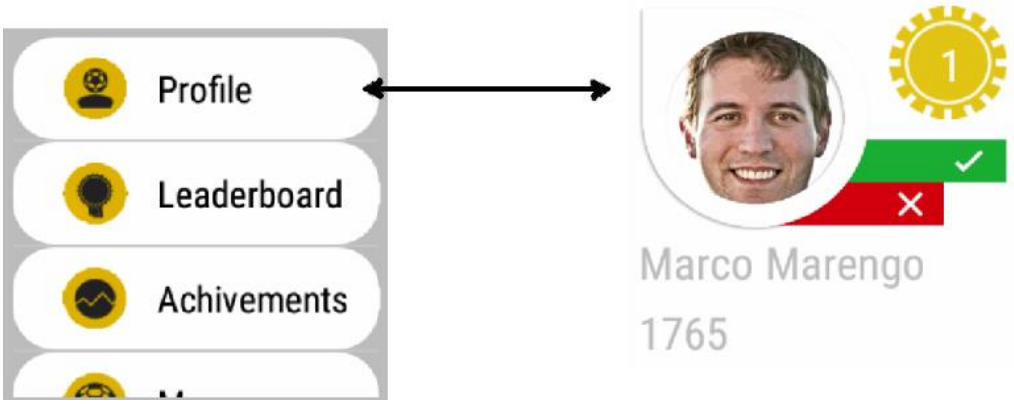


Figure 23: Showa player profile in the smartwatch screens

3.2.6.2. Gameplay on the smartwatch

After a user that will host a game organize and create the team, a notification is issued to the participants of the match will receive a notification in their phones and to their smartwatches if they have one. At the watch the following notification is displayed.

Figure 24: Notification displayed in the smartwatch for the players.

Then, when the user start playing the game, the screen with the buttons where the player can interact with the score is provided, with a button to add a goal in the center, a dismiss goal button at the left bottom and an auto goal button at the right bottom. The user can interact with them and the score is displayed in the results displayed in top of the screen with its respective colors of the team.

Figure 25: In game screens for the blue team and the read team, and the screens for victory and lose.

Finally, in each case the users of a team win or lose a match the following screens are displayed in the smartwatches using a base badge with their goals scored.

3.2.6.3. Match history on the smartwatch

When the match history option is selected, the user is provided with a brief list of the last ten matches that he/she was part of. Each match is signaled with a cross and the message "Lost!" if the player lost that game, or a check and the message "Won!" if the player won the game. If the player wants to know a more detailed information of the match, it is possible to click an item on the list and will provide a screen with the background color of the victorious team, the score, and the photo of the participants. If the user wants to see the players profile information, he/she can click on the image and the profile information will be displayed.

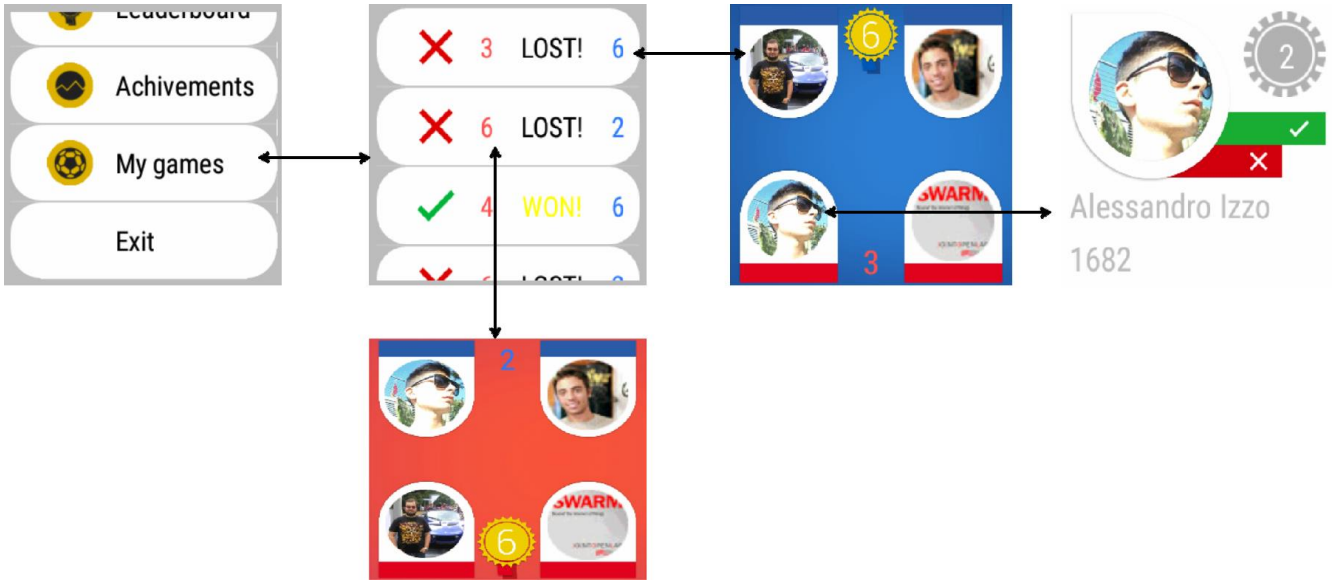


Figure 26: Showa player's match history in the smartwatch screens

3.2.6.4. Leader board on the smartwatch

When the leader board is selected a list of the ten first places in the leaderboard is loaded and take between 3 to 4 seconds. After waiting, the list displays the basic information of the player in the displayed position with his photo, name and position in a badge. If the user wants to see the profile of the player in a certain position, he/she can select a player in the list and it will show the basic profile information of the player.

Figure 27: Showa player leader board in the smartwatch screens

4. Development of the Solutions

The solution modeled was based on the previous Table Me application, that was studied to identify the main items that the new items will interact and the new functionalities in the already implemented in the first application.

The following domain model (Annex: Domain Model) illustrate the whole solution for the mobile application, the smartwatch application and the server. The next sections will describe the individual implementation and behavior of the solutions on each device.

4.1. Server

4.1.1. Class diagram

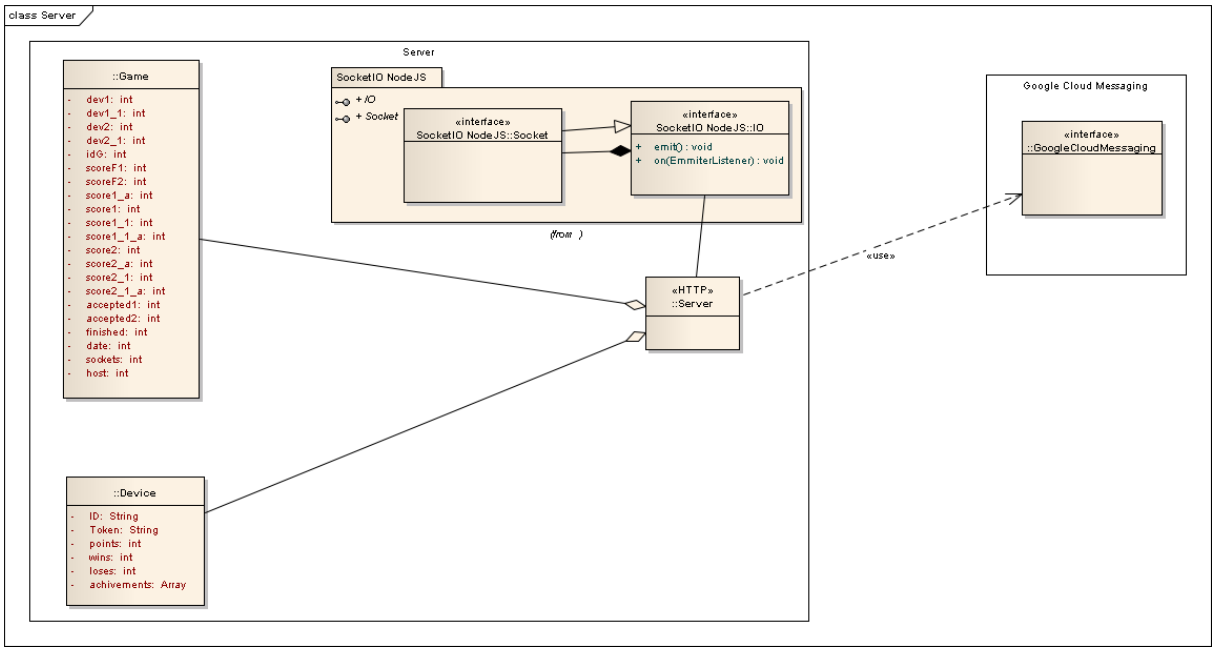


Figure 28: Multiplayer Server Class Diagram

As explained before the server uses the Socket IO and the Google Cloud Messaging libraries. The Socket IO offers two interfaces to communicate via web sockets to a singular client using socket or io.sockets.connected that are the set of active connected sockets.

The server contains two important elements of the data model that are the ones that intervene in the lockstep control of a match, the game and the device.

- **Device,** the device is the representation of a user of the Table Me application, this is used following the implementation proposed by Google for the device registration for the usage of Google Cloud Messaging. Additionally, is the soft replica of a user from the JOL Table Me Server, this replica is destined for multiplayer matches and pushing notifications following the architecture of the game Colyseus[17].
- **Game**, an exact representation of a match from the Table Me application for the phone, to control the score and the goals scored by the players, this is a soft replica to control following the architecture peer to peer and for the game Colyseus[17]. The game contains the score information of a match for each player, knowing when someone score a goal or an auto goal, helping to keep track

of a match. Additionally, this instance contains attributes that allow to follow if at least one player with a smartwatch wants to join a match and if the match has ended. Finally, for the representation of the players the id is the only information stored and the id of the web socket connection is stored for the players and the host of the match.

4.1.2. Server event behavior

The implementation was done based on events as the Socket IO library determine. The server manages the events based on a key string. The events relevant for the multiplayer server are the following:

- **Device registration ("dev_reg")**, the event registers a device that has a new Google Cloud Messaging token. The phone must send the player's id from the application, and the token retrieved from GCM. After receiving the data, the server creates a new device if there is none existing with the same information. The server checks if the user was registered before, in that case if the token changed from the one registered before, the device is updated.
- **Send notification ("send_not"),** after the user that host the match create it, a request for sending the notification is received checking if the players are registered and if they contain the GCM token to be able to push the notification of a new game. After that revision, an instance of a game based on an id counter is created and stored in a set of active games. The game id is returned to the host and notified to the players inside the notification message. Finally, the server sends the request to the service in a JSON object doing a retry of 10 times in case of error.
- **Set the match host ("set_host"),** after the user that host the match create it, the application notifies the server that the user will be the match host setting the web socket id of the host in the game.
- **Register web sockets ("reg_socket"),** after a player accept the challenge of a game in the smartwatch, the application registers the player in the server, initializing the game for multiplayer if at least one player with a smartwatch send the registration request. After that, the game start sending the actual score of the game (at the beginning on zeros), to all participants. This event is reused in case a player reconnects to the server.
- **Adding and dismissing goals ("add_goal"),** during a match the players and the host will interact sending multiple commands to the server about the match behavior, the command sent contains the id of the game, who did the command and which type. The server searches if the game id is active and then verify who did the action, updating the score and checking if a team won notifying the players in each case. If a team win the game is removed from the active games and the application in the players end the match.
- **Dismissing a game ("game_end"),** if the match host dismiss a game that have not finished yet, the players are notified so the application is closed in the smartwatches.
- **Disconnection of a player or host("disconnect"),** if a player is disconnected is removed from the array of player's sockets and later on can rejoin the game by registering the socket. However, if the host disconnects the game is finished.

- **Retrieve last match(“get_match”),** if the match host minimize the application, the service in the mobile device continues managing the game. If the host return to the application, the data is updated with the match instance in the server.4.2. Mobile Client

4.2.1. Class diagram

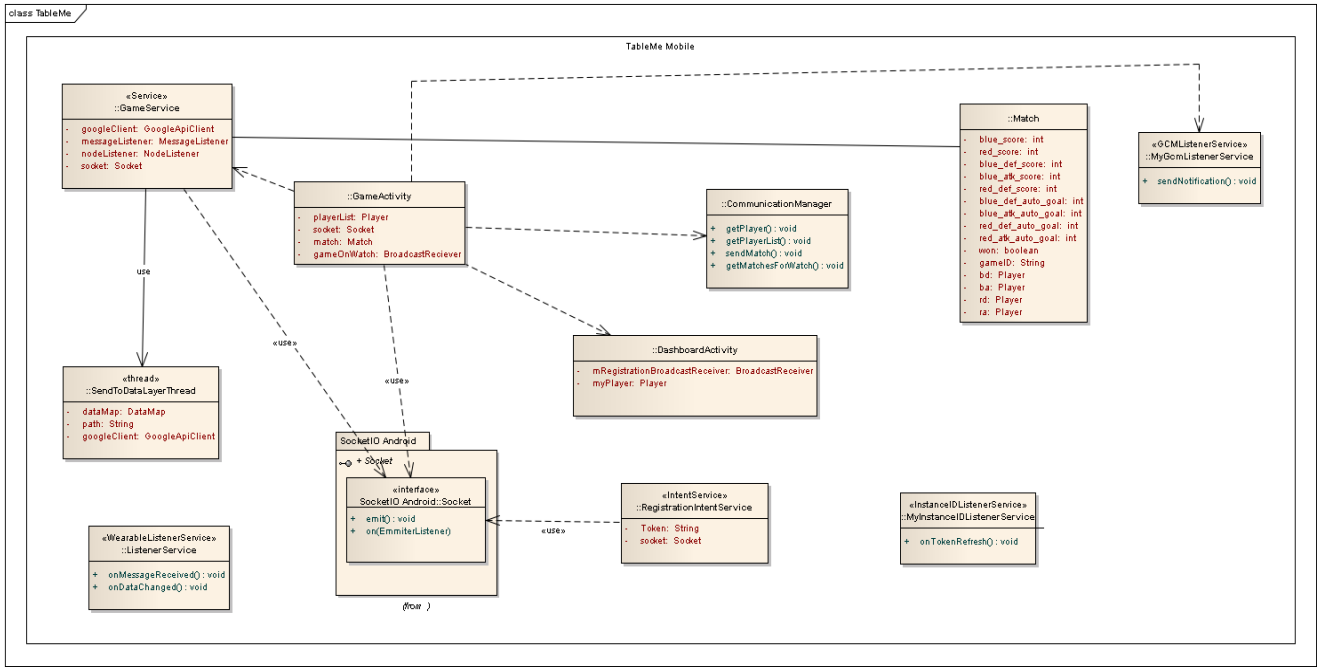


Figure 29: Table Me Domain Diagram

The Figure 29 illustrate the domain diagram of the application for the mobile devices. This diagram shows the developed items for this project and only the items used from the previous version of the Table Me application. To see the complete diagram (Link here).

The implementation for the mobile application is constituted mainly by the following elements:

- Elements from the previous implementation:
  - **Match**, representation of a match with the complete information of each player retrieved from the JOL Table Me server.
  - **Communication Manager,** manager of the HTTP request to the JOL Table Me server to retrieve profile data, the history matches and the leader board.
  - **Dashboard Activity,** activity that manage the layout of the home view of the application for the user, allow to see profile information, leader board, match history and create a new game. This activity is used to register the device using the Google Cloud Messaging [6] service and sending the information to the multiplayer server.
  - **Game Activity,** activity that manages the layout for the player selection for a game, the game management and the final results when a team wins. This activity is in charge of multiple task for the creation of a match to be used by the host and to notify the players. Additionally, direct the game score based on the host and the player’s actions.

- New elements:
    - **My GCM Listener Service**, a service that is initialize when a notification is received from the Google Cloud Messaging [6] service, and the message received has the game id and the participants of the game to inform the player in the smartwatch. For this, first a brief notification is raised in the phone and the Game Service is started to send a message to the smartwatch to raise a notification with the complete information of the message.
    - **My Instance ID Listener Service,** a service that is activated when the token from the Google Cloud Messaging [6] service using the Registration Intent Service to register the token to the server.
    - **Registration Intent Service,** a service that register a device in the Google Cloud Messaging service and sending the information to the multiplayer server.
    - **Wearable Listener Service,** a service that is active when a message or a data map is sent to the phone from the watch. This service receives the remote calls and start the Game Service with the required feature to execute depending on the message. In most, the service is used to receive request for the profile information, leader board, and match history.
    - **Send to Data Layer Thread,** a custom thread that send a data map using the Data API from the phone to the smartwatch.
    - **Game Service,** the main service that contains the features for the communication with the smartwatch. This activity handles the information transfer with the Message API for sending and receiving remote calls to the smartwatch, and sending data with the Data API. The game service uses the Communication Manager to retrieve data from the JOL Table Me server for the profile, leader board and match history information. Additionally, the serve manages the communication with the multiplayer server for sending commands of a game. Concerning the game data, the service has a soft instance of the match following the architecture of the game Colyseus [17].

4.2.2. Mobile event behavior

The mobile implementation has two type of events received, from the smartwatch and from the server using Socket IO.

- Socket IO events
    - On Game Service
        - **Update score (“add”)**, when the server sends a score update the data is separated and sent to the smartwatch with only the total score. If a host is using the smartwatch the event is received and a broadcast is sent to the Game Activity to update the layout.
        - **A team won (“winner”),** when a team wins the message is separated and is checked if the current player of the device is part of the winners on the

message, then the message of victory or lose is sent directly to the smartwatch.

- **Game ended by the host("res_game_end"),** if a game is closed by the match host without finishing the game the smartwatch is notified to close the application.
- **Disconnect,** if a player disconnects with the server a flag to reconnect is set.
- **Reconnect,** if a player reconnects with the server the web socket is registered to the server.

  - On Game Activity
    - **Sending notification response,** after sending the request for pushing challenge notifications to a set of players, the answer of the server is the game id of the instance created in the server.

- Message API events
  - **Player ready,** a player has accepted a challenge notification and wants to join a game, then a web socket registration is sent to the server.
  - **Add or dismiss a goal,** the type of command is received (add goal "+", dismiss goal "-" or auto goal "a"), and is sent to the server with the id of the current player.
  - **History detail,** in the match history feature if the user requests the info of a match, the application will retrieve the data of the players with the Communication Manager, then sent in a data map to the smartwatch.

4.2.3. Sequence diagram of the behavior of the Game Service

The Game Service is the manager of the functionalities with the smartwatch and the Game Activity manage the creation of a match, for this reason is necessary to do an emphasis of the basic behavior of the implementation for each functionality. Based on the previous explanation of the event behavior, the following diagrams illustrate the order how they are invoked in each phase.

4.2.3.1. User registration with the GCM token

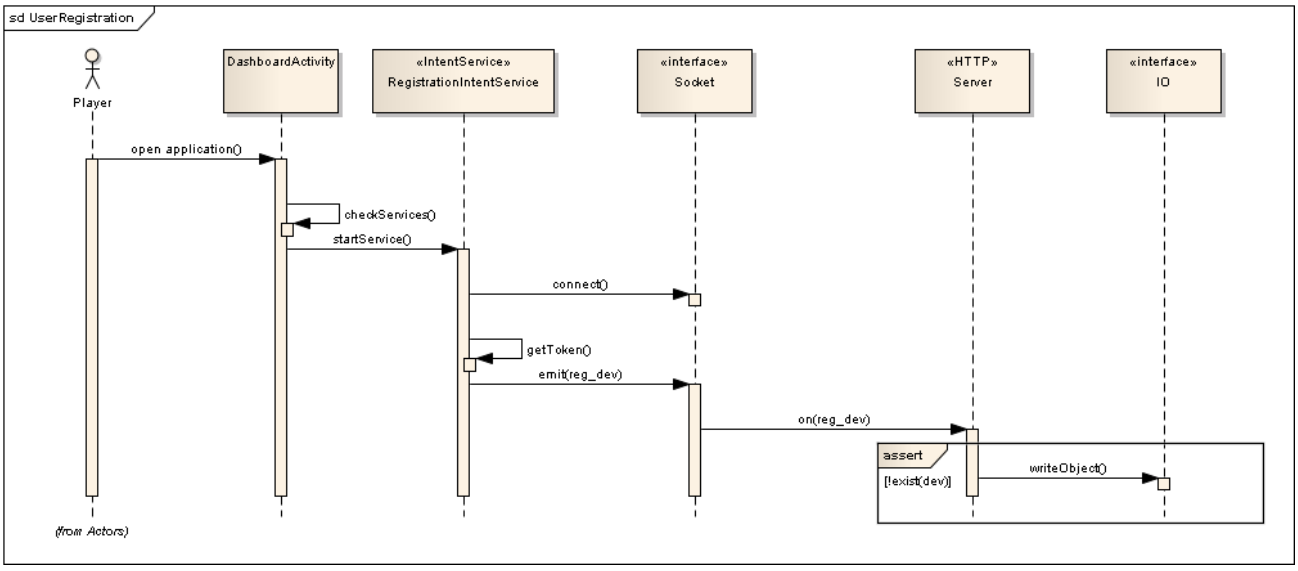


Figure 30: Device registration of the Google Cloud Messaging service token

When a user opens the application after logging in through the JOL Table Me server, the Dashboard activity is displayed, at the beginning on the creation of this layout the device registers to the Google Cloud Messaging service obtaining a token that is sent to the server with the id of the user.

4.2.3.1. Send challenge notification

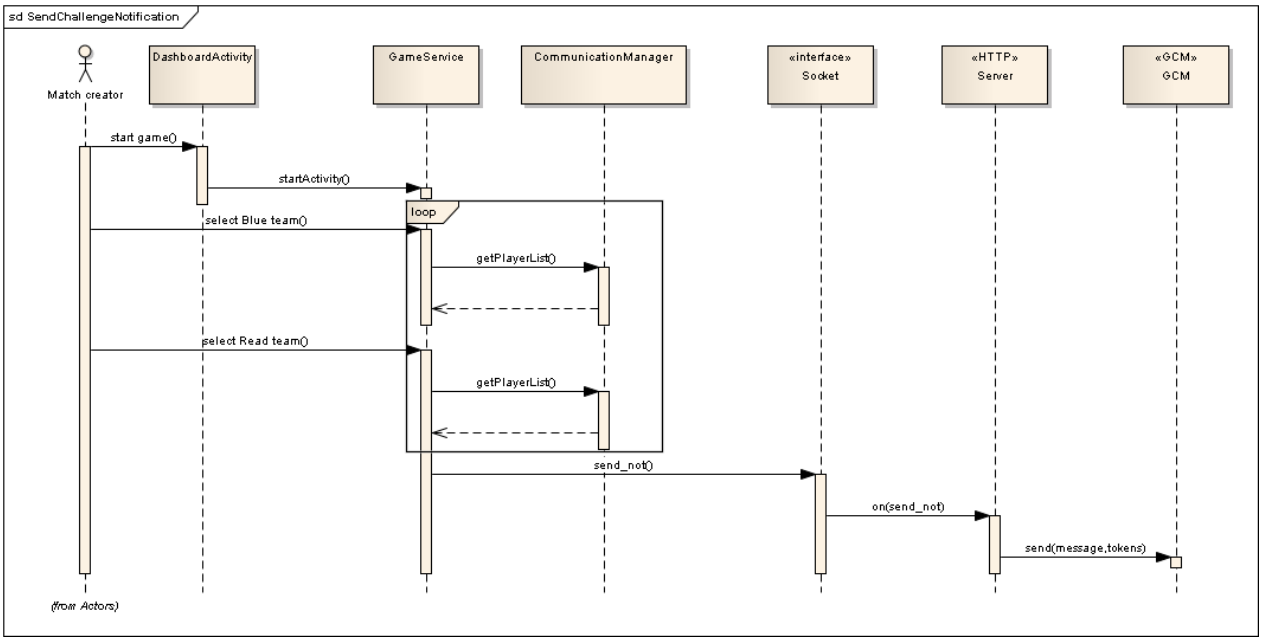


Figure 31: Sequence diagram for sending a challenge notification to a player

The match creator from the Table 1:Actors is the actor in charge of creating a match with the mobile device, after starting the Game Activity, the data of the teams is collected through the Communication Manager and after the selection a request notification is sent to the server with the key string "send_not" and the server send a message with the game id and the participants to the Google Cloud Messaging with the tokens of the players.

4.2.3.2. Receive Challenge Notification

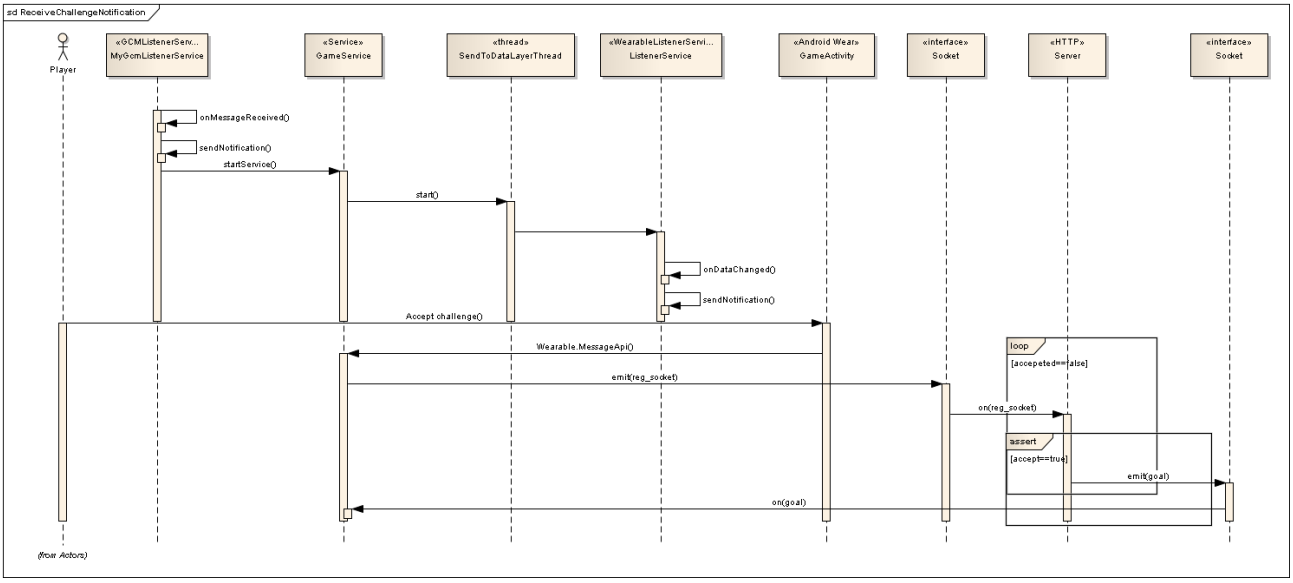


Figure 32: Sequence diagram for receiving a challenge notification from Google Cloud Messaging and showing it in the phone and the smartwatch

When a notification arrives from the Google Cloud Messaging, the service My GCM Listener Service start and show a brief notification in the phone and send the message through the Data API using Send to Data Layer Thread, to the smartwatch to raise a more detailed notification in the watch. After the player from the Table 1:Actors, sees the notification and accepts the challenge the Game Service start, then it proceeds to register the web socket to the server and after that the player begins to receive updates from the server.

4.2.3.3. In Game

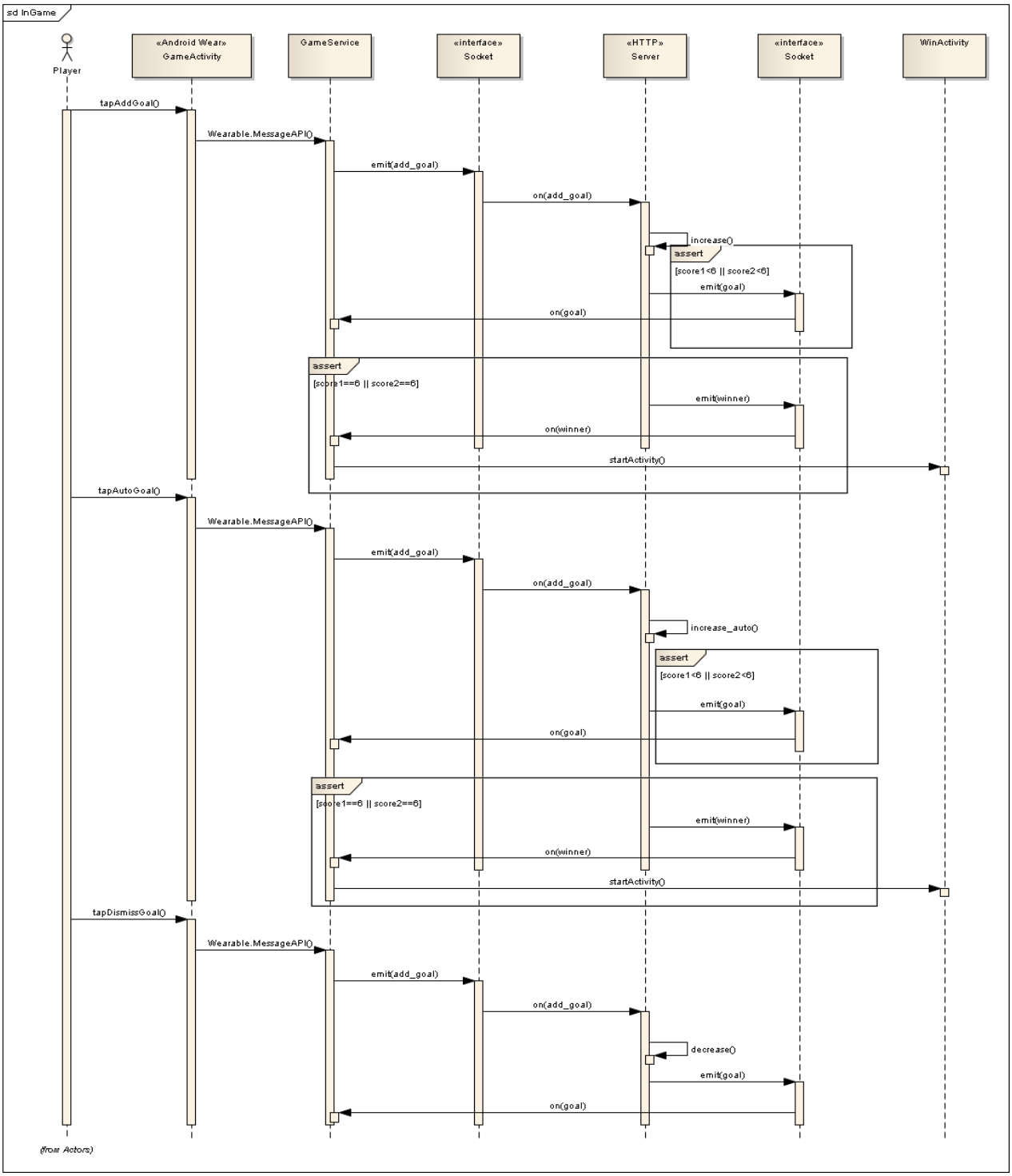


Figure 33: Sequence diagram for adding goals, auto goals and dismissing a goal during a match intthe smartwatch

This phase is based in the interaction of the player with the smartwatch in the Game Activity, tapping in the different options (add a goal, add an auto goal or dismiss a goal). After a player tap on a button the smartwatch communicates with the phone to send a command to the server to update the score. When a team wins the result is sent to the server so every player is notified of a victory with the list of players that won the match. In the case of dismissing a goal, the Match object take care of decrease properly the score and as before the score is sent to be updated.

4.2.3.4. Show profile information

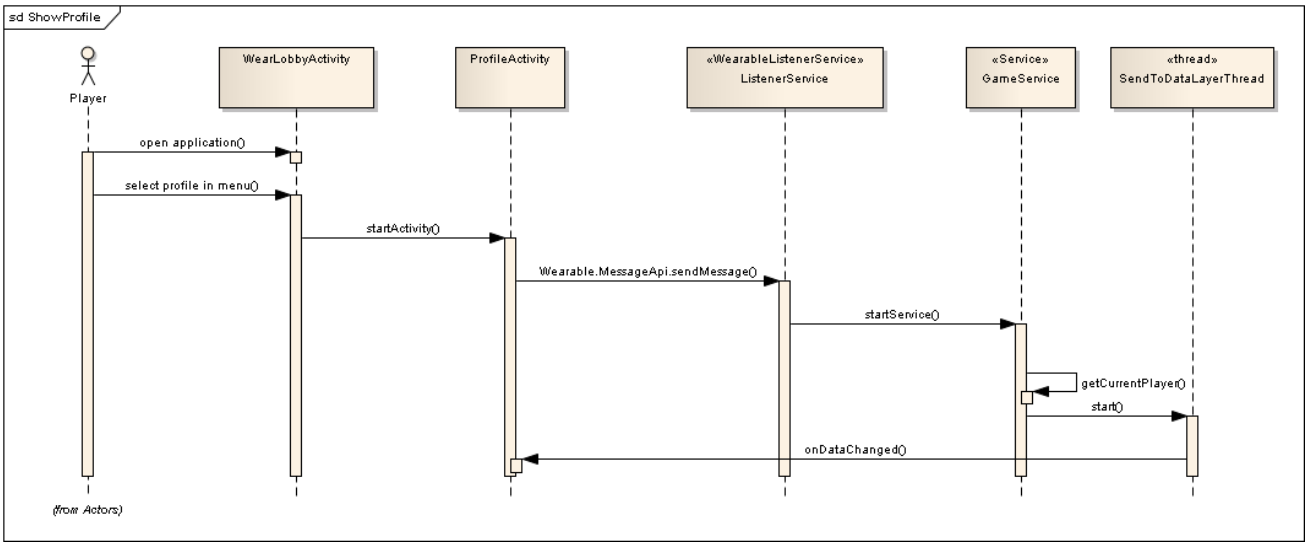


Figure 34: Sequence diagram for showing the profile information in the smartwatch

When a player opens the application in the smart watch and select the profile option, the phone is notified with a remote call using the Message API. After receiving the request, the phone start the Game Service and send the current information of the player registered in the device. All the relevant information (photo, name, elo and position) is sent through the Data API to the smartwatch, finally is displayed to the user.

4.2.3.5. Show match history

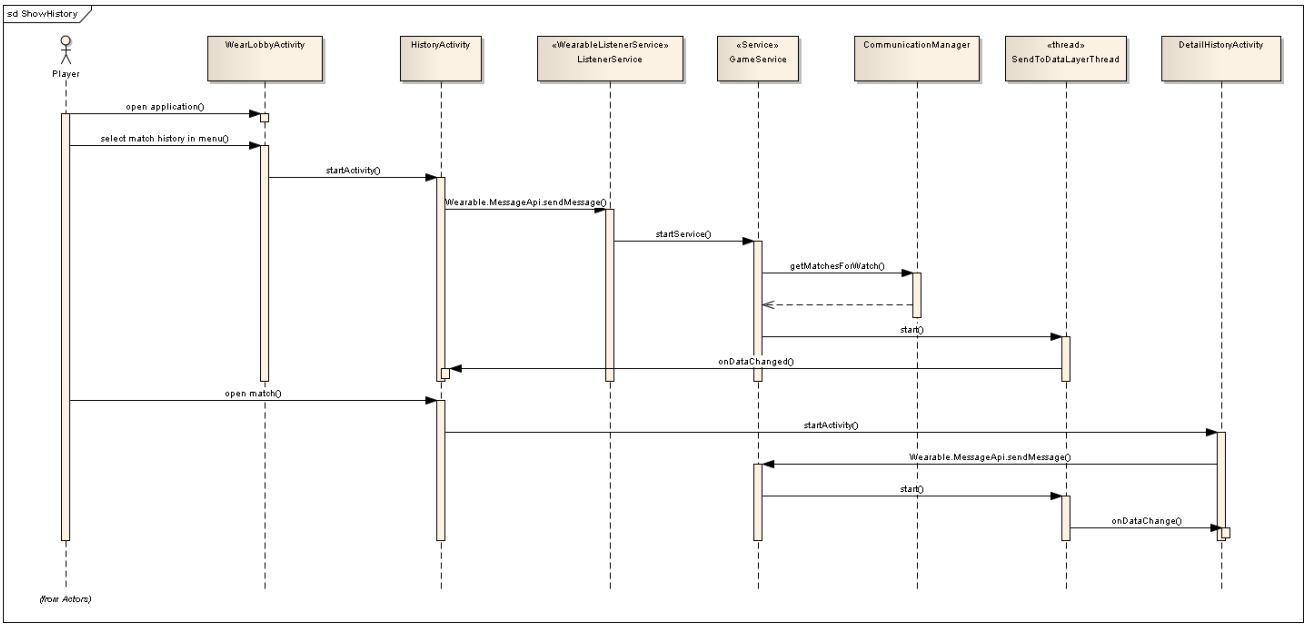


Figure 35: Sequence diagram to show the match history in the smartwatch of the player of the device

When a player opens the application in the smart watch and select the match history option, the phone with the Message API as before. The application uses the Communication Manager to retrieve the list of last ten games of the player. Then the score is selected and sent through Data API to the smartwatch. After that, if the player wants to see the detail of a match, the watch notifies with a remote call to the phone and the Game Service will receive the request of the information of a specific match. With that, the

service searches the information of the player's participant on the match and all the relevant information of the four players (photo, name, elo and position) is sent to the smartwatch.

4.2.3.6. Show leader board

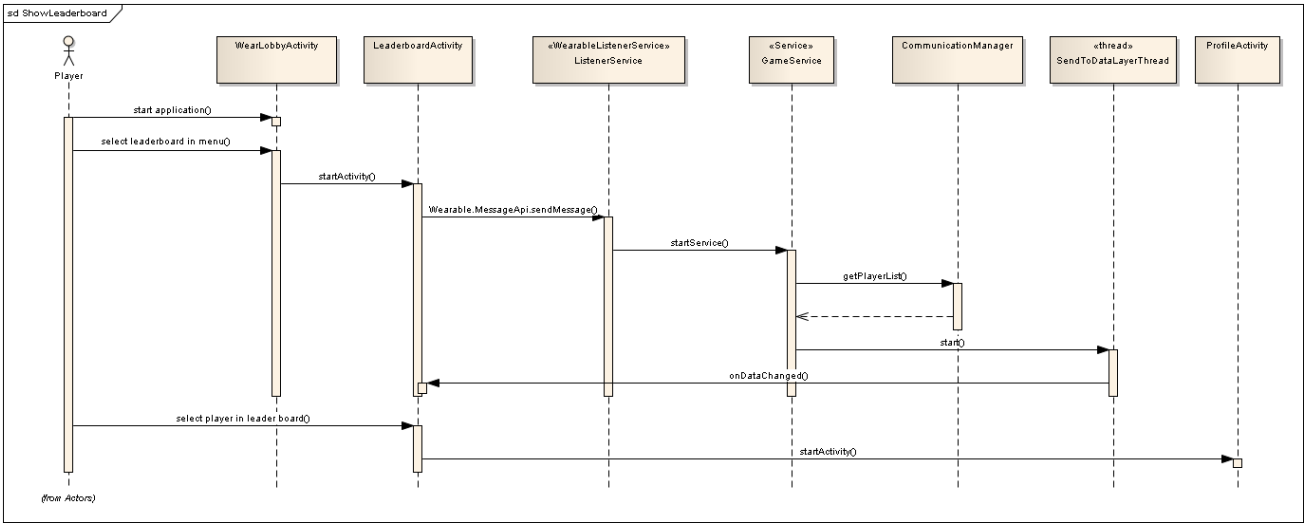


Figure 36: Sequence diagram for showing the leader board in the smartwatch

When a player opens the application and selects the leader board, the application is notified via Message API with a remote call and the Game Service start and retrieve the list of players organizing them by the position and selecting the first ten players. The all the relevant information (photo, name, elo and position) is sent to the smartwatch and displayed in a list with the photo, the name and the position in a badge with a distinctive color for the first three players. Moreover, if the player wants to see the profile information in detail of a player on the leaderboard, after clicking it on the list, the profile layout is displayed with the information of the specific player.

4.3. Wear Client

4.3.1. Class diagram

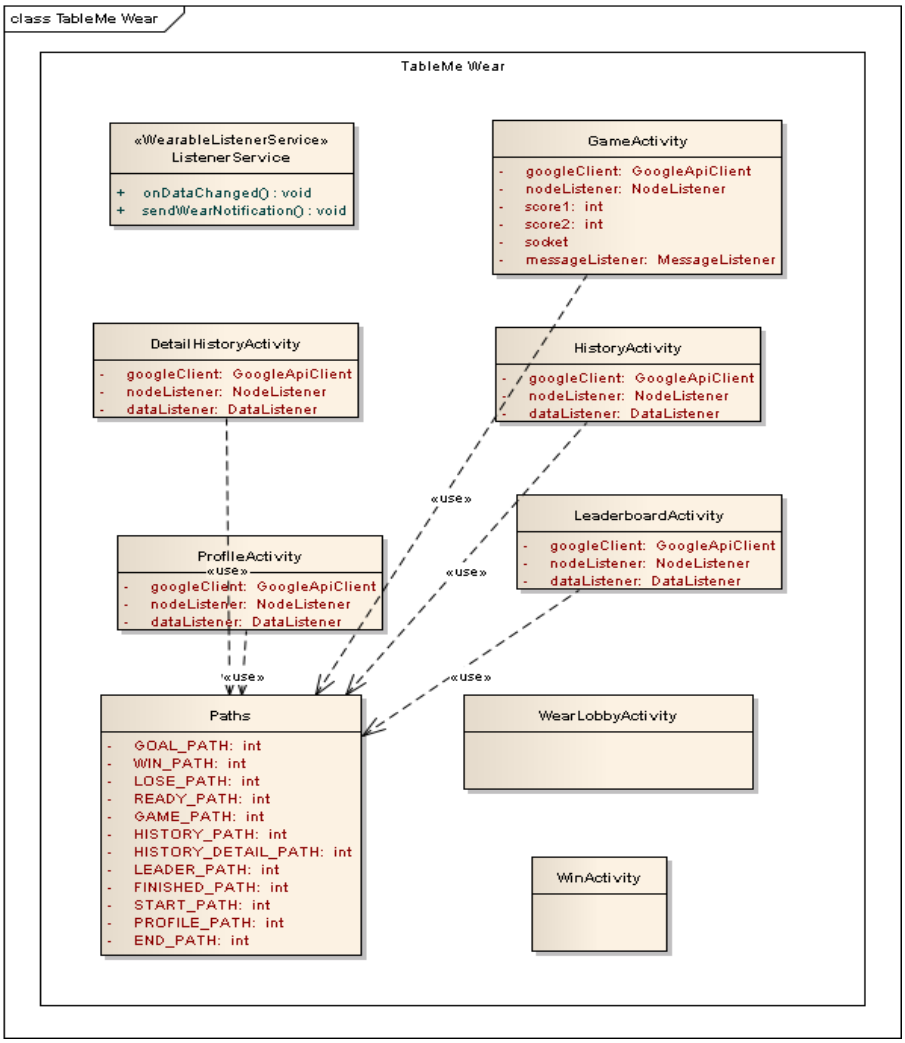


Figure 37: Table Me Wear Domain Diagram

The wear application for the smart watches was design to only deal with communication management and a light processing load of work that will be done by the mobile application. The application is directed by the Wear Lobby Activity that is the activity that manages the home layout of the application, and the Listener Service that receive the remote calls and information from the phone. Moreover, the communication relays only on the Wearable API through the Bluetooth connection between the devices. To see the complete class diagram, Annex: Class Diagram.

The next activities to be explain send a remote call through the Message API to invoke the Game Service in the phone so it can send the requested data in a data map through the Data API. Each activity creates an individual connection with the phone and receive the information.

The following elements are the structure of the Wear client.

- **Listener Service,** a service that start when receive a data map through the Data API, the service is dedicated to start the Game Activity when a match start or a player accept a challenge and to rise a notification sent from the phone.

- **Wear Lobby Activity,** the home layout when the user opens the application with a list with the options profile, match history, leaderboard, achievements and exit. The user can select the options and it will be redirected to the activity with the required information.
- **Win Activity,** activity that control a layout that shows when a team win or lose a match with the final score.
- **Profile Activity,** activity that control a layout that shows the relevant information (photo, name, elo and position) of a player. This activity is reused to show the detail of any other player in the leader board or in the match history.
- **Leader board Activity,** activity that controls a layout that shows in a list the ten first positions in the leaderboard with a photo, the name of the players and the position in a badge. If the user clicks on any of the players in the list, a detail of the selected player's profile is shown.
- **History Activity,** activity that controls a layout that shows a list of the ten last games played by the user. The list shows the score and an indicator if the user won or lose the game
- **Detail History Activity,** If the player clicks to see the detailed score, an activity will rise with the photos and positions of the teams and the score. Additionally, if the user clicks the photo of any participant player it displays the profile information of the selected player.

4.4. Tests

The applications were tested in every iteration of the development to assure the fulfillment of the use cases and main objectives of the project. When the functionalities of the application were finished, they were tested multiple times and the test were done incrementally (every item developed was always re tested together with the previous items to verify their behavior).

The following are the tests done during the development to ensure the completion of the objectives.

4.4.1. Use case testing

Functionality	Use Cases tested	Description of tests
Show profile information	UC-001, UC-024, UC-038	The tests performed were done by recovering the profile information from the Table Me server, and the communication with the smartwatch using the data layer, when the application is open and closed in the phone.
Show leader board	UC-001, UC-025, UC-039, UC-024, UC-038	The tests performed were done by recovering the list of players from the Table Me server and organizing them by the position. Additionally, the communication with the smartwatch using the data layer, when the application is open and closed in the phone.
Show match	UC-040, UC-026, UC-	The tests performed were done by recovering the list of matches of the player from the Table Me

history	001, UC-024, UC-038	server. Additionally, the communication with the smartwatch using the data layer, when the application is open and closed in the phone.
Challenge players	UC-003, UC-005, UC-006, UC-007, UC-002, UC-004, UC-008, UC-009	The tests performed were done in the Multiplayer server sending multiple notifications to the challenged players by a host, when the application is open and closed in the phone.
Start multiplayer game	UC-010, UC-011, UC-018, UC-019, UC-031, UC-027, UC-028, UC-011, UC-032	The tests performed were done in the Multiplayer server and the application accepting and creating a match with none active players up to four active players, when the application is open and closed in the phone.
Manage a multiplayer game	UC-015, UC-018, UC-019, UC-020	The tests performed were done in the Multiplayer server sending multiple commands from different games at the same time.
Show final results	UC-018, UC-019, UC-021, UC-026	The tests performed were done in the Multiplayer server processing the final results of a match and in the application sending the results to the Table Me server.
Android Wear Connection	UC-001, UC-010, UC-011, UC-015, UC-035	The tests were performed connecting and disconnecting the smartwatch during a match and in idle state when the user is in the menu.
Server Connection	UC-036, UC-031, UC-032, UC-028, UC-030, UC-018, UC-002	The tests were performed connecting and disconnecting the phone from internet during a match and in idle state when de user is not using the application or is in the dashboard.

Table 4: List of Use case testing performed.

4.4.2. User Interface Testing

The user interface in the smartwatch was validated via the lineaments given by Android [21], following the rules described by them. With this the following metrics were applied to verify each feature and screen to validate the usability.

- **Design for big gestures** [21]**:** in list menus as the home screen must display at maximum 3 items big enough to cover the screen to be easily touched.
- **Stream Cards** [21]**:** for the notifications, they were prioritized to be queued when a challenge arrives. If the player does not attend the notification the next one will replace it. The notification can be reused for multiple games without dismissing it.
- **Do one thing, really fast** [21]**:** Each feature of the application has followed the maximum amount of four touches and swipes to performed the functionality.
    - **Profile:** Open application, touch profile, dismiss card.
    - **Leader board:** Open application**,** touch leader board, dismiss card.
    - **Game history:** Open application, touch game history, touch summary, dismiss.

- **Game:** In game interaction, dismiss result card.

- **Design for the corner of the eye** [21]**:** The application has summarized information on each card to not keep the player too long in the application, so it was decided to check at maximum to have five items per screen that fill the interface that are easy to read.
  - **Profile:** Photo, name, elo, classification.
  - **Leader board:** List with photo and classification.
  - **Game history:** List with indicator of victory or loss, score, text of victory or loss.
    - **Game summary:** Four photos and score.
  - **Game:** Add goal button, add auto goal, remove goal, score.
    - **End game:** result badge and text of victory or loss.
- **Don't be a constant shoulder tapper** [21]**:** The application only will notify when the user is challenged vibrating just once per notification.

5. Conclusions

- The objective of creating a multiplayer version of the Table Me application was totally fulfilled developing and adapting the main features of the Table Me application, following the design lineaments of Android and the user's requirements.
- The implementation of the Peer to Peer server with lockstep without simulation using Socket IO to improve the communication allow to the application an easy to use interface in real time to communicate the application with the server allowing to not increment significantly the delay the communication between the items that have a small traffic interaction.
- The communication performance given by the application using Android Wear technology in the testing environment gave excellent results with a maximum delay of 1 to 2 seconds to deliver the information to the smartwatch, this delay due to the Bluetooth connection range and obstacles that can be between the devices.
- The APIs from the Wearable package used in the project to develop the communication with the application were the Message API due to the easy usage and the functionalities that provide to send appropriately remote procedure call with the games commands (goals, dismiss goals, auto goals) adapting the Peer to Peer server's structure Finally, the Data API to synchronize large amounts data (profile, leader board, match history) using Assets and supporting basic data types.
- The Android Wear application was design to not consume large amounts of energy of the smartwatch during the operation during the game or when the user is consulting information. For this reason, the application does not keep the screen on until the user actively focusses the screen to use it, and the heavy load operations are provided by the service in the phone that runs only when the user request information through an interaction.
- The design of the application for the smartwatch was based on sketches previously done by the group in the laboratory, and the last version fulfilled the final user's expectation giving enhanced and easy to use interfaces during the gameplay and consulting the player information following the design lineaments of Android [21].
- The methodology used for the development of the project allowed to create an organize and schedule to deliver the application and deliverables items to fulfill the items to be developed with the approval of the final user on time.
- The Android wear technology is advancing faster, allowing different functionalities to be used with the phone or a direct connection with internet, and the basic channels of communications between the devices, offer the basic data exchange and easy to use. It allows an easy and fast development of android wear applications. However, the management of large amount of data to be exchanged via Bluetooth can take too long and present inconsistent data. With the time when the technology migrates to the usage of Wi-Fi or better connection with the smartwatch allow a better network performance.

6. Bibliography

[1] "The Spiral Model > UltimateSDLC.com," *The Ultimate Guide to the SDLC*, 13-Aug-2011.

[2] Enrico Catalano, "Table Me - table soccer app on Behance." [Online]. Available: https://www.behance.net/gallery/31215047/TableMe-table-soccer-app. [Accessed: 10-Apr-2016].

[3] "2359 Media: Mobile Application Development | Android for wearables: Opportunities and Limitations of Watch Apps." [Online]. Available: http://2359media.com/2014/09/26/android-for-wearables-opportunities-and-limitations-of-watch-apps/. [Accessed: 25-Apr-2016].

[4] "Sending and Syncing Data | Android Developers." [Online]. Available: http://developer.android.com/intl/es/training/wearables/data-layer/index.html. [Accessed: 11-Apr-2016].

[5] "Wearable," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/Wearable. [Accessed: 11-Apr-2016].

[6] "Google Cloud Messaging: Overview," *Google Developers*. [Online]. Available: https://developers.google.com/cloud-messaging/gcm. [Accessed: 10-Apr-2016].

[7] Dan Adrei, "How To Create a Server to Send Push Notifications with GCM to Android Devices Using Python," *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-server-to-send-push-notifications-with-gcm-to-android-devices-using-python. [Accessed: 10-Apr-2016].

[8] "Creating a Notification for Wearables | Android Developers." [Online]. Available: http://developer.android.com/intl/es/training/wearables/notifications/creating.html. [Accessed: 10-Apr-2016].

[9] "MessageApi," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageApi. [Accessed: 10-Apr-2016].

[10] "WearableListenerService," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/WearableListenerService. [Accessed: 11-Apr-2016].

[11] "Accessing Google APIs," *Google Developers*. [Online]. Available: https://developers.google.com/android/guides/api-client. [Accessed: 11-Apr-2016].

[12] Michael Hahn, "Data Layer Messages — Android Wear Docs 1.2 documentation." [Online]. Available: http://android-wear-docs.readthedocs.org/en/latest/sync.html. [Accessed: 10-Apr-2016].

[13] "DataApi," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/DataApi. [Accessed: 10-Apr-2016].

[14] "Asset," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/Asset. [Accessed: 11-Apr-2016].

[15] Michael Hahn, "Data Layer DataMap Objects — Android Wear Docs 1.2 documentation." [Online]. Available: http://android-wear-docs.readthedocs.org/en/latest/data.html. [Accessed: 10-Apr-2016].

[16] "P2P." [Online]. Available: http://ww2.cs.fsu.edu/~jungkkim/P2P.html. [Accessed: 28-Apr-2016].

[17] "A Distributed Architecture for Interactive Multiplayer Games, Ashwin R. Bharambe Jeff Pang Srinivasan Seshan." .
[19] "php - What are Long-Polling, Websockets, Server-Sent Events (SSE) and Comet? - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet. [Accessed: 28-Apr-2016].
[20] "drewww/socket.io-benchmarking @ GitHub." [Online]. Available: http://drewww.github.io/socket.io-benchmarking/. [Accessed: 25-Apr-2016].
[21] "Design Principles for Android Wear | Android Developers." [Online]. Available: http://developer.android.com/intl/es/design/wear/principles.html. [Accessed: 25-Apr-2016].
[22] "FURPS - Ingeniería Software." [Online]. Available: http://clases3gingsof.wikifoundry.com/page/FURPS. [Accessed: 28-Apr-2016].
[23] Dan Adrei, "How To Create a Server to Send Push Notifications with GCM to Android Devices Using Python."
[24] "Android Wear Book: Create an Advanced Wearable List View," *Home*. .
[25] Dejan Đurovski, "Android Wear - Wearable Message Api." [Online]. Available: http://dejan.djurovski.net/2015/01/15/android-wear-wearable-message-api/. [Accessed: 10-Apr-2016].
[26] Paresh Mayani, "Android Wear - Part 4 - Simple notifications." [Online]. Available: http://www.technotalkative.com/android-wear-part-4-simple-notifications/. [Accessed: 10-Apr-2016].
[27] "Developer Guidelines," *Google Developers*. [Online]. Available: https://developers.google.com/nearby/developer-guidelines. [Accessed: 10-Apr-2016].
[28] E. Terpstra, "Building Multiplayer Games with Node.js and Socket.IO," *Modern Web*, 30-Sep-2013.
[29] "Android Wear Docs, michael Hahn."
[30] "Ingenieria de requerimientos, Herramienta para Implementar LEL y Escenarios." .
[31] S. Okamoto, M. Kamada, M. Kohana, and T. Yonekura, "Rapid Authoring of Web-based Multiplayer Online Games," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, New York, NY, USA, 2013, pp. 639:639–639:643.
[32] "ChannelApi," *Google Developers*. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/wearable/ChannelApi. [Accessed: 11-Apr-2016].
[33] R. D. Richard K. Lomotey, "Efficient mobile services consumption in mHealth." [Online]. Available: http://dl.acm.org/citation.cfm?id=2500279&CFID=599129620&CFTOKEN=16880902. [Accessed: 10-Apr-2016].
[34] arungupta, "REST vs WebSocket Comparison and Benchmarks," *Miles to go 3.0 ...*, 24-Feb-2014.
[35] Lorie Pisicchio, "SSE vs Websockets," *Streamdata.io*, 21-Apr-2015. .
[36] cjihrig, "Server-Sent Events in Node.js," *Colin J. Ihrig's Blog*, 08-Aug-2012. .
[37] Nodejs Hispano, "Introducción a Socket.io #nodejs | Node.js Hispano." .
[38] "Sending and Receiving Messages | Android Developers." [Online]. Available: http://developer.android.com/intl/es/training/wearables/data-layer/messages.html. [Accessed: 10-Apr-2016].
[39] "Gcm http://developer.android.com/intl/es/training/wearables/data-layer/index.html Service," *Google Developers*. [Online]. Available:

https://developers.google.com/android/reference/com/google/android/gms/gcm/GcmListenerService. [Accessed: 11-Apr-2016].
[40] "Il modello di comunicazione long-polling," *HTML.it*. [Online]. Available: http://www.html.it/pag/33600/il-modello-di-comunicazione-long-polling/. [Accessed: 11-Apr-2016].
[41] "Simple Long Polling Example with JavaScript and jQuery." [Online]. Available: http://techoctave.com/c7/posts/60-simple-long-polling-example-with-javascript-and-jquery. [Accessed: 11-Apr-2016].
[42] "Research of Web Real-Time Communication Based on Web Socket, by Qigang Liu, Xiangyang Sun." .
[43] "The Web Sockets API." [Online]. Available: https://www.w3.org/TR/2009/WD-websockets-20091222/. [Accessed: 11-Apr-2016].
[44] "WebSocket," *Wikipedia, the free encyclopedia*. 01-Apr-2016.
[45] "Tecnología Push," *Wikipedia, la enciclopedia libre*. 26-Nov-2015.
[46] "Socket.IO JavaScript framework ready for real-time apps | InfoWorld." [Online]. Available: http://www.infoworld.com/article/2607757/javascript/socket-io-javascript-framework-ready-for-real-time-apps.html. [Accessed: 12-Apr-2016].
[47] "Real-Time Systems." [Online]. Available: https://users.ece.cmu.edu/~koopman/des_s99/real_time/. [Accessed: 25-Apr-2016].
[48] "Socket.IO — Introducing Socket.IO 1.0." .
[49] "A Node.js speed dilemma: AJAX or Socket.IO? | CUBRID Blog." [Online]. Available: http://www.cubrid.org/blog/cubrid-appstools/nodejs-speed-dilemma-ajax-or-socket-io/. [Accessed: 25-Apr-2016].
[50] "Android Wear," *Wikipedia, the free encyclopedia*. 20-Apr-2016.
[51] "About | Node.js." [Online]. Available: https://nodejs.org/en/about/. [Accessed: 28-Apr-2016].
[52] "El Libro para Principiantes en Node.js» Un tutorial completo de node.js." [Online]. Available: http://www.nodebeginner.org/index-es.html. [Accessed: 28-Apr-2016].
[53] "Understanding the Node.js Event Loop," *NodeSource*, 21-Jan-2015. [Online]. Available: http://nodesource.com/blog/understanding-the-nodejs-event-loop/. [Accessed: 28-Apr-2016].
[54] "What is Spiral model- advantages, disadvantages and when to use it?" [Online]. Available: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/. [Accessed: 28-Apr-2016].
[55] "Android Wear | Android Developers." [Online]. Available: https://developer.android.com/design/wear/index.html#Other. [Accessed: 05-Oct-2016]
[56] "Gaffer on Games | Deterministic Lockstep." Available: http://gafferongames.com/networked-physics/deterministic-lockstep/ [Accessed: 05-Oct-2016].
[57] "Introduction | Understanding Model-View-Controller." [Online]. Available: https://stefanoborini.gitbooks.io/modelviewcontroller/content/. [Accessed: 05-Oct-2016].
[58] "WebSockets vs Server-Sent Events vs Long-polling." [Online]. Available: http://dsheiko.com/weblog/websockets-vs-sse-vs-long-polling/. [Accessed: 05-Oct-2016].

7. Appendixes

7.1. Software Requirement Specification

POLYTECHNIC OF TURIN

Faculty of Engineering

Master's Degree

in Computer Engineering

Software Requirement Specification

Andres Camilo Jimenez Vargas

October 2016

Index

Figure table

Table index

1. Introduction

The Joint Open Lab (JOL), a research group of Telecom Italia (TIM), developed an application named Table Me for Android and IOS mobile devices, to manage a game of table foosball. This document is done to describe the process of recollection and analysis of the requirements for a distributed game version of the Table Me application using smart watches, the application will be developed for Android mobile devices and Android Wear devices.

For the correct way to structure and implement this solution is necessary to control the requirements that the system needs based on the user's necessities and the previous scope of the Table Me application done before. The requirements are the base of the development process of the system, for this reason these must be analyzed in an adequate way to identify correctly the features of the application and the restrictions that this and the projects are attach.

Therefore, this can be used as a manual for the development of the project, this will allow to do the appropriate evaluation of the punctual functionalities that must be in the application, develop a prioritization to know the order of the implementation of the features, and finally trace all the dependencies of the system.

The present will have a scope of the following characteristics:

- Description of the process followed for the requirement recollection and analysis.
- Description of the prioritization process of the requirements to determine the functionalities to implement.
- Description of the process for obtaining the use cases.
- The domain model of the Server, the Android and the Android Wear applications.

2. Process of development of requirements

2.1. Obtaining the use cases and requirements

The recollection of the use cases and the requirements was done based on the description of the game given by Marco Marengo, Cecchi Gian Luca and Alessandro Izzo knowledgeable people that through a series of reunions had explain application functionality and the desired features to be implemented with the Android Wear technology. Additionally, the usage of the actual working application in the testing version on the Google Play Store. From this was developed a description of the main features and their principal elements.

2.2. Refinement of the use cases

For the refinement of the use cases, every time a set of use cases were developed, the application was presented to Marco Marengo and from that, all the feedback was implemented in the application and corrected in the use case and requirement specification. Additionally, with the guidance Marco Marengo, Cecchi Gian Luca and Alessandro Izzo and the usage of the Table Me application, it is possible to concrete the scenarios when and where the application will be in use.

2.3. Functional requirements

Based on the description of the application's features, the experience acquires form the presentations and the use cases that have been developed the functional requirements of the system. Where are specified the characteristics that the project must include for the completion.

2.4. Nonfunctional requirements

Based on the presentations of the application and the design decision taken, it has been developed the nonfunctional requirements of the system. These describe the attributes and restriction that the project must achieve to be accepted by the client.

2.5. Requirement identification

The identification was done through the use cases based on the description of the application. Based on these, the application's flow was described in the assert or failure situations and from that the requirements were identified contemplating the lineaments of the previous Table Me application. Thanks to this process it was able to classify the requirements adequately easing the requirement specification.

2.6. Requirement specification

The attributes used for the specification of the requirement were:

- **ID:** Unique deification of the requirement.
- **Description:** Definition of the requirement.
- **State:** Shows the phase of progress of the requirement in the following classification: Proposed, Validated, Implemented.
- **Priority:** priority of each requirement that is based on the modified formula of Karl Wiegers, that give a score that if is greater than 1 has more priority.
- **Type:** Functional or Nonfunctional.
- **Relations:** Show the relation of the requirement with the other requirements and the use cases, these can be of dependence or realization.

2.7. Requirement prioritization

The prioritization of the requirements was estimated based on the formula of Karl Wiegers. The method is described in the following way (All the measures are done from 1 to 9):

- **Benefit:** relative benefit of the implementation of the requirement.
- **Penalization:** relative penalization if the requirement is not implemented.
- **Cost:** calculation of the relative cost, based in the following factors: complexity, design and code reuse, and documentation needed.
- **Risk:** relative risk based on the technic complexity and the feasibility needed.

It is necessary to remember that these indicators are subjective to the person involved in the development and implementation of the project.

The formula of Wiegers is the following:

$$\frac{Value\ \%}{(\%\ cost * cost\ weight + \%\ risk * risk\ weight)}$$

Figure 38: Original Karl Wiegers formula for requirement prioritization

Where the value is the sum of the benefit and the penalization in percentage. The cost weight and the magnitude assigned to each item, in this case there has been assigned the same magnitude to both for this their value is 1.

In conclusion, it has been decided for the project's requirement prioritization to use this measurement where the higher calcification shows the most complex and costly requirements to implement.

2.8. Verification and validation

The verification of the requirements was done through the reunions with Marco Marengo, Cecchi Gian Luca and Alessandro Izzo and the presentations of the application refining the requirements.

The validation of the requirements was done using a checklist by the Construx Software Builders that verify the following attributes.

- Is unique?
- Is explanatory?
- Is consistent?
- Is feasible?
- Has correct references?
- Is precise and not ambiguous?
- Us atomic?
- Is traceable?
- Is a declaration of a necessity and not a solution?
- Is it necessary?
- Show how to execute it?

2.9. Requirement traceability

2.9.1 General traceability

The traceability of the requirements is developed through different tools that were used to perform the follow up, identification of requirements and the dependencies to help the development of the application and the integration with the previous Table Me application.

The tools used to manage the traceability of the requirements are the following:

- **Use cases:** the use cases are used for the identification of the requirements, moreover are necessary to establish the information flow in the different scenarios.
- **Prioritization table:** The table of requirements prioritization to identify easily the order of implementation of which requirements must be implemented in the correct way achieving the essential functionalities of the system. **(**Annex: **Prioritization Table)**
- **Realization Matrix:** This matrix is used to identify the relations between the requirements and use cases, so in the same way in a graphic way identify the dependencies between requirements and how to find the prerequisites between them.

- Annex: **Realization Matrix for functional requirements.**

2.9.2. Individual traceability

The individual traceability of the requirements is done by the attributes that allow to identify the relations of dependency and implementation through the system.

The attributes of each requirement to be traced are the following:

- **Use case relations:** the use case where the requirement was obtaining, and has a relation that implements it.
- **Requirement relations:** The requirements that are associated and dependent to the requirement.

3. Global Description

3.1. External interfaces

3.1.1. User Graphic Interface

3.1.1.1 Profile information on the smartwatch

After opening the application, the user can use a list of option to select about the user's profile information and the game. The profile option shows in a similar way as in the Table Me application the basic information of the user. Firstly, the photo, the position in the leader board inside a badge, two bars that indicate proportionally the number of victories (green bar) and number of lost games (red bar). The at the end the name of the user and his/her ELO score.

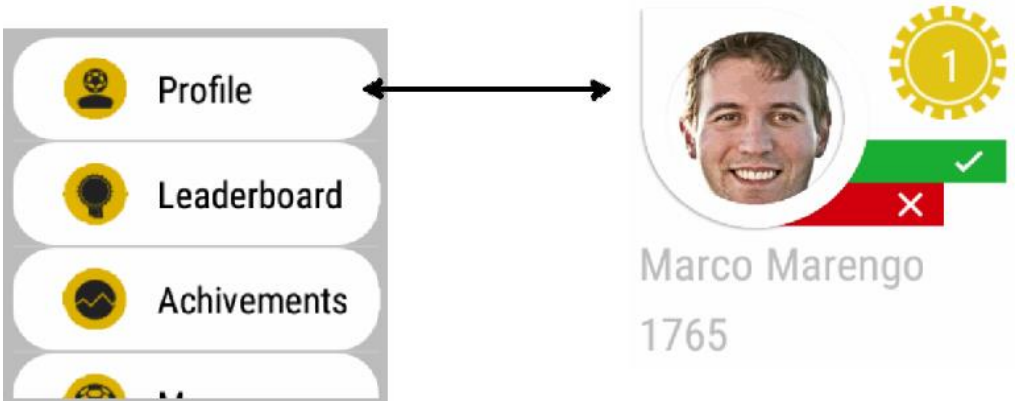


Figure 39: Profile information

3.1.1.2. Gameplay on the smartwatch

After a user that will host a game organize and create the team, a notification is issued to the participants of the match will receive a notification in their phones and to their smartwatches if they have one. At the watch the following notification is displayed.

Figure 40: Notification on the smartwatch

Then, when the user start playing the game, the screen with the buttons where the player can interact with the score is provided, with a button to add a goal in the center, a dismiss goal button at the left bottom and an auto goal button at the right bottom. The user can interact with them and the score is displayed in the results displayed in top of the screen with its respective colors of the team.

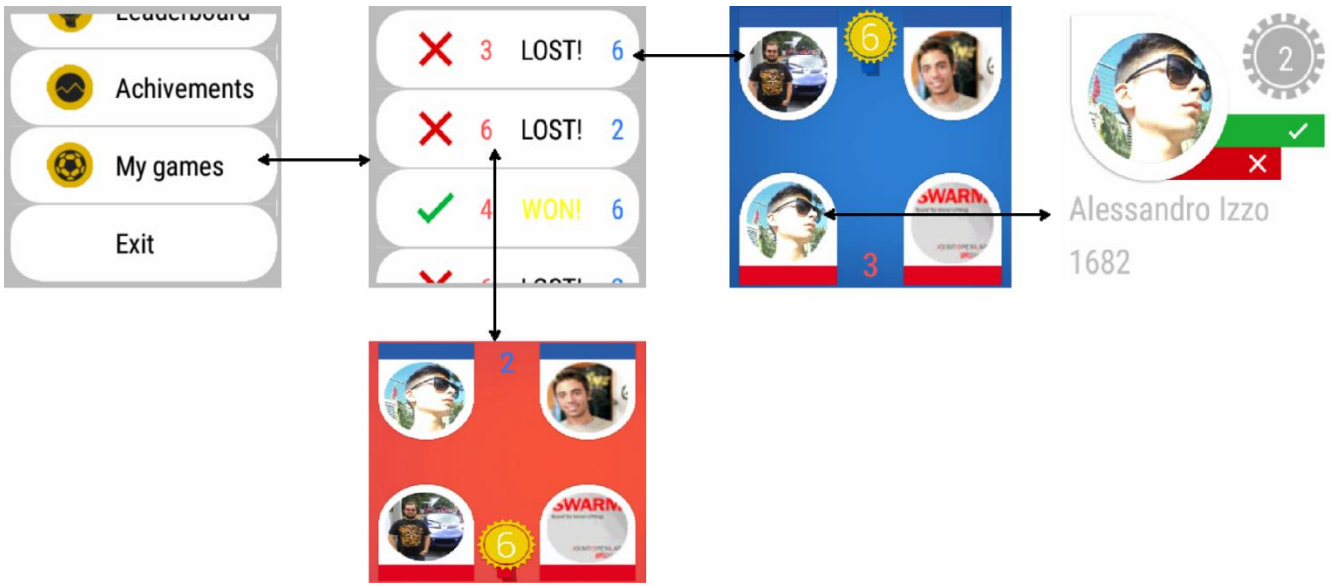
Figure 41: In Game screens with victory and lose screens

Finally, in each case the users of a team win or lose a match the following screens are displayed in the smartwatches using a base badge with their goals scored.

3.1.1.3. Match history on the smartwatch

When the match history option is selected, the user is provided with a brief list of the last ten matches that he/she was part of. Each match is signaled with a cross and the message “Lost!” if the player lost that game, or a check and the message “Won!” if the player won the game. If the player wants to know a more detailed information of the

match, it is possible to click an item on the list and will provide a screen with the background color of the victorious team, the score, and the photo of the participants. If the user wants to see the players profile information, he/she can click on the image and the profile information will be displayed.



3.1.1.4. Leader board on the smartwatch

When the leader board is selected a list of the ten first places in the leaderboard is loaded and take between 3 to 4 seconds. After waiting, the list displays the basic information of the player in the displayed position with his photo, name and position in a badge. If the user wants to see the profile of the player in a certain position, he/she can select a player in the list and it will show the basic profile information of the player.

Figure 42: Leader board screen

3.1.2. Software and Hardware interface

The Android mobile application is able to deploy on mobile devices that have a distribution from Android 5.0 (Lollipop) to Android 4.1 (Jelly bean). And for the Android Wear application that have a distribution from Android 4.4 (KitKat) to Android 5.0 (Lollipop). The application can be used in smart watches with round screen or rectangular

screen. Finally, the server can be deployed in a computer that has installed Node JS and had installed Socket IO, and Google Cloud Messaging plugins for Node JS.

3.2. Application characterization

The final system must allow the following characteristics:

- The server application must support multiple client connections in multiple matches.
- The server application must allow to create multiple matches.
- A client through the TableMe application can create a game notifying the server and the involved players allowing them to use their smartwatches if they have any.
- The match host and the players can help to manage a game's score.
- Every player with a smartwatch can score a goal, dismiss a goal or score an auto goal.
- All the players and the host will be notified in real time from the server.
- The host of the game will see the progress of the game in the user graphic interface of a match from the previous TableMe application.

3.3. User characterization

User type	Description	Privileges	Previous knowledge
Player	A player can enter to a game is has a smartwatch, the notification is displayed on the watch and can be opened from it.	• Connect to a match. • Score goals, dismiss goals and score auto goals. • Win a match.	The user must have the application in the mobile device, in the smartwatch. Additionally, know how to play table foosball.
Host	The host is a user that creates a game from the TableMe application inviting to play other players.	• Control all the scores of all the players. • Create a match. • Finish a match.	The user must have the application in the mobile device. Additionally, know how to play table foosball.

Table 5: User characterization table

3.4. Restrictions

For the development of the project are the following restrictions:

- **Language restriction:**
    - The clients must be implemented in Android for the Android mobile and Android Wear devices.
    - The server must be implemented in Java Script using the platform Node JS.
- **Programming restriction:** The project will be implemented using the Object Oriented paradigm for software development.
- **Connection restriction:**
    - The server must admit multiple user connection simultaneously for multiple matches.
    - The mobile client must connect to a single server that host the matches.
    - The match will be alive until the match host finish the game or it gets disconnected.
- **Game restriction:**
    - The system must implement a table foosball match rules with four players, following the lineaments stipulated by the previous TableMe application.
- **Architecture restriction:**
    - The system must develop following the Client/Server architecture for the communication for between the server and the mobile devices.
    - The system must use the Colyseus multiplayer game architecture for the implementation of the game instances in the clients and the server.
- **Persistent restrictions:**
    - The server application must persist the devices registered with a new Google Cloud Messaging token.
    - The server will not use a data base for the persistency.
    - The persistency will be done through normal files.
- **User interface restrictions:**
    - The interfaces of the system will be in English.

3.5. Assumptions and dependencies

The following dependencies are considerate for the development and execution of the system:

- The server used in the previous TableMe application will provide information of the user's profile, and match history. Additionally, the leaderboard of the players.
- The mobile devices used must be connected to internet.
- The mobile devices may or not have a smartwatch paired.

- The requirements of the system are formulated by Marco Marengo though presentations of the application and the guideline of the previous Table Me application.

The following assumptions are considered for the development and execution of the system:

- The mobile and smartwatch devices where the application is installed can execute the application.

3.6. Domain Model

The domain model of the application is represented in the following diagram: and specified in the following document: Annex: Domain Model

3.7. Requirement distribution.

Based on the process of identification of requirements, it has been stipulated that the following classification of requirements must be divided in different functionalities and phases of the game and the application. The following diagram shows the classification of the requirements:

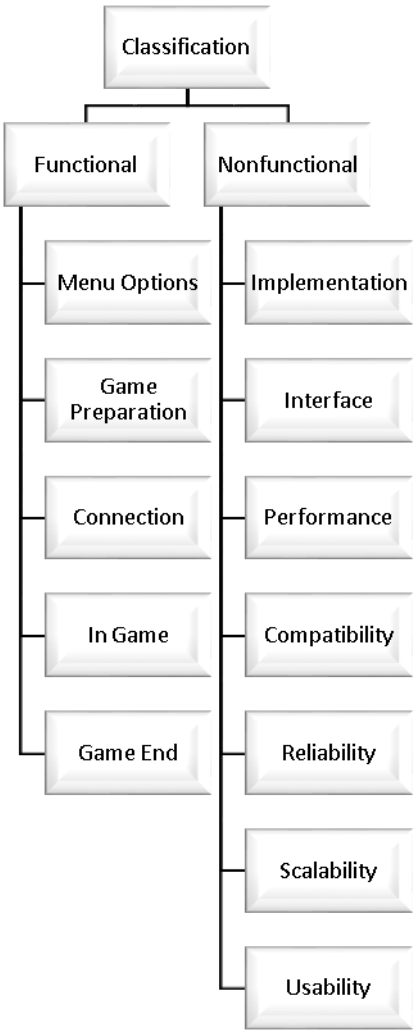


Figure 43: Requirement classification

The following table shows a brief explanation of the different categories of requirements defined above.

Type	Classification	Description
Functional	Menu options	Requirements about displaying the profile information, match history and leader board.
	Game preparation	Requirements about the process to prepare a game and notify the players that will be involved in the game.
	Connection	Requirements about the connection between the server and the mobile client, and the mobile client and the smartwatch.
	In game	Requirements about a current match with or without multiple players, hosted by a user of the TableMe application.

	Game end	Requirements about the finishing phases of a game.
Non-functional	Interface	Requirements that allow the flow between the application information and the user.
	Usability	Requirements that provide the system's way to use for the users.
	Performance	Requirements of the performance expected from the applications.
	Compatibility	Requirements of the compatibility dependencies needed to deploy and execute the system.
	Reliability	Requirements that allow support in failure cases.
	Scalability	Requirements that guide the system on how must grow as the user grow too.
	Implementation	Requirements of the system implementation.

Table 6: Requirement classification specification

POLYTECHNIC OF TURIN

Faculty of Engineering

Master's Degree

in Computer Engineering

Use case Specification

Andres Camilo Jimenez Vargas

October 2016

Index

Name of the use case:	UC-035 Reconnect Wear Device		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**17/04/2016**		
Notes:		**Modification date:**	**17/04/2016**
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1.The wear device is disconnected 2.The server is notified from the disconnection. 3.Wait for reconnection. 4.1. If connected 4.1.1. Register device in the server and restore the game's information. 4.2. If not connected. 4.2.1. Wait for connection.		
Basic Path Conventions: • Mobile application. • Wear application. • Server. **Restrictions**	1.The wear device is disconnected 2.The server is notified from the disconnection. 3.Wait for reconnection. 4.1. If connected 4.1.1. Register device in the server and restore the game's information. 4.2. If not connected. 4.2.1. Wait for connection.		
Invariant (Approved)	**A wear device has been disconnected from the mobile.**		
Invariant (Approved)	**The user is in a current game.**		

Name of the use case:	UC-036 Reconnect Player		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**17/04/2016**	**Modification date:**	**26/04/2016**

Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Prepare the mobile device for reconnection. 2.1. If reconnected 2.1.1. Register device and restore the score of the game. 2.2. if not connected. 2.2.1. Waiting for connection event.		
Restrictions			
Invariant (Approved)	**A player has disconnected from the server**		

Name of the use case:	UC-038 Request user information		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**28/04/2016**	**Modification date:**	**28/04/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The user request a wear synchronization. 2. Request the user update to the JOL Table Me server 3. Receive server answer 4. Send information to synchronize		
Restrictions			
Invariant (Approved)	**A user paired the wear device**		

Name of the use case:	UC-039 Request match history		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**28/04/2016**	**Modification date:**	**28/04/2016**
Notes:			
Scenarios:			

Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The user request a wear synchronization. 2. Request the match history to the JOL Table Me server 3. Receive server answer 4. Send information to synchronize
Restrictions	
Invariant (Approved)	**The user paired the wear device**

Name of the use case:	UC-040 Request leader board		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**28/04/2016**	**Modification date:**	**28/04/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The user request a wear synchronization. 2. Request the leader board to the JOL Table Me server 3. Receive server answer 4. Send information to synchronize		
Restrictions			
Invariant (Approved)	**The user paired wear device**		

Name of the use case:	UC-001 Synchronize wear data		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**19/11/2005**	**Modification date:**	**28/04/2016**
Notes:			
Scenarios:			
Basic Path	1. Open the application.		

Conventions: • Mobile application. • Wear application. • Server.	2. Search for Bluetooth connected nodes. 3. Compile all user information and put it in a data map. 4. Send information to the wear device.
Restrictions	
Pre-condition (Proposed)	**The player opened the application.**

Name of the use case:	UC-002 Register device to the server		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**20/11/2005**	**Modification date:**	**24/02/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Send a token request to the GCM service. 2. Receive a JSON response and extract the token. 3. Send to the server the token given by the GCM service.		
Restrictions			
Pre-condition (Approved)	**The user opens the application at least once before.**		

Name of the use case:	UC-003 Create a match		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions:	1. The user presses the button to start a new game. 2. A new instance of the game is instantiated. 3. UC-006 Select blue team.		

• Mobile application. • Wear application. • Server.	4. UC-005 Select red team. 5. UC-011 Start Match.
Restrictions	
Pre-condition (Approved)	**The user has registered to the server.**
Pre-condition (Approved)	**The user synchronized data with the wear application.**

Name of the use case:	UC-004 Send notification to a device		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Receive the teams in the match. 2. UC-031 Create match. 3. Send the notification to all team players using GCM service.		
Restrictions			
Pre-condition (Approved)	**All the players must be registered on the server.**		

Name of the use case:	UC-005 Select read team		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path	1. The mobile application sends a request to the server to retrieve the		

Conventions: • Mobile application. • Wear application. • Server.	list of opponents. 2. The player clicks the button of the defense position of red team. 3. The player selects the player in the list. 4. The player clicks the button of the offense position of red team. 5. The player selects the player in the list. 6. UC-011 Start match.
Restrictions	
Pre-condition (Approved)	**All the players must be registered on the server.**

Name of the use case:	UC-006 Select blue team		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**09/03/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The mobile application sends a request to the server to retrieve the list of opponents. 2. The player clicks the button of the defense position of blue team. 3. The player selects the player in the list. 4. The player clicks the button of the offense position of blue team. 5. The player selects the player in the list. 1. UC-011 Start match.		
Restrictions			
Pre-condition (Approved)	**All the players must be registered on the server.**		

Name of the use case:	UC-007 Use GCM service for a token		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			

Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Get the instance id of the application. 2. Send a token request to the GCM service with the id of the application. 3. Retrieve the token. 4. UC-002 Register device to the server.
Restrictions	
Pre-condition (Approved)	**The user opens the application.**
Post-condition (Approved)	**A unique token for the device from the GCM service.**

Name of the use case:	UC-008 Receive challenge notification		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The notification is received in the mobile. 2. The mobile sends a message to the wear application with the notification. 3. UC-009 Show challenge notification.		
Restrictions			
Pre-condition (Approved)	**The server sends a notification to the GCM service to the teams.**		

Name of the use case:	UC-009 Show challenge notification		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			

Scenarios:	
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Receive the message from the mobile in a listener with the game id. 2. Prepare to start the game of the contender in an intent with the game id. 3. Create the notification with the data provided from the mobile. 4. Rise the notification in the wear device. 5. UC-010 Accept challenge.
Restrictions	
Pre-condition (Approved)	**The user synchronized with the wear application.**
Pre-condition (Approved)	**The mobile received the notification from the GCM service.**
Pre-condition (Approved)	**The mobile sent a message to the wear device with the notification.**

Name of the use case:	UC-010 Accept challenge		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Alternate Conventions: • Mobile application. • Wear application. • Server.	1. The user sweeps the notification to the left for the link to open the game. 2. The user taps the button to open the game. 3. UC-029 Send decline game. 4. Close the activity.		
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The user sweeps the notification to the left for the link to open the game. 2. The user clicks the button to start the game. 3. UC-027 Notify player ready. 4. A new activity rises and wait for the opponent to be ready.		
Restrictions			
Pre-condition (Approved)	**All players must be registered on the server**		
Pre-condition	**The challenge notification was shown.**		

(Approved)

Name of the use case:	UC-011 Start Match		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**09/03/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Create a game instance with the teams. 2. UC-004 Send notification to device. 3. Receive the game id from the server.		
Restrictions			
Pre-condition (Approved)	**The match creator selected all team's players.**		
Pre-condition (Approved)	**All players are registered in the server.**		

Name of the use case:	UC-015 Manage Score		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**26/04/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The player clicks the button to score a goal, dismiss or auto goal. 2. The wear application sends a message to the phone to inform a goal or auto goal. 3. UC-018 Update score. 4. The mobile application receive the score updated with the game id. 5. The mobile application sends the score to the wear device. 6. UC-021 Display score.		

Restrictions	
Pre-condition (Approved)	**A match already started.**
Pre-condition (Approved)	**All players are registered in the server.**

Name of the use case:	UC-018 Update score		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**24/02/2016**
Notes:			
Scenarios:			
Alternate Conventions: • Mobile application. • Wear application. • Server.	1. Receive a goal or auto goal notification with the id of whom scored it. 2. Search the game instance. 3. Send to the registered sockets the score of the game. 4. Check if no one has score six point. 5. UC-018 Display score. 6. UC-020 Select winner.		
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Receive a goal or auto goal notification with the id of whom scored it. 2. Search the game instance. 3. Send to the registered sockets the score of the game. 4. Check if no one has score six point. 5. UC-018 Display score.		
Restrictions			
Pre-condition (Approved)	**A match has already started.**		
Pre-condition (Approved)	**A user scored a goal.**		
Post-condition (Approved)	**All the related devices are notified of the score.**		
Pre-condition (Approved)	**Game id.**		

Name of the use case:	UC-019 Display score		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Receive an updated score from the mobile. 2. Update the user interface.		
Restrictions			
Pre-condition (Approved)	**A match has already started.**		
Pre-condition (Approved)	**The mobile application sent a message with the updated score to the wear application.**		

Name of the use case:	UC-020 Select winner		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**24/02/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. A user scored six points. 2. Send a message to the mobiles notifying to each player if it is has won or lost the match. 3. Finish the game instance. 4. UC-021 Display winner or loser.		
Restrictions			
Pre-condition (Approved)	**A match has already started.**		
Pre-condition (Approved)	**A player scored a goal.**		

Name of the use case:	UC-021 Display winner or loser		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**24/02/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The mobile receive the information of the winner or the loser. 2. The mobile save the match in a history 3. The mobile send a message to the wear device with the information. 4. The wear device receives information and display it in the user interface.		
Restrictions			
Pre-condition (Approved)	**A match has already started.**		
Pre-condition (Approved)	**The mobile sent the information of a winner or loser of the match.**		

Name of the use case:	UC-024 Show profile data		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The player clicks the option of profile in the menu. 2. The wear application checks the persisted data of the profile. 3. The wear application shows a new screen with the profile data.		
Restrictions			
Pre-condition (Approved)	**The user synchronized the wear application.**		

Name of the use case:	UC-025 Show leaderboard		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The player clicks the button in the menu to see the leader board. 2. The wear application sends the request to the mobile application. 3. The mobile application sends the request to the server application. 4. UC-034 Send latest leader board 5. Receive the leader board. 6. Display the list of players of the leader board.		
Restrictions			
Pre-condition (Approved)	**The player registered in the server.**		

Name of the use case:	UC-026 Show matches history		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The player clicks the option of match history in the menu. 2. The wear application checks the persisted data of the match history. 3. The wear application shows a new screen with the list of matches organized from the newest to the oldest.		
Restrictions			
Pre-condition (Approved)	**The user synchronized the wear application.**		

Name of the use case:	UC-027 Notify player ready		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Receive the message from the wear application. 2. Attach to the message the game id and the player's id. 3. Send the message to the server application.		
Restrictions			
Pre-condition (Approved)	**All players are registered in the server.**		
Pre-condition (Approved)	**Wear application sent that the player is ready to play.**		

Name of the use case:	UC-028 Register player ready		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. The server application receives a player register request. 2. The server application searches the game and the player of the registration. 3. The server application set in ready the player. 4. If all players are ready. 5. Start match.		
Alternate Conventions: • Mobile application. • Wear application. • Server.	1. The server application receives a player register request. 2. The server application searches the game and the player of the registration. 3. The server application set in ready the player. 4. If not all players are ready. 5. Wait for the other players to be ready to start the match.		

Restrictions	
Pre-condition (Approved)	**All players registered in the server.**
Pre-condition (Approved)	**Game id.**
Pre-condition (Approved)	**Player id.**
Pre-condition (Approved)	**A match was created.**

Name of the use case:	UC-030 Register player decline		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**07/05/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. A player decline a game. 2. Notify host 3. UC-032 Cancel match		
Restrictions			
Invariant (Approved)	**A game challenge was sent to a set of players**		
Invariant (Approved)	**All players are registered**		

Name of the use case:	UC-031 Create match		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**09/03/2016**
Notes:			
Scenarios:			

Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. A new game instance is created with a unique id. 2. All information of the teams is stored in the game instance. 3. Wait for the player to be ready.
Restrictions	
Pre-condition (Approved)	**All players must be registered in the server.**
Pre-condition (Approved)	**A notification request arrived from a mobile application.**

Name of the use case:	UC-032 Cancel match		
Author:	**Andres Camilo Jimenez Vargas**		
Creation date:	**24/02/2016**	**Modification date:**	**07/05/2016**
Notes:			
Scenarios:			
Basic Path Conventions: • Mobile application. • Wear application. • Server.	1. Send message to finish game to players. 2. Send message to finish game to host.		
Restrictions			
Invariant (Approved)	**All players are registered**		
Invariant (Approved)	**A player declines a game**		

POLYTECHNIC OF TURIN

Faculty of Engineering

Master's Degree

in Computer Engineering

Functional Requirements Specification

Andres Camilo Jimenez Vargas

October 2016

Index

REQ-001 Send profile information mobile

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.4***
	***Description:* The mobile application must send the profile information to the wear application.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-080 Request profile***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-001**		

REQ-002 Save match history mobile

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must store the match history of each created.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-026 Start game server***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-021**		

REQ-003 Send match history mobile

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.4***
	***Description:* The mobile application must send the match history to the wear application.**		
	Dependencies:		

	Type of element: **Requirement** ***Dependency:*** **Requirement:** ***REQ-081 Request match history***
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-001**

REQ-004 Send leader board mobile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must send the latest leader board to the wear application.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-005 Request leader board***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-025**		

REQ-005 Request leader board

«Functional»	***State: finished***	***Difficulty:*** **Easy**	***Priority: 0.4***
	Description: **The mobile application must request the latest leader board to the server**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-025**		

REQ-006 Show profile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The wear application must allow to a player to show**		

	the profile information of the player.
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-001 Send profile information mobile***
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-024**

REQ-007 Show match history

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The wear application must allow to a player to show the match history of the player.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-003 Send match history mobile***		
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-026**		

REQ-008 Show leader board

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The wear application must allow to a player to show the latest leader board.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-004 Send leader board mobile***		
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-025**		

REQ-009 Select blue offense player

«Functional»	***State: finished***	***Difficulty:* Hard**	***Priority: 0.6***
	***Description:* The mobile application must allow the match creator to select the player of the blue team that will play in offense position.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-013 Create match***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-006**		

REQ-010 Get GCM token

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must use the GCM service to retrieve the token of the device.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-011 Send registration***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-007**		

REQ-011 Send registration

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.3***
	***Description:* The mobile application must register the device with the token in the server.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-002**		

REQ-012 Register device

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.4***
	***Description:* The server application must store the token and the id of the registered devices.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-010 Get GCM token***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-002**		

REQ-013 Create match

«Functional»	***State: finished***	***Difficulty:* Hard**	***Priority: 0.6***
	***Description:* The mobile application must allow the match creator to create a new match.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-016 Send notification request*** **Requirement:** ***REQ-011 Send registration*** **Requirement:** ***REQ-023 Start game wear*** **Requirement:** ***REQ-026 Start game server*** **Requirement:** ***REQ-015 Retrieve players***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-003**		

REQ-014 Select red defines player

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must allow the match creator**		

	select the player of the red team that will play in defines position.
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-013 Create match***
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-005**

REQ-015 Retrieve players

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must retrieve from the server a list of players.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-005**		

REQ-016 Send notification request

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must allow the match creator send the notification request of the challenge to the players to start the game.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-009 Select blue offense player*** **Requirement:** ***REQ-011 Send registration*** **Requirement:** ***REQ-050 Select red offense player*** **Requirement:** ***REQ-014 Select red defines player*** **Requirement:** ***REQ-017 Select blue defines player***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-011**		

REQ-017 Select blue defines player

«Functional»	***State: finished***	***Difficulty:* Hard**	***Priority: 0.6***
	***Description:* The mobile application must allow the match creator select the player of the blue team that will play in defines position.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-013 Create match***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-006**		

REQ-019 Save game request

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must store the game request with the information of each team.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-026 Start game server*** **Requirement: *REQ-016 Send notification request*** **Requirement: *REQ-013 Create match***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-031**		

REQ-020 Send notification to rise

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must notify the wear application when a challenge notification arrives.**		
	Dependencies: ***Type of element:* Requirement** **Requirement:** ***REQ-019 Save game request*** **Requirement:** ***REQ-026 Start game server*** **Requirement:** ***REQ-021 Rise notification in wear***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-008**		

REQ-021 Rise notification in wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must rise a notification to accept the challenge.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-009**		

REQ-022 Join game wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must start the game when a player clicks the button on the challenge notification.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-010**		

REQ-023 Start game wear

«Functional»	*State: finished*	*Difficulty:* **Hard**	*Priority: 0.6*
	Description: **The mobile application must notify the wear device to start the game.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-022 Join game wear***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-011**		

REQ-024 Ready player wear

«Functional»	*State: finished*	*Difficulty:* **Medium**	*Priority: 0.5*
	Description: **The wear application must notify the mobile application that the player is ready**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-022 Join game wear***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-027**		

REQ-025 Ready player mobile

«Functional»	*State: finished*	*Difficulty:* **Medium**	*Priority: 0.5*
	Description: **The mobile application must notify the server that the player is ready**		
	Dependencies: ***Type of element:*** **Requirement**		

	Dependency: **Requirement:** ***REQ-022 Join game wear***
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-027**

REQ-026 Start game server

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The server application must start a game when at least the host or any player is ready**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-022 Join game wear***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-018**		

REQ-027 Manage game by id

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The server application must manage a game based on the game id and the player's id.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-026 Start game server***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-018**		

REQ-028 Add goal wear

«Funct	***State: finished***	***Difficulty:*** **Hard**	***Priority: 0.6***

ional»	***Description:*** **The wear application must allow the player add a goal in a match.**
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-030 Send score update wear***
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-018**

REQ-029 Subtract goal wear

«Funct ional»	***State: finished***	***Difficulty:*** **Hard**	***Priority: 0.6***
	Description: **The wear application must allow the player subtract a goal in a match.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-030 Send score update wear***		
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-015**		

REQ-030 Send score update wear

«Funct ional»	***State: finished***	***Difficulty:*** **Easy**	***Priority: 0.4***
	Description: **The wear application must send the score to be updated to the mobile application.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-031 Send score update mobile***		
	Realization: ***Type of element:*** **Requirement** **Realization: Use case: UC-018**		

REQ-031 Send score update mobile

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must send the score to be updated to the server.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-033 Send score update***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-018**		

REQ-032 Update score received server

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must update the score when it receives a request to be updated.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-018**		

REQ-033 Send score update

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must send to the players wear application the updated score.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-032 Update score received server***		
	Realization:		

	Type of element: **Requirement** **Realization:** **Use case:** **UC-018**

REQ-034 Receive updated score mobile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.4***
	Description: **The mobile application must receive the updated score from the server.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-032 Update score received server***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-019**		

REQ-035 Send update score wear

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must send to the wear application the updated score.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-019**		

REQ-036 Display update wear

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The wear application must display the updated score in the player's game screen.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-035 Send update score wear***		

	Requirement: ***REQ-026 Start game server***
	Realization:
	Type of element: **Requirement**
	Realization: Use case: UC-019

REQ-037 Identify winner player server

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The server application must identify a winner when a player scores six points.**		
	Dependencies:		
	Type of element: **Requirement**		
	Dependency: **Requirement:** ***REQ-038 Notify mobile winner***		
	Requirement: ***REQ-026 Start game server***		
	Realization:		
	Type of element: **Requirement**		
	Realization: Use case: UC-020		

REQ-038 Notify mobile winner

«Functional»	***State: finished***	***Difficulty:*** **Easy**	***Priority: 0.4***
	Description: **The server application must notify the mobile application when a player won the game.**		
	Dependencies:		
	Type of element: **Requirement**		
	Dependency: **Requirement:** ***REQ-040 Receive winner mobile***		
	Realization:		
	Type of element: **Requirement**		
	Realization: Use case: UC-020		

REQ-039 Notify mobile lose

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.4***
	***Description:* The server application must notify the mobile application when a player lost the game.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-020**		

REQ-040 Receive winner mobile

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.3***
	***Description:* The mobile application must receive the server's notification of a winner.**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-021**		

REQ-041 Send winner wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must send a winner notification to the wear application.**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-040 Receive winner mobile***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-021**		

REQ-042 Display winner wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must display the winner layout if the player won the game.**		

	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-041 Send winner wear***
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-021**

REQ-043 Display lose wear

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The wear application must display the loser layout if the player lost the game.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-039 Notify mobile lose***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-021**		

REQ-044 Return home

«Functional»	***State: finished***	***Difficulty:*** **Easy**	***Priority: 0.3***
	Description: **The wear application must allow the user return home when the game ends.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-043 Display lose wear*** **Requirement:** ***REQ-042 Display winner wear***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-034**		

REQ-045 Deny game

«Functional»	*State: finished*	*Difficulty:* **Medium**	*Priority: 0.5*
	***Description:* The wear application must allow to deny a challenge when a notification had risen.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-010**		

REQ-046 Deny game server

«Functional»	*State: finished*	*Difficulty:* **Medium**	*Priority: 0.5*
	***Description:* The mobile application must send a challenge deny to the server.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-030**		

REQ-047 Deny challenge mobile server

«Functional»	*State: finished*	*Difficulty:* **Medium**	*Priority: 0.5*
	***Description:* The server application must deny a challenge when a mobile application notifies it.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-030**		

REQ-048 Receive lose mobile

«Functional»	*State: finished*	*Difficulty:* **Easy**	*Priority: 0.3*
	***Description:* The mobile application must receive the server's notification of a loser.**		
	Realization: ***Type of element:*** **Requirement**		

	Realization: **Use case:** **UC-021**

REQ-049 Send lose wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must send a loser notification to the wear application.**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-021**		

REQ-050 Select red offense player

«Functional»	***State: finished***	***Difficulty:* Hard**	***Priority: 0.6***
	***Description:* The mobile application must allow the user select the player of the red team that will play in offense position.**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-013 Create match***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-005**		

REQ-052 Recognize wear connection

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must recognize when the wear device is connected to the mobile**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-035**		

REQ-053 Recognize wear disconnection

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must recognize when the wear device is disconnected of the mobile device**		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-035**		

REQ-054 Notify server wear disconnection

«Functional»	***State: finished***	***Difficulty:* Easy**	***Priority: 0.4***
	***Description:* The mobile application must notify the server when the wear device is disconnected**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-053 Recognize wear disconnection***		
	Realization: ***Type of element:* Requirement** **Realization: Use case: UC-036**		

REQ-057 Notify reconnection wear

«Functional»	***State: finished***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The mobile application must notify to the wear the score of the game when a wear device is connected to the mobile**		
	Dependencies: ***Type of element:* Requirement** ***Dependency:* Requirement: *REQ-052 Recognize wear connection*** **Requirement: *REQ-053 Recognize wear disconnection***		
	Realization:		

	Type of element: **Requirement** **Realization:** **Use case:** **UC-036**

REQ-058 Notify score connection mobile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must notify the actual score of a game to any participant player that connect in the server**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-057 Notify reconnection wear*** **Requirement:** ***REQ-034 Receive updated score mobile***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-038**		

REQ-059 Notify score reconnection mobile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must notify the actual score of a game to any participant player that reconnect in the server**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-024** **Use case:** **UC-025** **Use case:** **UC-026**		

REQ-079 Send auto goal

«Functional»	***State: finished***	***Difficulty:*** **Hard**	***Priority:0.6***
	Description: **The wear application must allow the player to score an auto goal in a match.**		
	Dependencies:		

	Type of element: **Requirement** ***Dependency:*** **Requirement:** ***REQ-030 Send score update wear***
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-015**

REQ-080 Request profile

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must request the latest user profile to the server**		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-024**		

REQ-081 Request match history

«Functional»	***State: finished***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must request the latest user match history to the server**		
	Dependencies: ***Type of element:*** **Requirement** ***Dependency:*** **Requirement:** ***REQ-002 Save match history mobile***		
	Realization: ***Type of element:*** **Requirement** **Realization:** **Use case:** **UC-026**		

POLYTECHNIC OF TURIN

Faculty of Engineering

Master's Degree

in Computer Engineering

Non Functional Requirements Specification

Andres Camilo Jimenez Vargas

October 2016

Index

1. Usability

1.1. REQ-064 Interface design by Google

«Usability»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.4***
	***Description:* The wear application must follow the user interface design rules form google.**		

2. Performance

2.1. REQ-066 Connection time mobile

«Performance»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The connection of the mobile application with the server must be done in a try during from 1 to 10 seconds**		

2.2. REQ-067 Notification GCM

«Performance»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The notification of the creation of a game must be done in a try during 1 to 10 seconds.**		

2.3.REQ-068 Receive incoming connections

«Performance»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must receive any incoming connections to an existing game in progress.**		

2.4. REQ-069 Game architecture

«Performance»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must save the instances of the**		

	devices connected in a persistent way following the architecture of Ashwin R. Bharambe, Jeff Pang and Srinivasan Seshan proposed for a multi player game.

2.5. REQ-070 GCM architecture

«Performance»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must save the GCM tokens from the users following the architecture proposed by Google in the google Cloud Messaging implementation.**		

3. Usability

3.1. REQ-071 Manual

«Usability»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The application must provide a user manual for the wear application usage.**		

4. Scalability

4.1. REQ-072 Players connected

«Scalability»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The server application must provide support to multiple games conformed by maximum 5 players connected playing simultaneously.**		

5. Compatibility

5.1.REQ-073 Wear compatibility

«Comp	***State: finish***	***Difficulty:* Medium**	***Priority: 0.4***

atibility »	***Description:*** **The wear application must be compatible to android wear devices from Kit Kat version to the actual version Lollipop**

5.2. REQ-074 Mobile compatibility

«Comp atibility »	***State: finish***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The mobile application must be compatible to android devices from Jelly Bean version to the actual version Lollipop**		

6. Reliability

6.1. REQ-075 Player recover

«Reliab ility»	***State: finish***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The system must recover the score of a game of a player that reconnect to a game.**		

7. Implementation

7.1. REQ-076 Server implementation

«Imple mentati on»	***State: finish***	***Difficulty:*** **Medium**	***Priority: 0.5***
	Description: **The server application must be implemented using node js.**		

7.2. REQ-065 Application integration

«Imple mentati on»	***State: finish***	***Difficulty:*** **Hard**	***Priority: 0.7***
	Description: **The wear and mobile application must be integrated with the latest version of the Table Me application**		

8. Interface

8.1. REQ-077 Design interfaces round wear

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must design the interfaces for the round smartwatches**		

8.2. REQ-078 Design interfaces square wear

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must design the interfaces for the square smartwatches**		

8.3. REQ-060 Interface menu list

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must show as home page list of main features to choose**		

8.4. REQ-061 Interface profile wear

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must show after click the profile item in the list the user's facebook logo, the victories and**		

	loses and the elo.

8.5. REQ-062 Interface history wear

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must show after click the history item in the list the list of matches of the user with detailed information**		

8.6. REQ-063 Show leader board wear

«Interface»	***State: finish***	***Difficulty:* Medium**	***Priority: 0.5***
	***Description:* The wear application must show after click the leader board item in the list, the latest leader board updated.**		

7.5. Requirement priorization

ID	Benefit	Penalization	Total value	Value %	Cost	Cost %	Risk	Risk %	Priority	Complexity
REQ-065 Application integration	9	9	18	1.7	9	1.8	3	0.7	0.713	Hard
REQ-013 Create match	8	9	17	1.6	8	1.6	4	0.9	0.668	Hard
REQ-050 Select red offense player	8	9	17	1.6	7	1.4	5	1.1	0.663	Hard
REQ-009 Select blue offense player	8	9	17	1.6	7	1.4	5	1.1	0.663	Hard
REQ-079 Send auto goal	8	9	17	1.6	7	1.4	5	1.1	0.663	Hard
REQ-028 Add goal wear	8	7	15	1.5	8	1.6	3	0.7	0.647	Hard
REQ-029 Subtract goal wear	8	7	15	1.5	8	1.6	3	0.7	0.647	Hard
REQ-023 Start game wear	9	9	18	1.7	7	1.4	6	1.3	0.646	Hard
REQ-017 Select blue defense player	7	9	16	1.5	7	1.4	5	1.1	0.624	Hard
REQ-075 Player recover	7	8	15	1.5	8	1.6	4	0.9	0.590	Medium
REQ-014 Select red defense player	6	9	15	1.5	7	1.4	5	1.1	0.585	Medium
REQ-022 Join game wear	8	8	16	1.5	7	1.4	6	1.3	0.574	Medium
REQ-024 Ready player wear	8	8	16	1.5	7	1.4	6	1.3	0.574	Medium
REQ-025 Ready player mobile	8	8	16	1.5	7	1.4	6	1.3	0.574	Medium
REQ-026 Start game server	8	9	17	1.6	7	1.4	7	1.5	0.564	Medium
REQ-027 Manage game by id	9	9	18	1.7	8	1.6	7	1.5	0.559	Medium
REQ-037 Identify winner player server	7	8	15	1.5	6	1.2	7	1.5	0.534	Medium
REQ-052 Recognize wear connection	7	8	15	1.5	6	1.2	7	1.5	0.534	Medium
REQ-053 Recognize wear disconnection	7	8	15	1.5	6	1.2	7	1.5	0.534	Medium
REQ-059 Notify score reconnection mobile	8	8	16	1.5	7	1.4	7	1.5	0.531	Medium
REQ-069 Game architecture	6	6	12	1.2	5	1.0	6	1.3	0.505	Medium
REQ-068 Receive incoming connections	6	6	12	1.2	5	1.0	6	1.3	0.505	Medium
REQ-020 Send notification to rise	7	5	12	1.2	5	1.0	6	1.3	0.505	Medium
REQ-066 Connection time mobile	6	7	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-067 Notification GCM	6	7	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-016 Send notification request	7	6	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-045 Deny game	7	6	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-046 Deny game server	7	6	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-047 Deny challenge mobile server	7	6	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-058 Notify score connection mobile	8	5	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-002 Save match history mobile	9	4	13	1.3	6	1.2	6	1.3	0.503	Medium
REQ-076 Server implementation	5	9	14	1.4	7	1.4	6	1.3	0.502	Medium
REQ-004 Send leader board mobile	7	7	14	1.4	7	1.4	6	1.3	0.502	Medium
REQ-034 Receive updated score mobile	6	8	14	1.4	6	1.2	7	1.5	0.499	Medium
REQ-003 Send match history mobile	7	7	14	1.4	6	1.2	7	1.5	0.499	Medium
REQ-061 Interface profile wear	7	8	15	1.5	7	1.4	7	1.5	0.498	Medium
REQ-062 Interface history wear	7	8	15	1.5	7	1.4	7	1.5	0.498	Medium
REQ-006 Show profile	8	8	16	1.5	8	1.6	7	1.5	0.497	Medium
REQ-007 Show match history	8	8	16	1.5	8	1.6	7	1.5	0.497	Medium
REQ-008 Show leader board	8	8	16	1.5	8	1.6	7	1.5	0.497	Medium
REQ-063 Interface leader board wear	8	8	16	1.5	8	1.6	7	1.5	0.497	Medium
REQ-073 Wear compatibility	5	6	11	1.1	6	1.2	5	1.1	0.467	Medium
REQ-074 Mobile compatibility	5	6	11	1.1	6	1.2	5	1.1	0.467	Medium
REQ-015 Retrieve players	7	6	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-042 Display winner wear	6	7	13	1.3	7	1.4	6	1.3	0.466	Medium

REQ-043 Display lose wear	6	7	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-049 Send lose wear	6	7	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-031 Send score update mobile	6	7	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-032 Update score received server	6	7	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-057 Notify reconnection wear	7	6	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-080 Request profile	7	6	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-081 Request match history	7	6	13	1.3	7	1.4	6	1.3	0.466	Medium
REQ-041 Send winner wear	5	7	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-010 Get GCM token	6	6	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-019 Save game request	6	6	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-036 Display update wear	6	6	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-060 Interface menu list	6	6	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-064 Interface design by Google	6	6	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-021 Rise notification in wear	7	5	12	1.2	6	1.2	6	1.3	0.465	Medium
REQ-033 Send score update	6	8	14	1.4	7	1.4	7	1.5	0.465	Medium
REQ-035 Send update score wear	6	7	13	1.3	6	1.2	7	1.5	0.463	Medium
REQ-072 Players connected	6	4	10	1.0	4	0.8	6	1.3	0.461	Medium
REQ-030 Send score update wear	6	7	13	1.3	7	1.4	7	1.5	0.432	Easy
REQ-005 Request leader board	7	5	12	1.2	7	1.4	6	1.3	0.430	Easy
REQ-001 Send profile information mobile	7	4	11	1.1	6	1.2	6	1.3	0.426	Easy
REQ-012 Register device	5	6	11	1.1	6	1.2	6	1.3	0.426	Easy
REQ-038 Notify mobile winner	5	6	11	1.1	6	1.2	6	1.3	0.426	Easy
REQ-039 Notify mobile lose	5	6	11	1.1	6	1.2	6	1.3	0.426	Easy
REQ-077 Design interfaces round wear	6	4	10	1.0	6	1.2	5	1.1	0.424	Easy
REQ-078 Design interfaces square wear	6	4	10	1.0	6	1.2	5	1.1	0.424	Easy
REQ-070 GCM architecture	5	6	11	1.1	5	1.0	7	1.5	0.423	Easy
REQ-054 Notify server wear disconnection	7	6	13	1.3	7	1.4	8	1.7	0.402	Easy
REQ-011 Send registration	5	6	11	1.1	7	1.4	6	1.3	0.395	Easy
REQ-040 Receive winner mobile	5	5	10	1.0	6	1.2	6	1.3	0.387	Easy
REQ-048 Receive lose mobile	5	5	10	1.0	6	1.2	6	1.3	0.387	Easy
REQ-044 Return home	4	4	8	0.8	5	1.0	5	1.1	0.372	Easy
REQ-071 Manual	4	4	8	0.8	4	0.8	7	1.5	0.334	Easy
Total			**1034**		**503**		**460**			

7.6. Installation Manual

Installation Manual

1. Server on Node JS:

The deployment and installation of the server will require for recommendation the Node JS environment from version 4.4.4 onwards. After the installation it is required the following libraries:

- **Socket IO**, for installation is necessary to type the following command “npm install socket.io”.
- **Google Cloud Messaging,** for installation is necessary to type the following command “npm install node-gcm”

For the update process of the information for the usage of the Google Cloud Messaging service. In this step is necessary to access to the Google Console Developer using the following procedure in the link. There you need to fill in with the name of the project in the console and the name of the package “it.telecomitalia.tableme”. Finally, it will create automatically the Server API Key and the Sender Id, this information must be updated in the server at the index.js with the API Key and in the application in the Quick references class updating the server id.

After that the server will deploy using the file system creating two folders, Config to save the data of the games ids, and Devices, where all the users that connect to the server will be registered.

2. Android application and Android Wear:

For the deployment of and installation of the libraries for the Table Me application in Android and Android Wear are done in the gradle build of the application adding the following dependencies:

- **Wearable API**, compile 'com.google.android.gms:play-services-wearable:8.3.0'
- **Socket IO**, for this library the application contains the version 0.7.0 for Android that is compatible with the version 1.0 onwards of the JavaScript implementation. This library was developed by Naoyuki Kanezawa, if a new version is needed can be downloaded from Naoyuki repository:
    - https://github.com/socketio/socket.io-client-java

  And would require the following dependency in the gradle build, where “x” is the version of the library:

    - compile('io.socket:socket.io-client:x) {

        exclude group: 'org.json', module: 'json'

      }

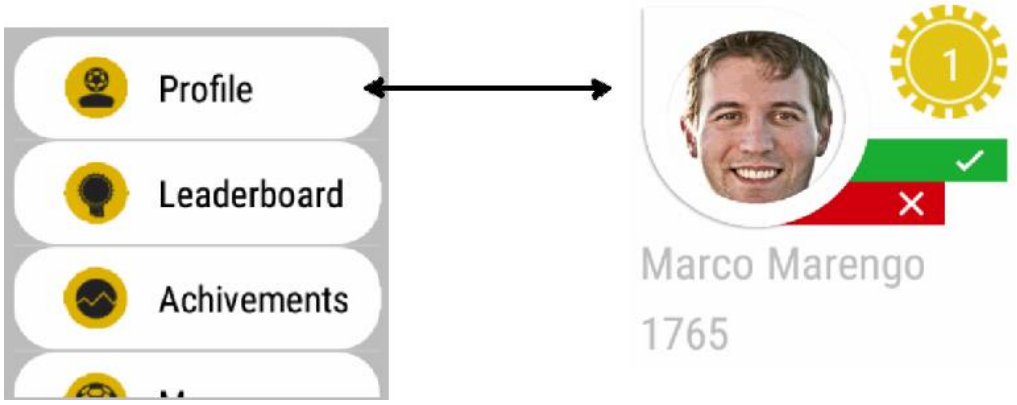
Important: If any changes are performed to the gradle of the app or the wear modules, it is necessary to update them with the same information. To be able to be compatible between the phone and the smartwatch, the gradles must provide the same

configurations for the android part such as compile Sdk version, build tools version, signing configs, default config, build types and dex options.

Finally, for the interfaces developed for the smartwatch application has two layouts for the different types of watches such as the square or the round ones. For this, it has been used watch stub views that allow to set which layout will be displayed if is used in a device with a round or square screen.

2.1. Profile information on the smartwatch

After opening the application, the user can use a list of option to select about the user's profile information and the game. The profile option shows in a similar way as in the Table Me application the basic information of the user. Firstly, the photo, the position in the leader board inside a badge, two bars that indicate proportionally the number of victories (green bar) and number of lost games (red bar). The at the end the name of the user and his/her ELO score.



2.2. Gameplay on the smartwatch

After a user that will host a game organize and create the team, a notification is issued to the participants of the match will receive a notification in their phones and to their smartwatches if they have one. At the watch the following notification is displayed.

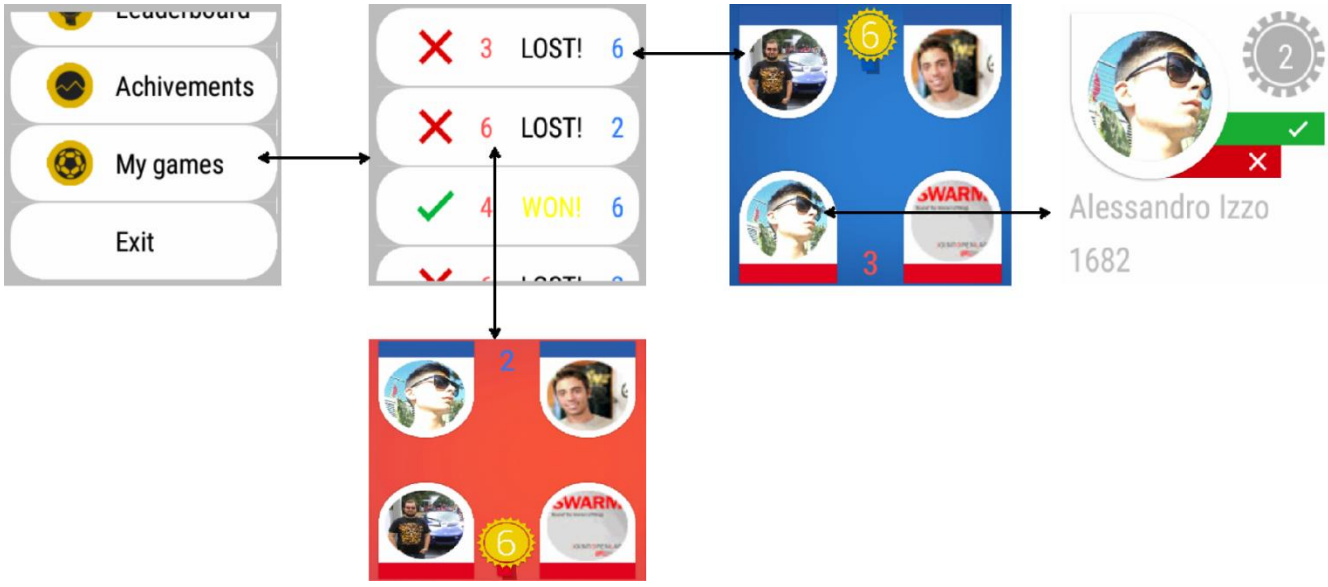
Then, when the user start playing the game, the screen with the buttons where the player can interact with the score is provided, with a button to add a goal in the center, a dismiss goal button at the left bottom and an auto goal button at the right bottom. The user can

interact with them and the score is displayed in the results displayed in top of the screen with its respective colors of the team.

Finally, in each case the users of a team win or lose a match the following screens are displayed in the smartwatches using a base badge with their goals scored.

2.3. Match history on the smartwatch

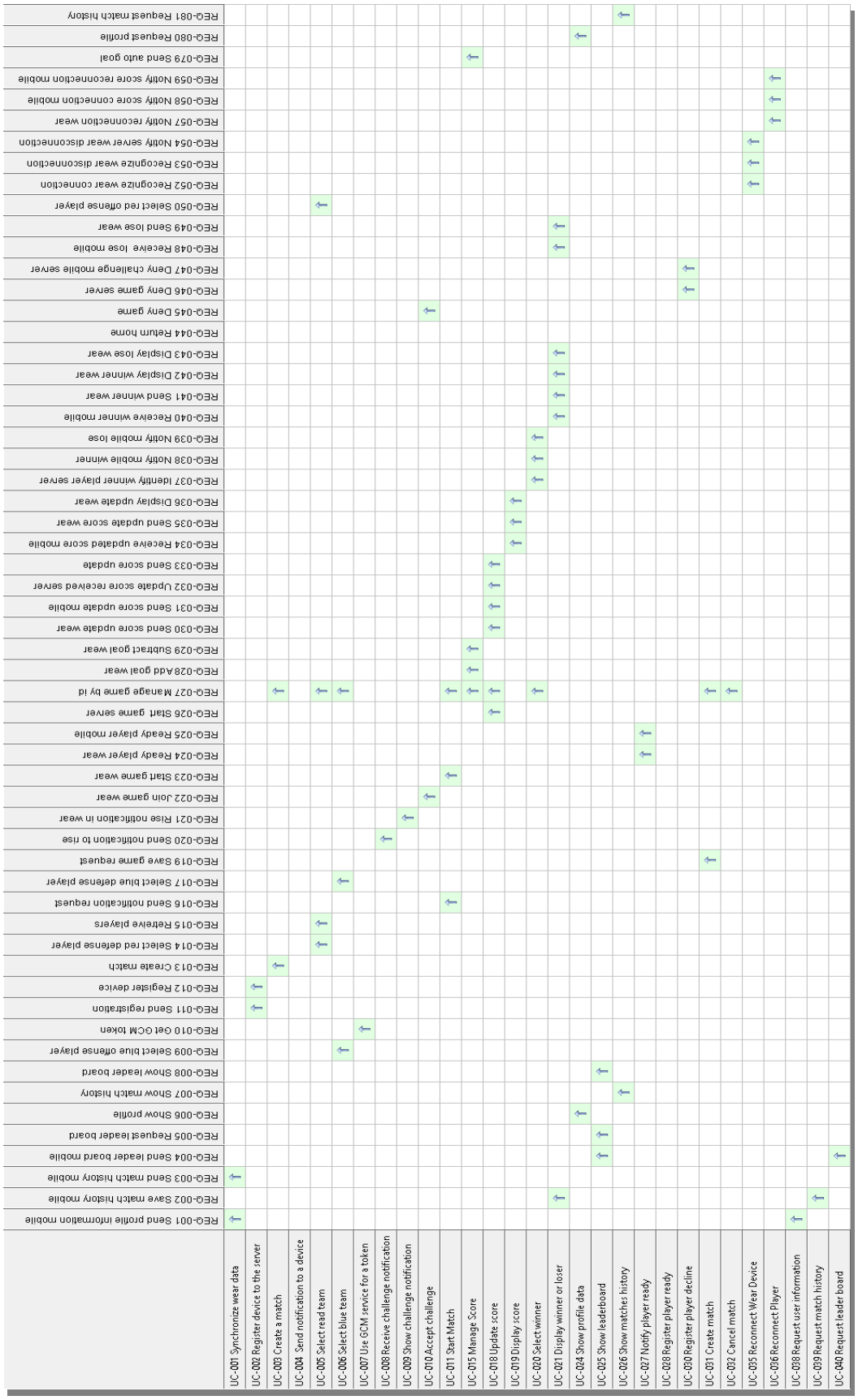
When the match history option is selected, the user is provided with a brief list of the last ten matches that he/she was part of. Each match is signaled with a cross and the message "Lost!" if the player lost that game, or a check and the message "Won!" if the player won the game. If the player wants to know a more detailed information of the match, it is possible to click an item on the list and will provide a screen with the background color of the victorious team, the score, and the photo of the participants. If the user wants to see the players profile information, he/she can click on the image and the profile information will be displayed.



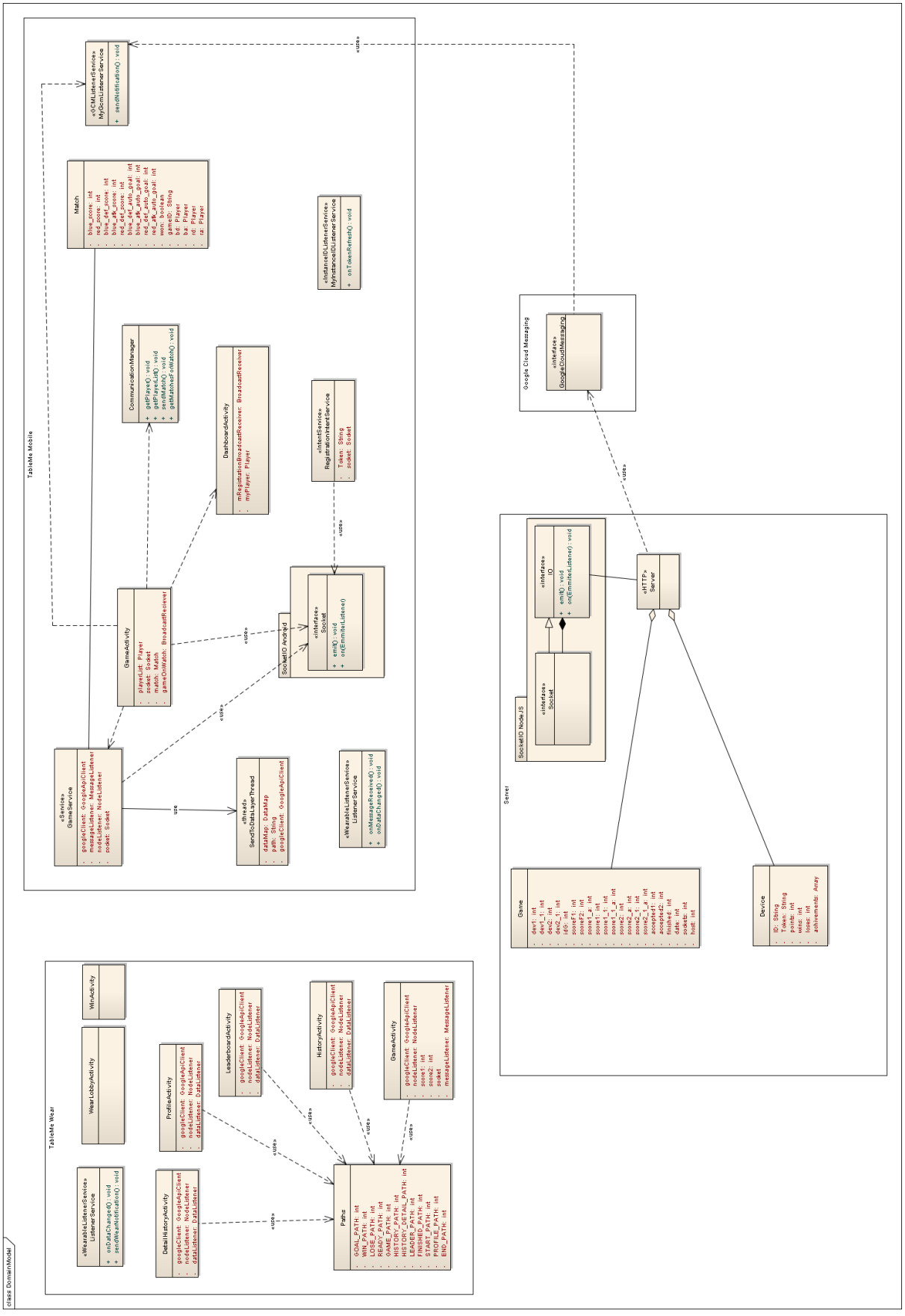
2.4. Leader board on the smartwatch

When the leader board is selected a list of the ten first places in the leaderboard is loaded and take between 3 to 4 seconds. After waiting, the list displays the basic information of the player in the displayed position with his photo, name and position in a badge. If the user wants to see the profile of the player in a certain position, he/she can select a player in the list and it will show the basic profile information of the player.

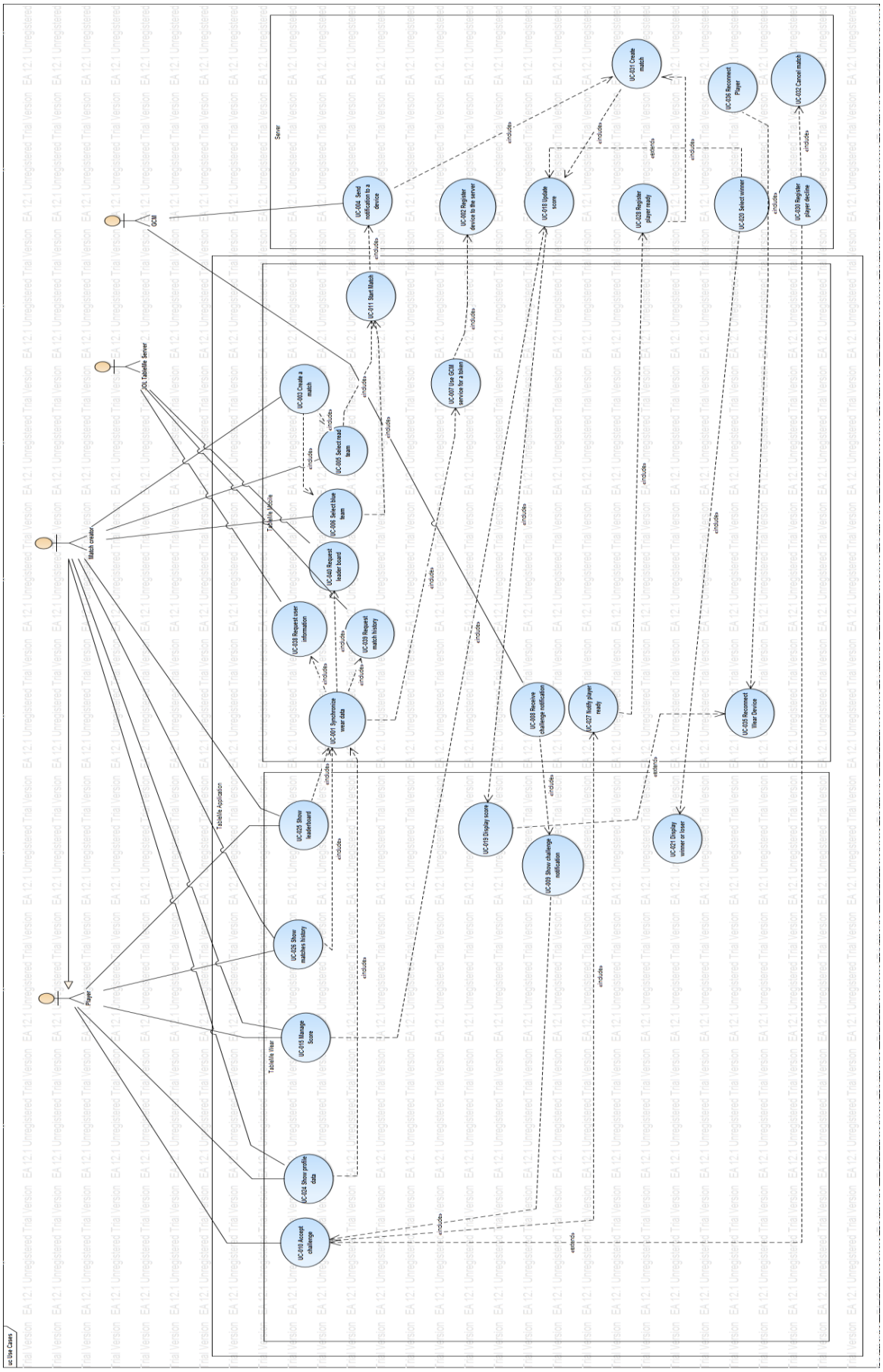
7.7. Relational Matrix of Functional Requirements



7.7. Domain Model



7.8. Use Case Diagram



7.9. Class Diagram Table Me