value through innovation

# SRS

## Contents

# 1. Requirements Specification

## 1.1. Introduction

Gym King is a product designed to offer a unique motivational tool for gym users. Gym King takes the concept of gamification and combines it with working out. Gyms will be able to create a collection of badges that gym users can earn by achieving their workout goals.

As an added motivation for people to push themselves harder in their workouts is the Gym Leader boards. The more badges a user has achieved, the higher ranking they would have on the Gym's leader board.

Using open-source technology, with a dash of Augmented Reality (AR), Gym King plans to revolutionise the Gym Community.

## 1.2. User Characteristics

There will be three types of users – Gym Owner, Gym Employee and Gym Member. Each user will be able to view the leader boards and badges available for the gym, as well as the details of the gym, such as its location on the map, and the facilities it has.

### 1.2.1. Gym Owners

Gym Owners are the owners of the gym. A Gym Owner can manage a single gym or multiple gyms. This user will be adding/updating information about the gym, such as its name, its location, and what facilities the gym has available.  This user will also be creating, editing, and deleting the badges that the gym offers. Gym Owners will also be able to manage the employee profiles for each gym's employees.

### 1.2.2. Gym Employees

Gym Employees will primarily be focusing on the assisting Gym Members workout. Gym Employees will be able to verify that a Gym Member has done enough to achieve a badge that the gym offers – such as running 10 kilometres. Whilst an Artificial Intelligence component will go the majority of the verification for the badge claims from Gym members, the Gym Employees will occasionally need to accept or reject claims that are disputed.

### 1.2.3. Gym Members

Gym Members will be the members and users of the gym. They will mainly be focusing on the actual workouts and earning badges for working out. They will submit evidence that they have workout, such as a photo of treadmill display that says that the user has run 10 kilometres that session and will then wait for their claim to be accepted or rejected. If their claim has been accepted, then they would earn a badge. The more badges a Gym Member has, the higher their ranking will be on the Gym's leader board. Gym Members will also be able to communicate with other gym members and their friends to compete in their workouts.

## 1.3. User Stories

| 1. Creating a GYM |
|---|
| User Story: As an owner of a Gym, I want to be able to create a gym on the application. I want to be able to name my gym, give details about my gym, and mark the location of my gym on the app's map. |
| Acceptance Criteria: Given the information provided, create a gym, saving the name of the gym, details of the gym, and save the location of the Gym on the Application's built-in map feature. |

| 2. Creating a USER /Registering a USER |
|---|
| User Story: As a user, I want to be able to register a new account with the app. |
| Acceptance Criteria: The system should allow all users to register a new account, and update the database in the back-end accordingly. |

| 3. Creating a BADGE |
|---|
| User Story: As a gym owner, I want to be able to create a wide variety of badges that members of my gym can earn. I want to be able to specify the name of the badge, the requirements that need to be completed to earn the badge, etc. |
| Acceptance Criteria: The system should allow a user to input the desired information to create a badge that gym members can earn. |

| 4. Editing/Deleting a BADGE |
|---|
| User Story: As a gym owner, I want to be able to delete or edit a badge that I have already created. Sometimes, I want to change details about the badge, or I no longer want that badge to be available. |
| Acceptance Criteria: The system should allow gym owners to be able to edit and/or delete any existing badges that they have. |

| 5. Uploading a BADGE CLAIM |
|---|
| User Story: As a gym member, I need to be able to create a claim to earn a badge. |
| Acceptance Criteria: The user should be able to input the necessary information, such as the workout activity they completed and proof of the activity (e.g., a picture of a treadmill readout) to confirm that the user has completed the workout and the claim must be submitted. |

| 6. Accepting/Rejecting a BADGE CLAIM |
|---|
| User Story: As a gym employee I want to be able to verify that a gym member actually did the workout activity that they claimed that they did, and accept or reject the claim, however, I don't want to have a lot of additional administrative work added to my workload. |
| Acceptance Criteria: The system should use the AI component to do basic verification of badge claims, and accept or reject the easily identifiable claims, so that only the edge cases of the badge claims need to be manually accepted or rejected by the gym employee. |

| 7. Editing/Deleting a GYM |
|---|
| User Story: As an owner of a gym, I want to be able to edit the details of my gym, for example, if the location of the gym changes, I want to be able to change the address of the gym on the app. I also might want to delete a gym, if the gym closes down. |
| Acceptance Criteria: The system should allow a gym owner to edit the details of the gym that they manage and update this information in the database. If a gym is deleted, the gym should be completely removed from the system and the database. |

| 8. Log In |
|---|
| User Story: As a User of the Gym King application, I want to be able to log in to the application using my credentials. |
| Acceptance Criteria: When the user inputs their log in details, the system should confirm that the details are correct, and then should log the user into their account. |

| 9. Using the MAP |
|---|
| User Story: As a user of the product, I want to be able to use the built-in map function to find gyms near me, and to be able to find specific gyms in an area. |
| Acceptance Criteria: The system should allow all users to use the map function, and the map function should be able to display the locations of the registered gyms, and the user's current location. |

| 10. The LEADER BOARDS |
|---|
| User Story: As a gym member, I want to be able to view the leader boards for my gym, so that I can see what ranking I have, and try to rank higher. |
| Acceptance Criteria: All users should be able to view the leader boards. |

| 11. PROFILE PAGE |
|---|
| User Story: As a user, I want to be able to view my account profile page, and I also want to be able to view the profile pages of other users. |
| Acceptance Criteria: The system should allow users to view their profile pages, as well as the profile pages of other registered users, whilst still maintaining privacy and information security – e.g., not displaying information that the user does not want displayed on the profile page. |

| 12. Viewing the PENDING BADGES |
|---|
| User Story: As a gym employee, I want to be able to see what claims are still outstanding. As a gym member, I want to be able to see the status of the badges that I have claimed for. |
| Acceptance Criteria: The system should allow the user to view the badge claims that are still outstanding. |

| 13. Finding FRIENDS |
|---|
| User Story: As a user, I want to be able to find friends through the app so that we could work out together. |
| Acceptance Criteria: The system should allow users to be able to find other users of the application. |

| 14. Accepting/Rejecting FRIEND REQUESTS |
|---|
| User Story: As a user, I want to be able to accept friend requests, but I also want to be able to reject friend requests from people I don’t know. |
| Acceptance Criteria: Users should be able to both accept and/or reject friend requests from other users, and the system should update according to the accept/reject selection. |

| 15. Viewing the PROFILE of FRIENDS |
|---|
| User Story: I want to be able to see what my friends are doing, so I want to be able to see their profiles. |
| Acceptance Criteria: The system should allow users to view the profiles of other users. |

| 16. FORGOT Password |
|---|
| User Story: As a user, I sometimes might forget my password. I want to be able to access my account if I forget my password. |
| Acceptance Criteria: Using an OTP system, the user should be able to reset their password, so long as they have an existing account with the application and have access to the telephone number and/or email address connected to the account to reset their password. |

| 17. Viewing BADGE in Augmented Reality (AR) |
|---|
| User Story: As a User of the application, I want to be able to view the badges that I can earn in Augmented Reality. |
| Acceptance Criteria: Badges should be able to be viewed in augmented reality, so long as the device the app is located on has Augmented Reality capability. |

## 2. Functional Requirements

- FR1 - Gym Owners must be allowed to create, edit and delete badges.
    - FR1.1 - Gym Owners must be able to create badges
    - FR1.2 - Gym Owners must be able to edit badges.
    - FR1.3 - Gym Owners must be able to delete badges.
- FR2 - The product's built-in map system should allow users to view the location of all registered Gyms, as well as being able to display details about each gym.
    - FR2.1 – The Map should be able to allow users to view the details of each gym located on the map.
    - FR2.2 – The Map should be able to recentre itself to the user's current location.
    - FR2.3 – The Map should be able to display all registered gyms.
    - FR2.4 – The Map should allow the Gym Owner to add a Gym to the Map.
    - FR2.5 – The Map should allow the Gym Owner to edit details about the Gym on the Map.
    - FR2.6 – The Map should allow the Gym Owner to delete a gym.
- FR3 - Gym Owners and Gym Employees must be able to accept/reject the Gym Members' badge claims.
    - FR3.1 – Gym Owners and Gym Employees must be able to reject the Gym Members' badge claims.
    - FR3.2 – Gym Owners and Gym Employees must be able to accept the Gym Members' badge claims.
- FR4 - Gym Members must be able to submit claims for badges.
- FR5 - The product must allow users to view the badges that the Gym has made available.
- FR6 - Users must be able to view the badges earned by other users.
- FR7 - All leader boards must be viewable by users.
    - FR7.1 – Leader boards should be able to display the username of the Gym Member and the total number of badges that they have earned.
- FR8 - The product must allow the users to be ranked on the leader boards according to the number of badges earned
- FR9 - The product must be able to support all augmented reality (AR) components (for example, the AR badge display for Gyms)
- FR10 – The product should allow users to register an account.
    - FR10.1 – The product should allow a gym owner to register an account.
    - FR10.2 – The product should allow a gym employee to register an account.
    - FR10.3 – The product should allow a gym member to register an account.
- FR11 – The product should allow users to log in.
    - FR11.1 – The product should allow a gym owner to log in to their account.
    - FR11.2 – The product should allow a gym employee to log in to their account.
    - FR11.3 – The product should allow a gym member to log in to their account.
- FR12 – The product should allow users to reset their password if the password is forgotten.
    - FR12.1 – The product should allow a gym owner to reset their password if the password is forgotten.
    - FR12.2 – The product should allow a gym employee to reset their password if their password is forgotten.
    - FR12.3 – The product should allow a gym member to reset their password if their password is forgotten.
- FR13 – The Artificial Intelligence (AI) component should be able to verify the gym member's badge claim.
- FR14 – The Map should allow users to search for a gym using the Map's search bar.
- FR15 – The product should allow users to accept and/or reject friend requests.

- FR16 – The product should allow users to create a friend request.
- FR17 – The product should allow users to view the profile of other users.

## 2. 1. Subsystems

### S1 – Map

The Map subsystem allows users of the application to find registered gyms. The Map allows users to scan their area, or specifically search for gyms using the search bar. The map uses the user's current location to centre the map and to render the map around them.

### S2 – Leader boards

The leader boards take the number of badges and points that a user has earned and then ranks them accordingly on the leader boards. The leader boards allow the user to view their ranking within the gym overall and within specific categories, such as strength and cardio. The higher the number of badges and points, the higher the ranking.

### S3 - Gym Management

The Gym management subsystem allows the owner of the gym to manage the gym. This subsystem links to the employee profiles and the ability to add new employees to gyms, as well as creating, editing and deleting badges for a gym, as well as adding and editing gyms for the other.

### S4 – Users

The user subsystem allows users to access their profiles, edit their account details, etc. Some of the parts of the system that the user can access depends on their user role. A gym owner has access to the gym management subsystem. The gym member users can upload claims for badges, etc. The gym employee users can accept or reject badge claims, etc. Each user role has access to the map subsystem.

### S5 – AR (Augmented Reality)

The Augmented Reality subsystem allows the application to display the badges made in the application to be viewed in AR through the user's device's camera. This subsystem is largely dependent on whether or not the device it is being used on has AR capabilities. Whilst, most modern mobile devices have this capability, not all do.

### S6 – AI (Artificial Intelligence)

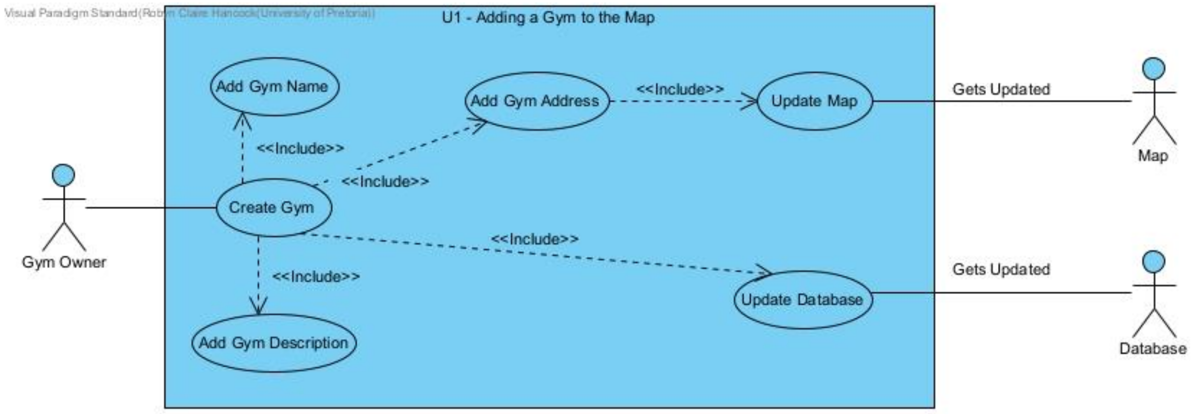
The Artificial Intelligence subsystem reduces the amount of work placed on the gym employees by automating the verification of the badge claims and the subsequent acceptance of or rejection of the badge claim. The gym employees will then only have to handle disputed badge claims – such as when the AI rejects a badge claim but the gym member believes this to be inaccurate, or edge cases that the AI is uncertain of.

### S7 – Friends

The friend subsystem allows users of the application to connect with other users of the application. Friends can be created through friend requests, which can either be accepted or rejected. Friends can workout together and help encourage other users to improve their workouts and earn more badges.

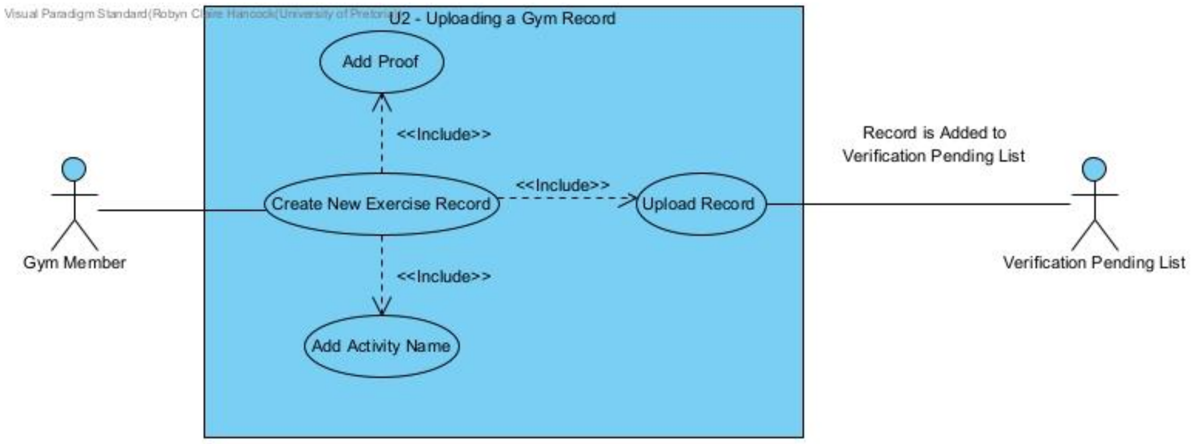
## 2.2. Use Cases

### 2.2.1. U1 - Adding Gym to Map



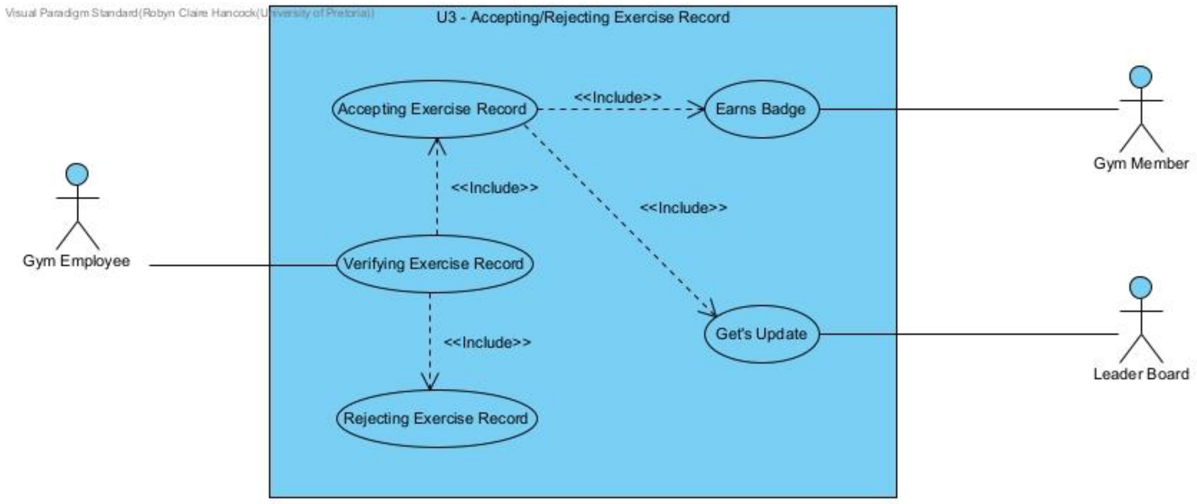
A Gym Owner creates a Gym, adding the Gym Name, a basic description, and the Gym's address. The Address is then added to the Map, so that Gym can be displayed. The Database is also updated with all the information gathered about the Gym.

### 2.2.2. U2 - Uploading Exercise Record


Visual Paradigm Standard(Robyn Claire Hancock(University of Pretoria))
U2 - Uploading a Gym Record
Add Proof
<<Include>>
Record is Added to Verification Pending List
Create New Exercise Record
<<Include>>
Upload Record
Gym Member
Verification Pending List
<<Include>>
Add Activity Name

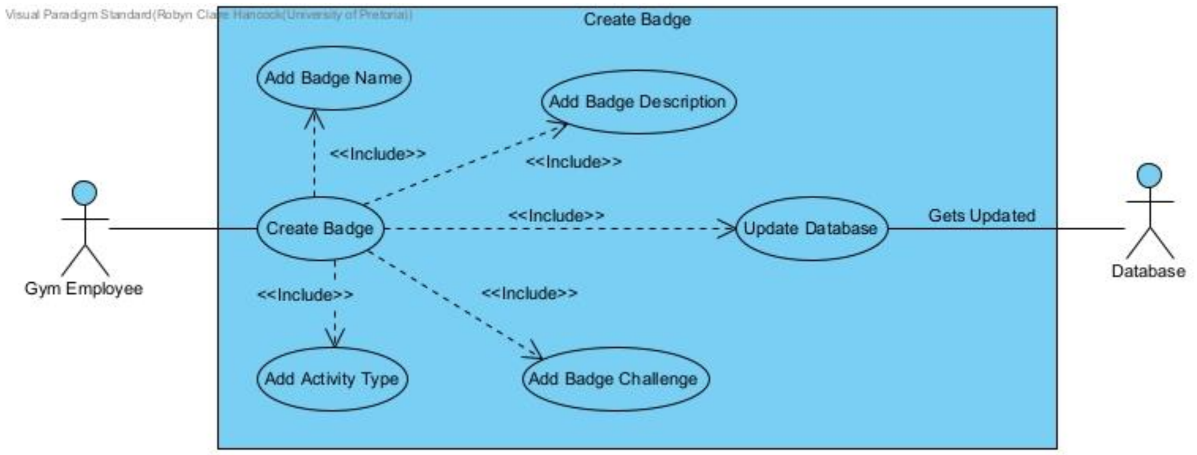
A Gym member creates a new Exercise record, stating the name of the activity completed (e.g. cycling) and uploads visual (photographic) proof of the exercise and presses the upload button. The record is then sent to the Verification Pending list, where it will be until a Gym Employee either accepts the record or rejects it.

### 2.2.3. U3 - Accepting/Rejecting Exercise Record


Visual Paradigm Standard(Robyn Claire Hancock(University of Pretoria))
U3 - Accepting/Rejecting Exercise Record
Accepting Exercise Record
<<Include>>
Earns Badge
<<Include>>
<<Include>>
Verifying Exercise Record
<<Include>>
Get's Update
Rejecting Exercise Record
Gym Employee
Gym Member
Leader Board

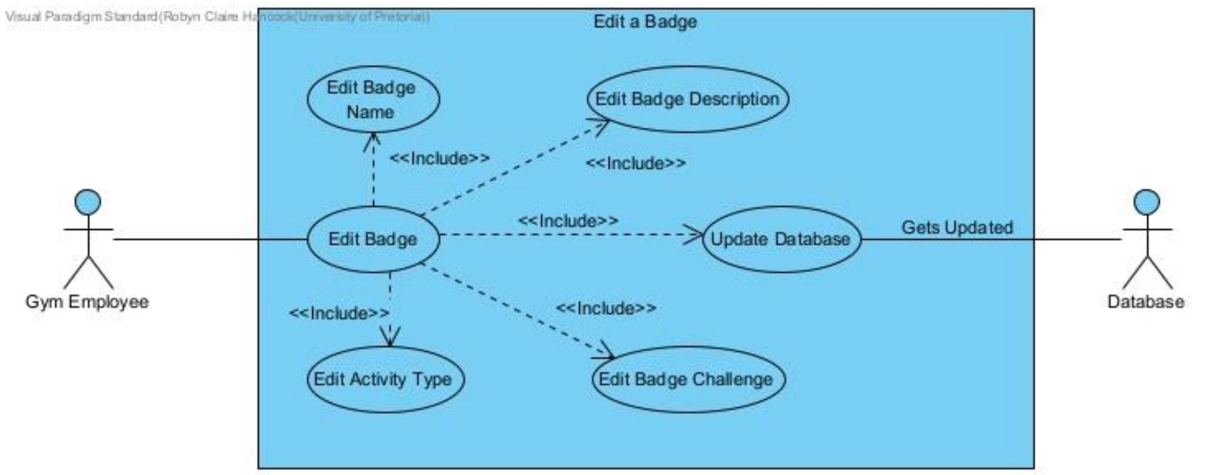
Here, a Gym Employee takes an Exercise record from the list of pending records, mentioned above, and verifies the record by either accepting or rejecting it. If it is rejected, the record is simply discarded, however, if the record is accepted, the Gym Member associated with the record receives the appropriate badge, and the leader board is then updated to count this new badge.

### 2.2.4 U4 – Creating a Badge



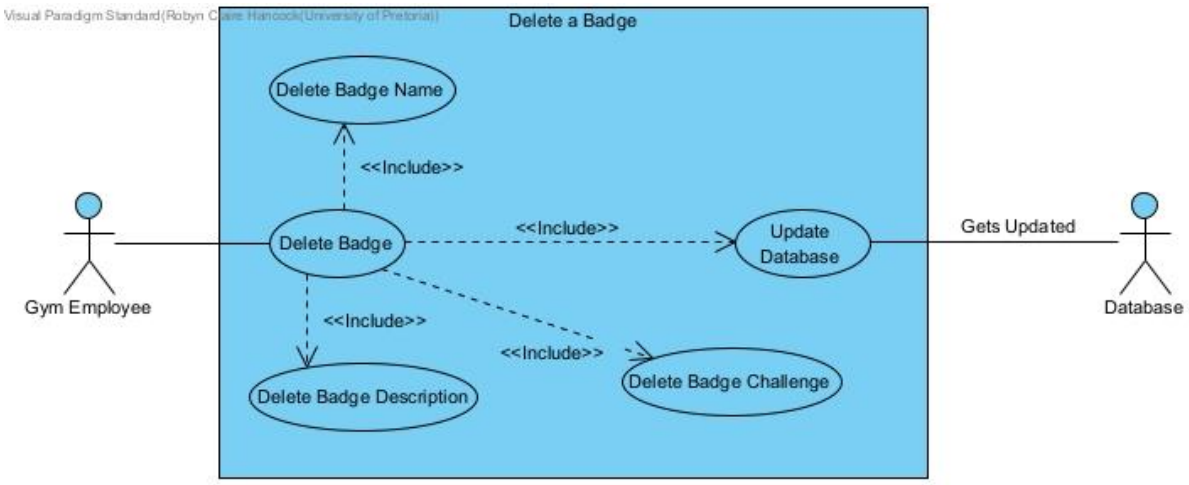
A key feature of the Gym King product is for Gym Employees and Gym Owners to be able to create new badges that their Gym Members can earn. Creating a badge is a simple task. When the Gym Employee decides to create a new badge, they are taken to the Create Badge page. On this page, the Gym Employee will be able to add a name and description of the badge, link the badge to a particular activity type, such as Cardio and add a challenge, such as cycling 5km in a single session, that needs to be completed to earn the badge. Once the Gym Employee has created the badge, the badge is available for Gym Members to earn.

## 2.2.5 U5 – Editing a Badge



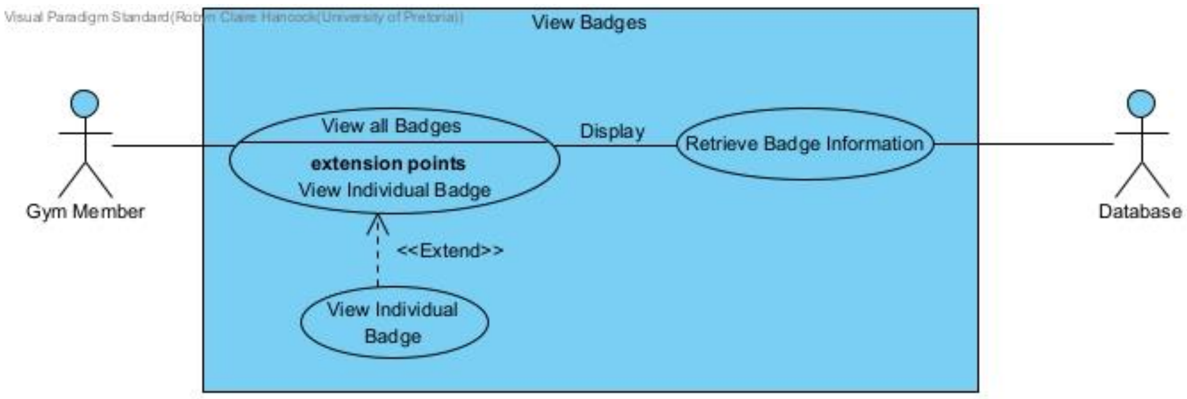
When a Gym Employee wishes to edit a badge, they will select the badge they wish to edit, and then select the edit option. They will then be taken to the Edit Badge page, which is similar to the Create Badge page. Here the Gym Employee will be able to edit any of the fields that make the badge, such as the name, description, the challenge and the activity type associated with the badge. Once they have edited the badge, they will save it, and the changes will take effect. The badge is updated in the database and the changes will be visible when viewing and earning the badge.

## 2.2.6 U6 – Deleting a Badge



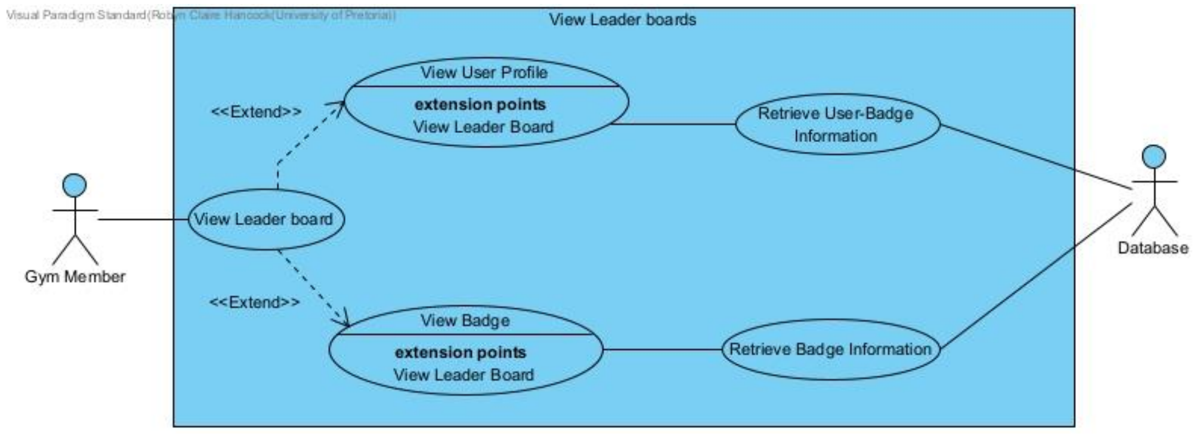
When a Gym Employee wishes to delete a badge they will select the badge and then select the delete badge option. This will delete the badge and badge will no longer be displayed with the other badges on the View Badge page, this also means that no Gym Member can earn this badge.

### 2.2.7 U7 – Viewing Badges



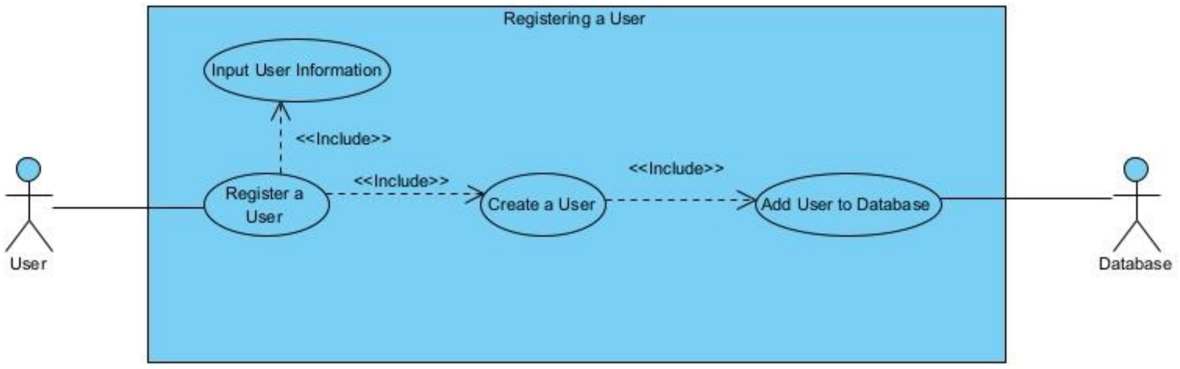
The View Badge page allows the user, whether Gym Employee or Gym Member to view all the badges that a gym has. The page will allow the user to click on a badge, which will then take the user to the Badge page where the user can view the details associated with the selected badge. The View Badge Page can also display badges for different gyms.

### 2.2.8 U8 – Viewing Leader boards



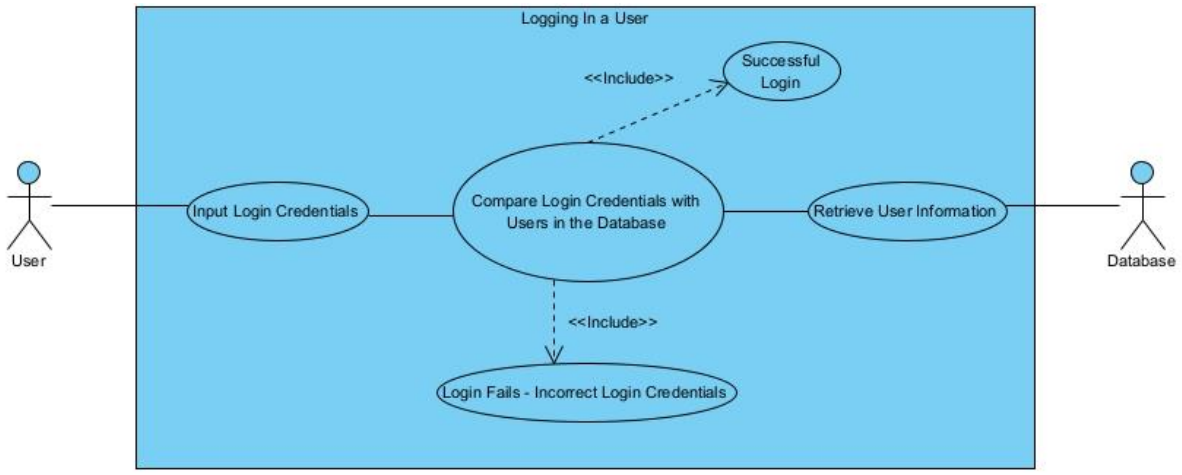
The Leader boards is where the users can view who has the most badges. Each user has the number of badges they have earned tallied up and ranked accordingly. The information is taken from the database and displayed for the user to see.

### 2.2.9 U9 – Registering a User



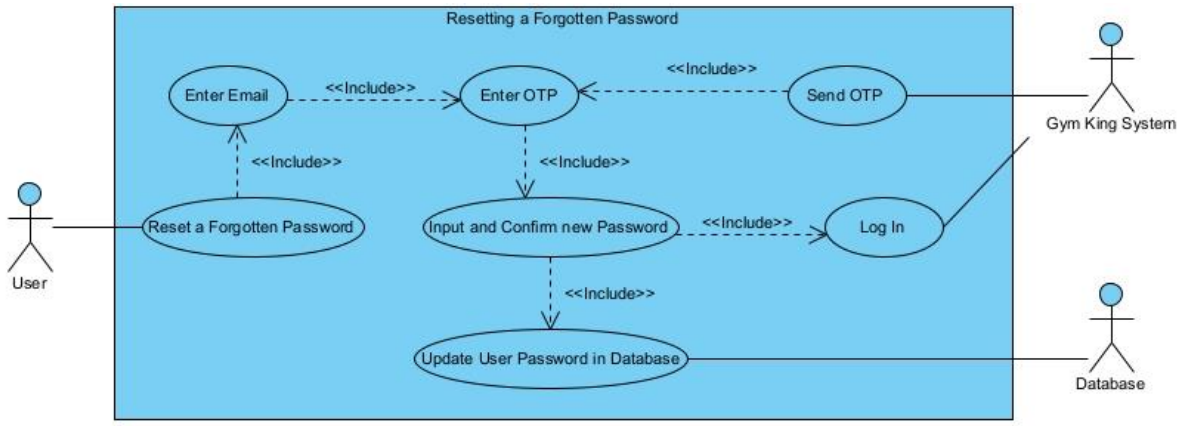
When the application opens, the user has two options, to register as an existing user or register as a new user. When registering as a new user, the user as to input information needed to create their account. Information such as their name, username, password, email address and phone number. This information is then saved in the database and a new user is created.

### 2.2.10 U10 – Logging In


Logging In a User
<<Include>>
Successful Login
User
Input Login Credentials
Compare Login Credentials with Users in the Database
Retrieve User Information
Database
<<Include>>
Login Fails - Incorrect Login Credentials

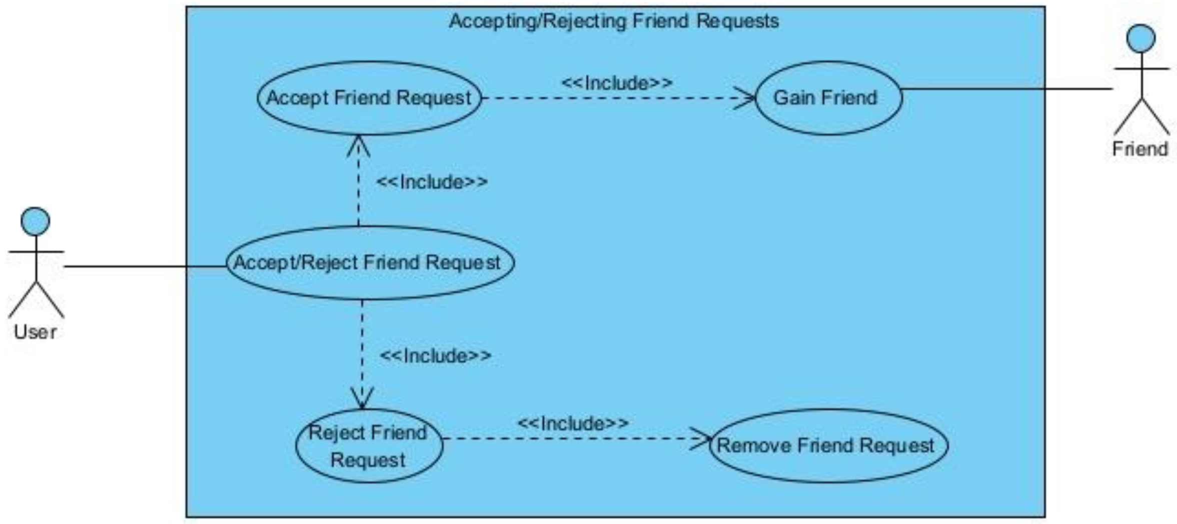
When the application opens, the user arrives on the Login screen. Here the user just has to enter their log in credentials, such as their email address and password. If the Log in is successful – i.e. the user's log in details match that which is stored in the database – the user is logged into their account. If the log in is unsuccessful, the user is notified via a pop-up (toast) notification that their details are incorrect.

### 2.2.11 U11 – Reset a Forgotten Password



In the event that a user forgets their password, they can reset their password using a small link at the bottom of the Login page. The user will then enter the email address linked to their account. The system will then send the user an OTP (One Time Pin) that lasts 5 minutes to their email. When the correct OTP is inputted, the user will then be able to enter a new password, and then enter this new password again to confirm the new password. The user's password is then updated in the database and the user is now logged in to their account with their new password.

### 2.2.12 U12 – Accepting/Rejecting Friend Requests



The application allows user to link with friends/other users that are registered with the application. The user can link with friends using friend requests. When the user receives a friend request, they can either accept or reject the friend request. If the friend request is accepted, the user then gains a friend on the application, else if the application is rejected, then the friend request is removed from the system.

## 3. Service Contracts

### 3.1. Users

#### 3.1.1. GETs

| Endpoints | Parameters | Returns |
|---|---|---|
| /badges/badge/{bid} | URL {string} bid - Give badge ID for specific badge or * for all badges. | The specified badges details or if the input was '*' you will get every single one of the badges' details. |
| /badges/gym/{gid} | URL {string} gid - Input of the gym ID to find all badges that belong to the gym. | A list with information on all badges that belong to the gym. |
| /brands/brand/ | URL no parameters | Returns all gym brands with their information |
| /brands/brand/{brandname} | URL {string} brandname - brand name | Returns the specific brand with its information. |
| /brands/brand/logo/{brandname} | URL {string} brandname - brand name | Returns the logo of the gym brand. |
| /brands/brand/badges/{brandname} | URL {string} brandname - brand name | Returns the badges that belong to brand. |
| /gyms/gym/name/{gymName} | URL {string} gymName - name of the gym. | Returns the specific gym with its information. |
| /badges/gym/{gid} | URL {string} gid - gym's ID | List all badges that belong to gym. |
| /gyms/gym/{gid} | URL {string} gid - Input of the gym ID to find a gym. | Information on the gym found by the gym ID. |
| /gyms/getAllGyms | URL no parameters | Returns all the gyms. |
| /leaderboard/score/{gid} | URL {string} gid - Gym ID. | List of all badges, with usernames and amount of times they earned that badge. |
| /users/user/{username} | URL {string} username - gym user username | Information about the user. |
| /users/owned/{username} | URL {string} username - gym user username | List of badges owned by the user. |
| /Model/iOS ?rank={rank}& emblem={emblem} | URL {string} rank - Rank of badge. g b or s. URL {string} emblem - | {USDZ file} Response is a download to get a USDZ file to then be used for |

| | | |
|---|---|---|
| | Emblem name of badge. bicep,cycle etc. | the AR component of the app. iOS only |
| /Model/Android ?rank={rank}& emblem={emblem} | URL {string} rank - Rank of badge. g b or s.<br>URL {string} emblem - Emblem name of badge. bicep,cycle etc. | {GLB file}<br>Response is a download to get a GLB file to then be used for the AR component of the app. Android only. |
| /users/user/picture/{username} | URL {string} username - Gym user username | {image URL}<br>Response is a public URL used to view the gym user's profile picture. |
| /leaderboard/score/{gid} | URL {string} gid - Gym ID. | List of all badges, with username, number of times they completed the badge. |
| /users/user/requests/getAllRequests | URL no params | List of all requests. to and from. |

### 3.1.2. POSTs

| **Endpoints** | **Parameters** | **Returns** |
|---|---|---|
| /user/badges | BODY {string} email - Input of user email<br><br>BODY {string} apikey - Input of user apikey | A list of badges the user has earned with the badges information.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |
| /users/claims/ | BODY {string} email - Input of user email<br><br>BODY {string} apikey - Input of user apikey | list of all claims, with the claim info, that the user has made.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |
| /claims/claim | BODY {string} bid - The badge ID of the badge.<br><br>BODY {string} email - The email of the user who claims they completed it. | Returns a json object saying {'success' : true}.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |

| | | |
|---|---|---|
| | BODY {string} apikey - The apikey of the user.<br><br>BODY {string} input1 - The first input<br><br>BODY {string} input2 - The second input<br><br>BODY {string} input3 - The third input<br><br>BODY {string} proof - image of proof. | |
| /users/login | BODY {string} email - Email of the user.<br><br>BODY {string} password - Password of the user.<br><br>BODY {string} usertype - Type of user.<br>gym_user,<br>gym_employee,<br>gym_owner | Returns a json object saying<br>{ 'success': true,'profile_picture': profile_picture,<br>'username':username,'apikey':apikey }<br><br>If authorisation of the user failed because of a wrong email or password. Returns a json object saying<br>{ 'success': false, 'results':'invalid email or password'} |
| /users/user/checkIfFriends | BODY {string} user1email - Email of the user.<br><br>BODY {string} apikey - apikey of the user.<br><br>BODY {string} user2email - Email of the user. | Boolean value showing if the users are friends or not. |
| /users/user/<br>checkIfPendingFriends | BODY {string} user1email - Email of the user. | Boolean value showing if the users are pending friends or not. |

| | | |
|---|---|---|
| | BODY {string} apikey - apikey of the user.<br><br>BODY {string} user2email - Email of the user. | |
| /users/user/suggestion | BODY {string} email - Email of the user.<br><br>BODY {string} apikey - apikey of the user. | Badges that belong to the user's gym membership gyms that are similar to the one the user takes part in. |
| /users/user | BODY {string} email - The email of the user.<br><br>BODY {string} name - The name of the user.<br><br>BODY {string} surname - The surname of the user.<br><br>BODY {string} number - The phone number of the user.<br>BODY {string} username - The username the user created.<br><br>BODY {string} password - The password the user created (NOT encrypted).<br><br>BODY {string} membership - The gym brand the user has a membership with. | Returns params of completed insertion.<br><br>{ 'success': true, 'results':{PARAMS}} |

| | | |
|---|---|---|
| /gyms/aroundme | BODY {string} latCoord - latitude of user<br><br>BODY {string} longCoord - longitude of user<br><br>BODY {string} radius - circle radius in KM to check for gyms | A list of gyms and their locations |
| /users/user/OTP | BODY {string} email - Email of user. | message indicating creation.<br>{ 'success': true } |
| /users/user/info | BODY {string} email - User's email.<br><br>BODY {string} apikey - User's apikey. | User’s information. |
| /users/user/getUser | BODY {string} email - User's email.<br><br>BODY {string} apikey - User's apikey.<br><br>BODY {string} username- User's username. | Get user’s more personal information. |
| /users/user/ checkIfSubscribed | BODY {string} email - User's email.<br><br>BODY {string} apikey - User's apikey.<br><br>BODY {string} gid - User's gym ID. | boolean value returning true or false. |
| /users/user/ CreateRequest | BODY {string} fromEmail - from user<br><br>BODY {string} toEmail - to user | Message confirming creation or not. |

| | | |
|---|---|---|
| /users/user/getFriends | BODY {string} userEmail - user email | Returns a list of friends the user has |
| /users/user/ getReceivedRequests | BODY {string} userEmail - user email | returns list of requests from friends. |
| /users/user/ getSentRequests | BODY {string} userEmail - user email | returns list of requests to friends. |
| /users/user/ createSubscription | BODY {string} fromEmail - user email<br><br>BODY {string} gid - gym ID | message confirming creation of subscription. |
| /users/user/ getGymSubscriptions | BODY {string} fromEmail - user email | List of gyms the user is subscribed to. |
| /users/user/ getSubscribedUsers | BODY {string} gid - gym ID | List of all users that subscribe to the gym ID. |
| /users/user/ SendSubscriberNotification | BODY {string} g_id - gym ID<br><br>BODY {string} pushMessage - Message<br><br>BODY {string} pushTitle - title of notification<br><br>BODY {boolean} isSilent- Is the notification silent or not | message confirming notification and the emails that the notification is using to send to users. |
| /users/user/ SendFriendsNotification | BODY {string} userEmail - user email<br><br>BODY {string} pushMessage - Message<br><br>BODY {string} pushTitle - title of | message confirming notification and the emails that the notification is using to send to friends. |

| | | |
|---|---|---|
| | notification<br><br>BODY {boolean} isSilent- Is the notification silent or not | |
| /users/user/ SendFriendsNotification | BODY {string} pushTarget - target<br><br>BODY {string} pushMessage - Message<br><br>BODY {string} pushTitle - title of notification<br><br>BODY {boolean} isSilent- Is the notification silent or not | message confirming notification and the emails that the notification is using to send to users. |

### 3.1.3. PUTs

| Endpoints | Parameters | Returns |
|---|---|---|
| /users/user/info | BODY {string} email - The email of the user.<br><br>BODY {string} name - The name of the user.<br><br>BODY {string} surname - The surname of the user.<br><br>BODY {string} number - The phone number of the user.<br><br>BODY {string} username - The username the user. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

| | | |
|---|---|---|
| | BODY {string} apikey- The apikey of the user.<br><br>BODY {string} membership - The membership of the user. | |
| /users/user/picture | BODY {string} email - User's email.<br><br>BODY {string} apikey - User's apikey.<br><br>BODY {string} profilepicture - User's profilepicture. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /users/user/password | BODY {string} email - The email of the user.<br><br>BODY {string} otp - OTP given by user.<br><br>BODY {string} newpassword - New password. | Returns a json object saying { 'success': true }<br><br>If invalid email is given or the OTP is incorrect. Returns a json object saying {'message':'Invalid email or OTP!'} |
| /users/user/ notificationToggle | BODY {string} email - User's email.<br><br>BODY {string} apikey - User's apikey. | returns message confirming toggle.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /users/user/pushToken | BODY {string} email - User's email. | message confirming push token modification |

| | | |
|---|---|---|
| | BODY {string} token- User's new token | |

### 3.1.4. DELETEs

| **Endpoints** | **Parameters** | **Returns** |
|---|---|---|
| /users/delete | BODY {string} email - unique email used to delete the user.<br><br>BODY {string} apikey - User's apikey. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /users/user/ deleteRequest | BODY {string} fromEmail - from user<br><br>BODY {string} toEmail - to user | Returns message confirming deletion of request. |
| /users/user/ deleteSubscription | BODY {string} fromEmail - from user<br><br>BODY {string} toGym - to gym ID | Returns message confirming deletion of subscription. |

## 3. 2. Employees

### 3.2.1. GETs

| Endpoints | Parameters | Returns |
|---|---|---|
| /claims/gym/{gid} | URL {string} gid - gym iD | Returns list of all claims that belong to a gym. |
| /employees/employee/picture/{username} | BODY {string} username - employee username. | an image that corresponds to the given username |

### 3.2.2. POSTs

| Endpoints | Parameters | Returns |
|---|---|---|
| /employees/employee/info | BODY {string} email - employee email<br><br>BODY {string} apikey - employee api key | Returns the employee's information.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |
| /claims/claim/getClaim | BODY {string} empEmail - employee email<br><br>BODY {string} email - user email<br><br>BODY {string} bid - badge ID used to match a claim.<br><br>BODY {string} apikey - employee api key | all the details of a specified claim from the claims table<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |
| /employees/employee | BODY {string} ownerEmail - the owner's email address.<br><br>BODY {string} apikey - the apikey of the owner.<br><br>BODY {string} email - the employee's email address. | -inserts an employee into the employee table<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {'message':'Invalid email or apikey!'} |

| | | |
|---|---|---|
| | BODY {string} name - the employee's name.<br><br>BODY {string} surname - the employee's surname.<br><br>BODY {string} number - the employee's phone number.<br><br>BODY {string} username - the employee's username.<br><br>BODY {string} password - the employee's password.<br><br>BODY {string} gid - the employee's gym ID. | |
| /badges/badge | BODY {string} apikey - the apikey of the owner or employee.<br><br>BODY {string} email - the employee or owner email address.<br><br>BODY {string} gid - the id of the gym the badge is being added to.<br><br>BODY {string} badgename - the name of the badge.<br><br>BODY {string} badgedescription - the description of the badge.<br><br>BODY {string} badgechallenge - the challenge required to achieve the badge. | inserts a new badge into the badge table<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

| | | |
|---|---|---|
| | BODY {string} badgeicon - ID of the badge image/ARmodel<br><br>BODY {string} requirement1- requirement of badge<br><br>BODY {string} requirement2- requirement of badge<br><br>BODY {string} requirement3- requirement of badge<br><br>BODY {string} activitytype - the type of activity (cardio/strength)<br><br>BODY {string} tags- tags of the badge ("tag1,tag2,tag3") | |
| /employees/employee/OTP | BODY {string} email - email of employee | returns message |

3.2.3. PUTs

| Endpoints | Parameters | Returns |
|---|---|---|
| /employees/employee/password | BODY {string} email The email of the employee.<br><br>BODY {string} otp - OTP given by employee.<br><br>BODY {string} newpassword - The new password. | changes the specified employee's password |
| /employees/employee/info | BODY {string} email - The email of the employee.<br><br>BODY {string} apikey - | changes the specified employee's information<br><br>if authorisation of the user failed |

| | | |
|---|---|---|
| | the apikey of the employee.<br>BODY {string} name - The name of the employee.<br><br>BODY {string} surname - The surname of the employee.<br><br>BODY {string} number - The phone number of the employee.<br><br>BODY {string} username - The username of the employee. | because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /employees/employee/picture | BODY {string} email - The email of the employee.<br><br>BODY {string} apikey - the apikey of the employee.<br><br>BODY {file} profilepicture - the picture. | changes the specified employee’s profile picture<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /claims/claim | BODY {string} bid - badge ID used to find badge.<br>BODY {string} email - email used to find the user.<br><br>BODY {string} empEmail - The email of the employee.<br>BODY {string} apikey - the apikey of the employee. | Update accepted badge_claim to badge_owned.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /badges/badge | -BODY {string} email - The email of the owner or employee.<br><br>BODY {string} apikey - the api key of the owner or employee. | Returns message confirming update.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

| | | |
|---|---|---|
| | BODY {string} bid badge ID used to find badge.<br><br>BODY {string} gid - gym ID of the badge.<br><br>BODY {string} badgename - edited badgename.<br><br>BODY {string} badgedescription - edited badgedescription.<br><br>BODY {string} badgechallenge - edited badgechallenge.<br><br>BODY {string} requirement1- requirement of badge<br><br>BODY {string} requirement2- requirement of badge<br><br>BODY {string} requirement3- requirement of badge<br><br>BODY {string} activitytype - edited activitytype.<br><br>BODY {string} badgeicon - edited badgeicon.<br><br>BODY {string} tags- edited tags | |

### 3.2.4. DELETEs

| Endpoints | Parameters | Returns |
|---|---|---|
| /badges/badge | BODY {string} apikey - the apikey of the owner or employee.<br><br>BODY {string} email - the employee or owner email address.<br><br>BODY {string} apikey - the apikey of the employee. | removes the specified badge<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /claims/claim | BODY {string} bid - unique bid used to delete the claim.<br><br>BODY {string} email - unique email used to delete the claim.<br><br>BODY {string} apikey - the apikey of the employee.<br><br>BODY {string} email - the employee email address. | removes the specified claim<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /employees/employee | BODY {string} owneremail - owner email.<br><br>BODY {string} apikey - the apikey of the owner.<br><br>BODY {string} employeeemail - the employees email. | removes the specified employee. requires owners permission and password<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

## 3.3. Owners

### 3.3.1. GETs

| Endpoints | Parameters | Returns |
|---|---|---|
| /owners/owner/picture/{:username} | URL {string} username- employee username. | {image URL}<br>Returns the public url for the owner's profile picture. |

### 3.3.2. POSTs

| Endpoints | Parameters | Returns |
|---|---|---|
| /owners/employees | BODY{string} email - email of the owner.<br><br>BODY{string} apikey- apikey of the owner. | List of all employees who work for gyms owned by an owner.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /gyms/owned/getGyms | BODY{string} email - email of the owner.<br><br>BODY{string} apikey- apikey of the owner. | List of all gyms that the owner owns.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /gyms/owned | BODY {string} gid - gym ID of the gym.<br><br>BODY {string} email - email of the owner.<br><br>BODY{string} apikey- apikey of the owner. | json object showing the parameters used to insert the gym owned.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /gyms/gym | BODY {string} email - email of the owner.<br><br>BODY{string} apikey- apikey of the owner.<br><br>BODY {string} gymName - gym name. | json object showing the parameters used to insert the gym.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

| | | |
|---|---|---|
| | BODY {string} gymBrandName - gym brand name.<br><br>BODY {string} gymAddress - gym address.<br><br>BODY {number} gymCoordLong - Longitude coord of gym.<br><br>BODY {number} gymCoordLat - Latitude coord of gym. | |
| /brands/brand | BODY {string} brandname - gym brand name. | message confirming insertion.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /owners/owner | BODY {string} email - owner email.<br><br>BODY {string}} fullname - owner full name.<br><br>BODY {string} number - owner number.<br><br>BODY {string} username - owner username.<br><br>BODY {string} password - owner password. | Returns a json object saying { 'success': true } |
| /owners/owner/OTP | BODY {string} email - Email of owner. | Returns a json object saying { 'success': true } |
| /owners/owner/info | BODY {string} email - owner's email.<br><br>BODY{string} apikey- apikey of the owner. | Returns the owner's information.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

### 3.3.3. PUTs

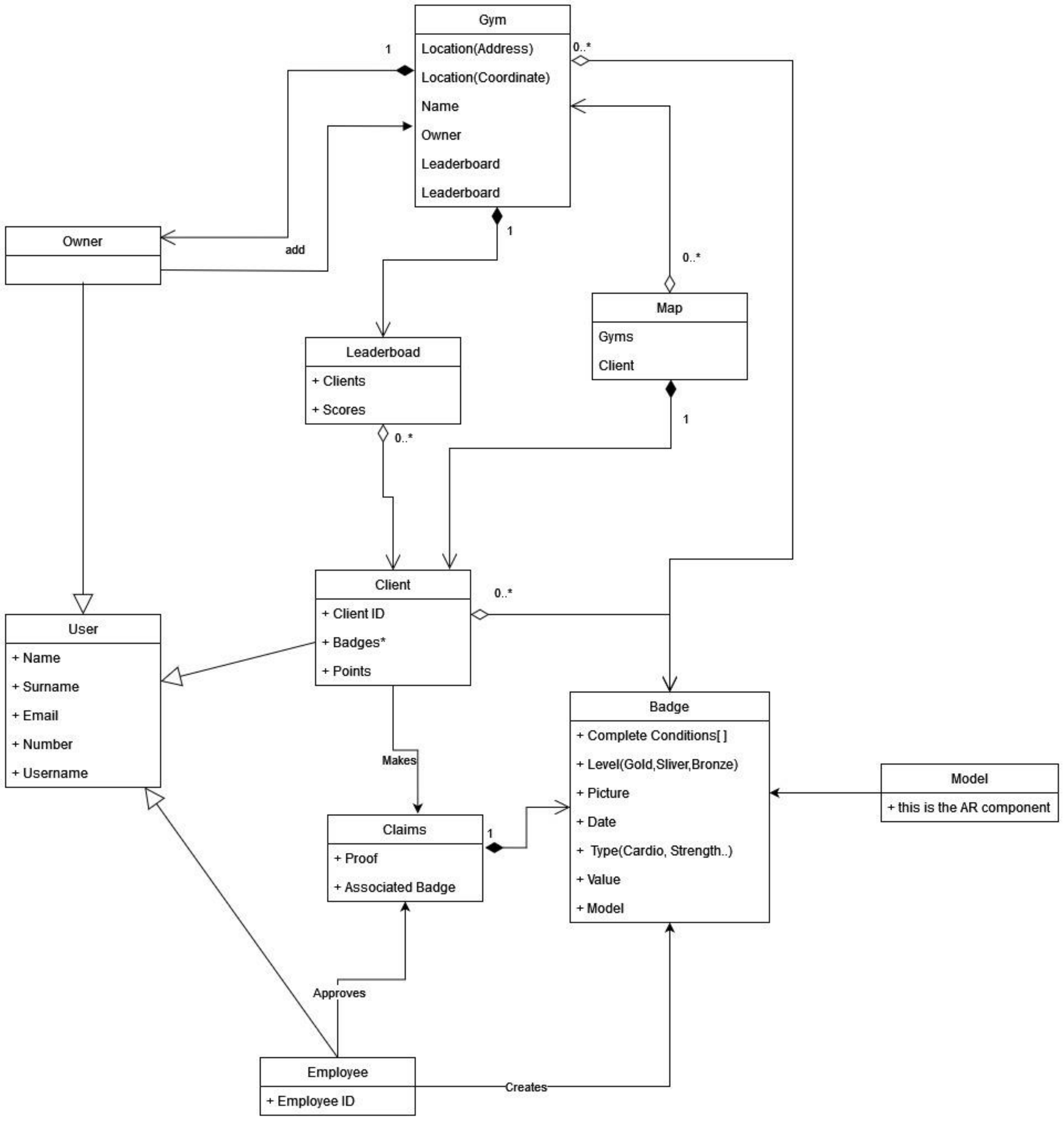
| Endpoints | Parameters | Returns |
|---|---|---|
| /owners/owner/picture | BODY {string} email - The email of the owner.<br><br>BODY{string} apikey- apikey of the owner.<br><br>BODY {file} profilepicture - The pi~~e~~cture. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /brands/brand/logo | BODY {string} brandname - gym brand name.<br><br>BODY {file} logo- gym brand logo.<br><br>BODY {string} email- owner email.<br><br>BODY {string} apikey - owner api key. | message confirming insertion.<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /owners/owner/info | BODY {string} email - owner email.<br><br>BODY {string} fullname - Edited owner full name.<br><br>BODY {string} number - Edited owner number.<br><br>BODY {string} username - Edited owner username.<br><br>BODY {string} apikey - owner api key. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

| | | |
|---|---|---|
| /gyms/gym/info | BODY {string} gid - gym id<br><br>BODY {string} email - owner email.<br><br>BODY {string} apikey - owner api key.<br><br>BODY {string} gymName- gym name<br><br>BODY {string} gym~~b~~Brand~~n~~Name - gym brand name<br><br>BODY {string} gym~~a~~Address - gym address<br><br>BODY {string} gymCoordLat- gym lat coordinates<br><br>BODY {string} gymCoordLong- gym long coordinates | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /owners/owner/password | BODY {string} email - The email of the owner.<br><br>BODY {string} otp - OTP given by owner.<br><br>BODY {string} newpassword - New password. | Returns the owners information.<br><br>If authentication failed because of email or otp. Returns {'message':'Invalid email or OTP!'} |

3.3.4. DELETEs

| Endpoints | Parameters | Returns |
|---|---|---|
| /owner/delete/gym | BODY {string} email - unique owner email used to delete the gym.<br><br>BODY {string} apikey - owner api key.<br><br>BODY {string} gid - gym id | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |
| /owners/delete | BODY {string} email - unique email used to delete the owner.<br><br>BODY {string} apikey - owner api key. | Returns a json object saying { 'success': true }<br><br>If authorisation of the user failed because of a wrong email or apikey. Returns a json object saying {‘message’:’Invalid email or apikey!’} |

## 4. Class Diagram



## 5. Quality Requirements

- QR1 - Privacy
    - The product must comply with the South African POPI Act.
- QR2 - Documentation
    - All Documentation should be always up to date.
- QR3 - Scalability
    - The product must be designed in a manner such that the system works as efficiently as possible regardless of the number of users and gyms registered.
    - The product must be able to always display multiple gyms on the map subsystem.
- QR4 - Correctness
    - All code produced must conform to industry standards.
    - All code must be tested to ensure correct operation of the product.
- QR5 - Performance
    - The system should be able to render all Augmented Reality (AR) components within 30 seconds.
    - The system should be able to display all information within 30 seconds.
- QR6 - Open Source
    - All libraries and technologies used must be open source.
- QR7 - Security
    - The product must be able to protect the system from external and internal threats and ensure that no user information is lost/stolen.
- QR8 - Usability
    - The product should be simple and easy to use regardless of environmental conditions and user capabilities.

## 6. Trace-ability Matrix

| | | USE CASES | | | | | |
|---|---|---|---|---|---|---|---|
| | | U1 | U2 | U3 | U4 | U5 | U6 |
| FUNCTIONAL REQUIREMENTS | FR1 | | | | X | X | X |
| | FR2 | X | | | | | |
| | FR3 | | | X | | | |
| | FR4 | | X | | | | |
| | FR5 | | | | | | |
| | FR6 | | | | | | |
| | FR7 | | | | | | |
| | FR8 | | | | | | |
| | FR9 | | | | | | |
| | FR10 | | | | | | |
| | FR11 | | | | | | |
| | FR12 | | | | | | |
| | FR13 | | X | | | | |
| | FR14 | X | | | | | |
| | FR15 | | | | | | |
| | FR16 | | | | | | |
| | FR17 | | | | | | |

| | | **USE CASES** | | | | | |
|---|---|---|---|---|---|---|---|
| | | U7 | U8 | U9 | U10 | U11 | U12 |
| **FUNCTIONAL REQUIREMENTS** | FR1 | | | | | | |
| | FR2 | | | | | | |
| | FR3 | | | | | | |
| | FR4 | | | | | | |
| | FR5 | X | | | | | |
| | FR6 | X | | | | | |
| | FR7 | | X | | | | |
| | FR8 | | X | | | | |
| | FR9 | X | | | | | |
| | FR10 | | | X | | | |
| | FR11 | | | | X | | |
| | FR12 | | | | | X | |
| | FR13 | | | | | | |
| | FR14 | | | | | | |
| | FR15 | | | | | | X |
| | FR16 | | | | | | X |
| | FR17 | | | | | | X |

| QUALITY REQUIREMENTS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **USE CASES** | | QR1 | QR2 | QR3 | QR4 | QR5 | QR6 | QR7 | QR8 |
| | U1 | | X | X | X | X | X | X | X |
| | U2 | | X | X | X | X | X | | X |
| | U3 | | X | X | X | X | X | | X |
| | U4 | | X | X | X | X | X | | X |
| | U5 | | X | X | X | X | X | | X |
| | U6 | | X | X | X | X | X | | X |
| | U7 | | X | X | X | X | X | | X |
| | U8 | X | X | X | X | X | X | X | X |
| | U9 | X | X | X | X | X | X | X | X |
| | U10 | X | X | X | X | X | X | X | X |
| | U11 | X | X | X | X | X | X | X | X |
| | U12 | X | X | X | X | X | X | X | X |

# 7. Architecture Design

## 7.1. Architectural Design Strategy

It was decided that a Decomposition design strategy would be the best for this project. The Decomposition design strategy integrates seamlessly with the technology used. The Front-end of the product uses Ionic React, which prefers the pure component implementation approach. The database and Server (Heroku) uses a ORM (Object-Relational Mapping) design, which treats each table in the database as a unique entity, and ARCore (Android) and ARKit (iOS), which are separate tools and need individual work. The Decomposition Design Strategy is efficient, ensures that components can be reused and allows for improved error-management.

## 7.2. Architectural Styles

### 7.2.1. Client-Server Architecture (multi-tier)

The Gym King server will be hosted on the cloud service Heroku to provide a service to many different users. The Client-Server architecture is a style that consists of one server that provides for a number of users which is what is required for the Gym King server. The server does not need to know each of the clients and the clients do not need to know each other. The clients are completely dependent on the server. The client-server architecture was chosen to reduce the storage space and computational power needed on the client device. This will save the device's power and will allow lower end devices to be supported as they do not need to process information themselves. This is as long as the lower end devices understand what is being sent to them from the server.

### 7.2.2. Component-based Architecture

The Gym King application is programmed using Ionic React. The application will be broken down into components that will then be used together to provide the required functionality. This follows the Component-based architecture style as the different components accomplish different tasks of the Gym King application.

### 7.2.3. Cloud Computing Architecture

As said above with the client-server architecture, we have to use a cloud-based platform to allow the server to always be available and online. Inside the cloud platform, we run our client-server architecture. Our server now acts as a cloud service which the Gym King application will use on behalf of the users to minimise use of the devices' resources. The application can make calls to the persistent server to ask for resources that it needs to serve the user. An example of this would be the Augmented Reality models. The Gym King application does not have these models saved locally to save space so it would rather ask the server for the models that are needed. The server is persistent on the internet and access to it is gained from the Heroku cloud platform. This means Gym King applications should be able to reliably receive resources from it.

## 7.3. Architectural Quality Requirements

### 7.3.1. Availability

The system is expected to have at least a 95% uptime. Deployed with no single point of failure.

### 7.3.2. Deploy-ability

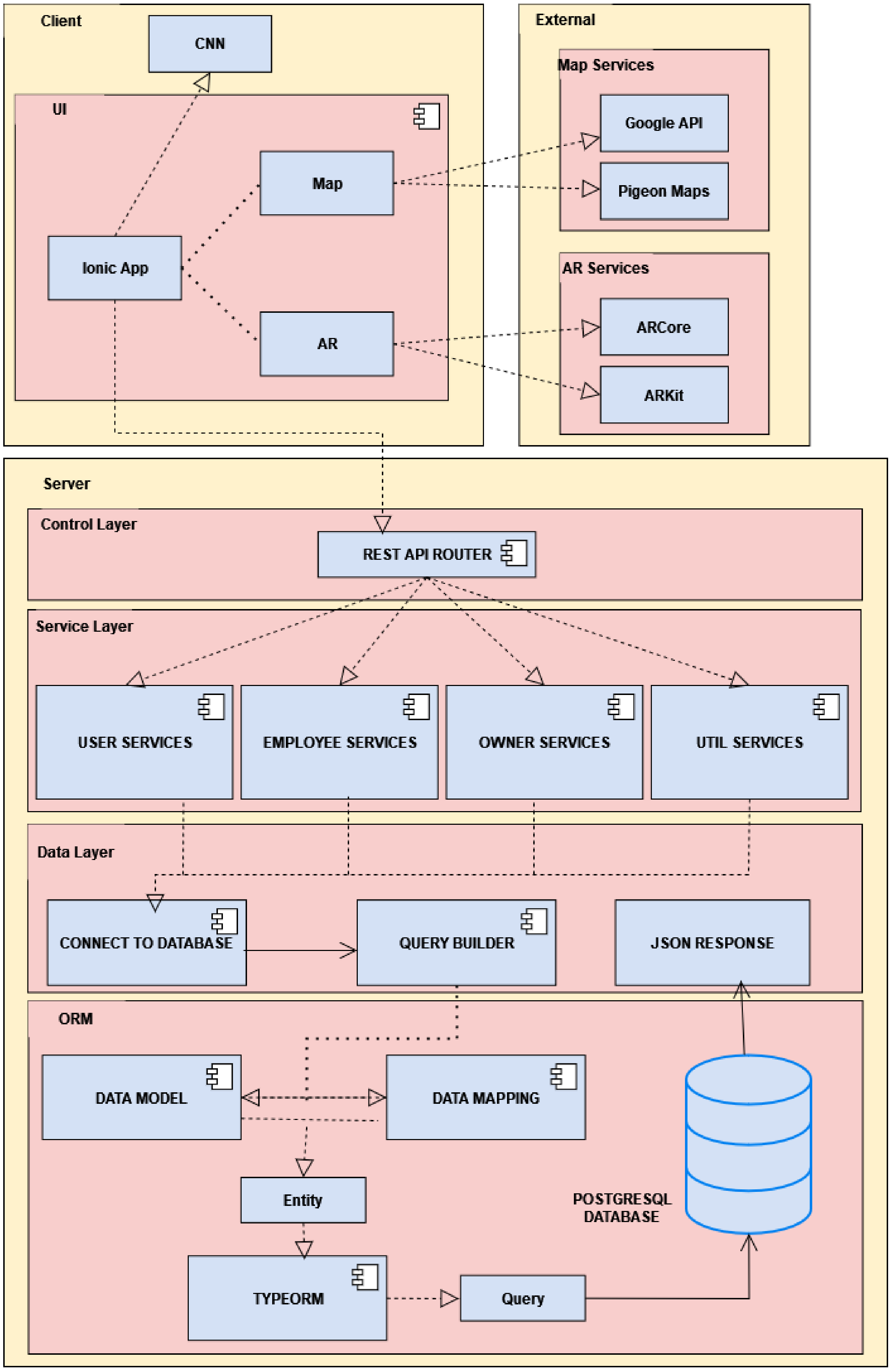
The Gym King application must be deployed, working, and supported on both Android and IOS devices.

#### 7.3.3. Maintainability
The system should be easily understood by future developers. The system should use pure components, such that an individual component can be improved upon without affecting the system. The system should last for several years after deployment, meaning that it should not rely on any packages or services that will become deprecated in the near future. The system should be fully documented, and the at least 50% of the project must have comments detailing their functions.

#### 7.3.4. Security
The system will use role-based access control. Unregistered users should not be able to make use of services offered by the Gym King application. Sensitive user information should be encrypted when stored and should not be accessible by unauthorized parties. The system should not be liable to SQLi attacks, and the components used should not be single-points of failure.

#### 7.3.5. Scalability
The system is designed to handle 50 user requests per second. The system should allow for the addition of extra functionality and/or components. The Gym King server can be scaled up by purchasing more dyno hours, higher tier Heroku server for more connections and better PostgreSQL tier for more connections and storage size.

### 7.4. Architectural Design and Pattern
As explained above, Gym King uses the Client-server pattern, the Cloud-computing pattern and the component-based pattern. The application also uses the ORM (Object-Relational Mapping) pattern. The Architecture is based on the Decomposition design strategy, where all the various components and systems with the application are broken down into smaller and more manageable parts. The architectural patterns were chosen to reflect this. The component-based pattern and the ORM pattern both break down the various components and tables and makes them separate entities. These separate entities are mostly independent of others and can be reused in other parts of the application. The Client-server pattern and the Cloud-computing pattern were chosen to reduce the strain that might be placed on the client device and will allow for improved scalability and manageability as there is a central control of the application.


Client
CNN
UI
Map
Ionic App
AR
External
Map Services
Google API
Pigeon Maps
AR Services
ARCore
ARKit
Server
Control Layer
REST API ROUTER
Service Layer
USER SERVICES
EMPLOYEE SERVICES
OWNER SERVICES
UTIL SERVICES
Data Layer
CONNECT TO DATABASE
QUERY BUILDER
JSON RESPONSE
ORM
DATA MODEL
DATA MAPPING
Entity
TYPEORM
Query
POSTGRESQL DATABASE

## 7.5. Architectural Constraints

1. Limited number of Geo calls per month for the Map component.
2. PostgreSQL's database is situated on a cloud platform.
3. NodeJS server works in TypeScript and is situated on a cloud platform.
4. The Heroku Server does not have any South African servers.
5. The Heroku PostgreSQL database has limits on simultaneous connections, row count and total storage size.
6. The Heroku Server has a dyno hour (uptime) limit of 550 hours per month.
7. The system is limited to open-source technology.
8. The system must run smoothly on a mid-range smartphone/smart device.
9. The system should ensure that battery drainage is minimised as far as possible.
10. AR capabilities are limited to devices that support ARCore (Android) and ARkit (iOS).

## 7.6. Technology Choices

### 7.6.1. Heroku

#### *7.6.1.1. Overview*

Heroku is a cloud platform that allows us to run a server on the web in its environment. We add our server code with a profile to tell Heroku how to start our server.

#### *7.6.1.2. Pros*

- Free to host a server
- Free PostgreSQL database
- Easy to use
- Lots of online help and documentation

#### *7.6.1.3. Cons*

- Limited dyno hours (uptime) per month.
- Does not have South African based servers
- Limits on simultaneous connections, row count and total storage size.
- Only one server running

#### *7.6.1.4. Reasoning*

Heroku allows NodeJS and its various libraries to operate on it. This allows it to work seamlessly within the client-server architecture. It also incorporates libraries to assist with the architecture, such as the express library to make a REST API and Type ORM which will allow the product to make entities of tables for better use of the PostgreSQL database. Heroku is an open source technology that does not have the same constraints that many other cloud service providers offer. Heroku was chosen since it had a wide range of supported languages and had a good uptime.

### 7.6.2. Ionic React

#### *7.6.2.1. Overview*

Ionic React is a native React version of the Ionic Framework. It is open-source and allows for development for both iOS and Android devices.

#### *7.6.2.2. Pros*

- Open Source
- Compatible with iOS and Android.

#### *7.6.2.3. Cons*

- Requires numerous plug-ins to properly function.
- Debugging can be a tedious task as error messages are sometimes vague.

#### *7.6.2.4. Reasoning*

Ionic React works well with the component-based pattern, since the technology works best by splitting components into pure components that can be reused throughout the system. The technology works well with the client-server pattern, since it does not need massive local storage space. The technology is also open source which is a key constraint on the development of the product. Ionic React was the only option for the front end component as it was specifically asked for by the clients.

### 7.6.3. ARCore and ARKit

#### *7.6.3.1. Overview*

ARCore is designed to assist with the creation of augmented reality (AR) components that seamlessly integrate the digital and physical worlds. We are using it for Android AR components.

ARKit is an iOS specific tool kit designed to create augmented reality (AR) components that brings the digital world to the physical world.

#### *7.6.3.2. Pros*

- ARCore: Works with multiple development environments, such as Android and iOS.
- ARCore: Large development and support community.
- ARCore: Maps the surroundings relatively accurately.
- ARKit: The best AR software for iOS devices.

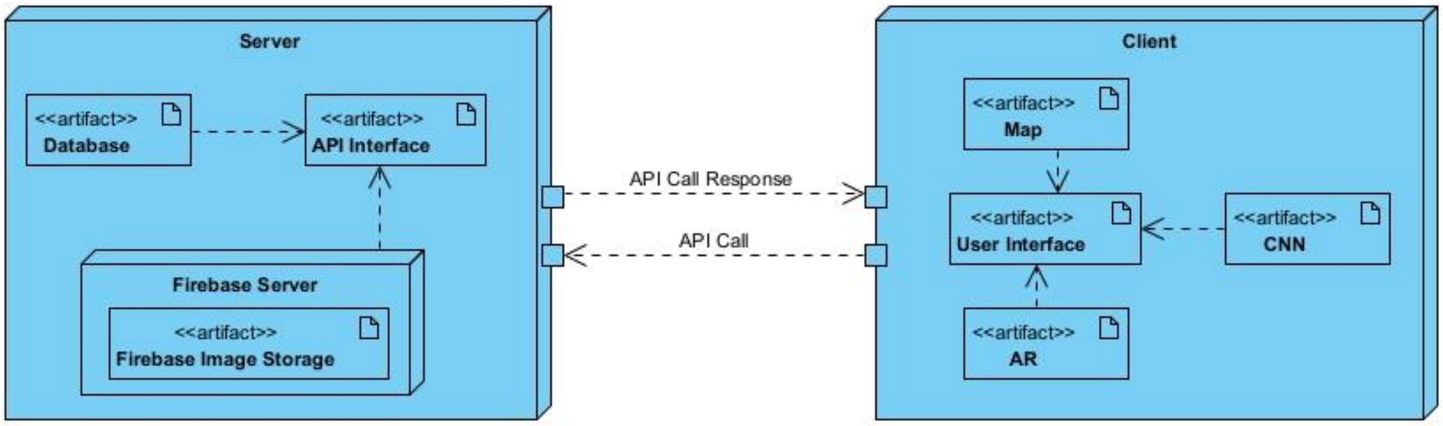
#### *7.6.3.3. Cons*

- ARKit: Only works with iOS devices.
- ARCore: Scanning accuracy could use improvement.

#### *7.6.3.4. Reasoning*

ARCore and ARKit work well with the client-server architecture and the cloud-based architecture. This is because the AR components are stored on the cloud/server side and then downloaded to the client side when the badge is viewed and/or earned. ARCore and ARKit work well with their respective operating systems and both allow for components to be separated from the rest of the product and features, which allows for improved error-management. It is an external technology that is important to the overall product, however, it does not have much of an impact on the core architecture. ARCore and ARKit were chosen as they were the best AR tool kits available that were largely open source. Since open source is an important requirement of the project, no other technology was possible.

## 8. Deployment Diagram



The Server is already deployed on the Heroku system. The front-end and the back-end only communicate using API calls, predominantly to retrieve or change information within the database located on the Server. The map and AR components in the client use external services to be properly rendered, whilst the CNN (Convolutional Neural Network) was trained offline, and accessed from the application itself.