SEM5640 Group Project
Report

Go!Aber

Submitted in partial fulfillment of
the requirements for the award of the degree of

MEng
in
Software Engineering

Submitted by

Student No.	Connor Goddard
Student No.	Helen Harman
Student No.	Craig Heptinstall
Student No. 110036072	Samuel Jackson
Student No.	Daniel McGuckin

Department of Computer Science
Aberystwyth University

December 2015

Abstract

Contents

List of Figures

Chapter 1

Project Overview

Following from the brief provided at the start of this project[3], we are able to clarify a detailed overview of the application requested by the client. A new system to be named 'GoAber' would allow members or 'participants' from a university to interact with other participants enabling them to record activity data such as step counts, distances travelled and heart rates.

The System means to allow a number of methods for entering activity data from sources such as Fitbit[1] and Jawbone[2] devices, alongside manual and external entry via an application API. The external API would then be able to cope with users who wish to use other devices such as smart watches or health tracking mobile apps.

Users should be able to access the system either via a vanilla log in or through their university emails, though the different sign-on systems of universities may have to be considered. With the data entered by users, they would then be able to interact with other participants on a deeper level, through challenges. With challenges, the overall purpose of the application is to enable staff and students of a wide set of universities to keep active, and give them incentive while comparing their efforts to others. The application will give each participant the opportunity to work as a team when performing challenges, by allowing them to become a part of a 'group'. For instance, a certain department or group of individuals at a university may want to be represented, and therefore compete with other groups.

Following on from groups and interacting with users, it is important that different levels of authorisation is followed for anyone logged into the system. There is a total of three level of authorisation required of the system:

- Participants- as mentioned previously, will be part of a group and university(community).
- Coordinator- A user with privileges to create challenges, and add users to a group that they created.
- Administrator- A higher level user, with all the abilities of the previous,

alongside the abilities to edit and remove participants activity data, groups, and communities.

The final high level features that the system should be able to perform is to use the activity data entered by participants to display some form of league table in order to rank both groups, institutions and individuals. Alongside this, a scheduled email system will be in place for users to receive updates on their performance in challenges, and produce emails when a user is inactive for a certain length of time.

The client has asked the system to be implemented in both JavaEE and .NET, allowing a more flexible choice of installation for universities that may want either a Linux or windows machine to run their server. Each running instance of the application should be self dependent, and only send or request data from the others when interaction such as challenges occurs. A thorough set of testing should be performed before the release of the application to ensure that the user interface functions correctly and that the bilingualism the user has asked for works, while the logic in the back of the application should be tested through unit tests and stress testing to ensure the application can withstand a number of concurrent users.

Finally, this report outlines the process of work completed during the project, detailing each stage of the project. Any modifications or clarifications to to the original client brief will be described in the following section.

1.1 Detailed requirements

The requirements spoken of here are based off the initial requirements specification document [3] and some of the key requirements are mentioned alongside their requirement codes. Before the design of the project began, a few clarifications was made for some of the requirements in a QA session with the client, all of which are described in the group project meeting minutes. These will be available in the 'docs/minutes' folder in the hand-in. To best detail the requirements of the application, parts of the application can be grouped by functionality.

1.1.1 Management of users, authentication and authorisation

The first of the requirements that should be available to the user is the registration and signing into the application. As mentioned in the requirements specification under D-FR1, the system should allow users to be added to the

system and to a group within a community. As discussed with the client in the first meeting, signing up for the site should be both available through the university credentials (SSO), or a vanilla log in, with a safe means of storage.

Alongside the registration of users, there should also be the functionality to edit users details and remove them. All levels of users should be able to perform these actions, though it was clarified in the QA session that only admin-authorised can edit and remove others users. Other administrator level activities only applicable to those users include renaming groups, removing groups, renaming their community, and deleting any other users' activity data. The auditing of other users' data is described in the auditing section of this chapter.

To clarify how users will connect through groups and communities, the example below highlights an example scenario:
A groups such as 'Computer science' would be contained within a community such as 'Aberystwyth University', and a participant would be added to Computer science by a coordinator that created that group. Then if for instance another group challenged Computer science the user's activity data would combine with others in their own group to compare their totals with the second groups. This is a similar story for communities, where totals of all participants of a given community could be compared with another community.

1.1.2 Activity data

The next subset of requirements required of the system is the integral need for activity data to be saved. As mentioned in the overview, the user will should be able to link Fitbit and Jawbone devices (of which data will be grabbed on a daily basis, or by syncing manually at any time), alongside manual entry of data. This has been outlined in D-FR2, which also mentions that there should be some means of saving all types of activity mutually.

This was another part of the requirements specification that was clarified with the client, where it was decided that categories of activity data could be expected as always a numerical format and should be dynamic to allow for more categories to be added. A final clarification here was that although dynamic addition of categories of data should be possible, three types (walking, cycling and running) will suffice at this point.

1.1.3 Data auditing

Alongside the entry of data, D-FR8 was a feature of the application that the client was intent on being implemented. The auditing of data should be

possible for administrators and for participants (where the participants are only changing or removing their own data), and notes should be left by the remover to allow for a reason to be left to a user wanting to know why data was modified or deleted.

An extension of the auditing data feature that was clarified again with the client is the auditing of all actions by the administrator. Unlike the data auditing, this would mean an auditing record should be created for any administrator level user that performs an action only available to them. Another highlighted clarification of this requirement is that all auditing should be final, and that no rollback functionality is required. For instance, if an administrator deleted data from a user, to undo this action would not be possible.

1.1.4 Challenges

Linking back to the way users interact in the system, challenges is one of the largest and most important aspect of the proposed system. C-FR1 to C-FR4 outline the challenges requirements, though again clarifications have been made since starting this project. Rather than assuming a challenge will be sent to another community or group for them to accept, instead the flow of challenges should be as follows:

- Coordinator of a group creates a challenge
- A coordinator from another group joins the challenge
- Both groups users can compare the competitions progress and statistics

In addition to the information about overall progress, users in a challenge should be able to see a league table within their own group. It was clarified in the QA session that summaries for each user should be available to all others in their groups, meanwhile outside their own group, only a very basic stat should be visible. This goes also for privacy of names and personal information, which should only be visible by people in their own group, although users will need control of their user name, which could be made anonymous if requested. As for higher level users, administrators will see all users details within challenges, and coordinators will see the same information but at a restriction of only overall data.

Upon completion of a challenge which will be decided by length of time of the event, it will be the responsibility of the coordinator that created the challenge to publish results and inform any participants taking part. Although this should be automated, initially sending out results should be completed

by the coordinator to give time for validation of the data. Alongside the emailed out results, data from completed challenges should be visible on any participants of the challenges' dashboard upon next log in to the application.

A smaller additionally desired but not essential feature discussed during the QA meeting was extra results for categories such as 'best walkers' etc.

1.1.5 Updates and emails

This brings this part of the requirements specification to how emailing will be handled. Emails are an additional though highly desired part of the brief responsible for both sending mail out for completed challenges and for reminding inactive users to register data. Where reminders and challenge news is sent out, the amount of time between each automated email should be modifiable by an administrator.

As per discussed with the client, the default times for each email type should be as follows:

- Participant progress emails- Weekly
- Missing readings emails- Daily
- Results of users- Manual

By combining the automatic tasks of sending emails and receiving data from external services, it should be possible to make use of a general scheduler to help perform a range of actions when the user decides.

1.1.6 User interface and internationalisation

Because the application will be a web-application, it is key that the application has an easy to use appearance that is preferably mobile friendly. The client has agreed that the default appearance using the Bootstrap[4] is allowed here. In addition to a friendly user interface, internationalisation is a required feature of the system, and should provide the user the ability to switch between at least the English and Welsh languages with opportunity for more.

1.1.7 External endpoint

Although part of D-FR5 which covers the requirement of receiving data from a variety of means, a special mention to the requirement that the application should have an API endpoint. The API (to be SOAP) will be required to

allow interaction between different servers running the application in order for different communities to communicate and send challenges.

Alongside this, the API endpoint should also be able to receive generic activity data to be inserted into the system (though the client has clarified that this part of the API will only require POST, as to only receive data).

1.2 Desired and required libraries

In addition to the requirements of the application, the client has requested that third party software should be made use of to encourage a faster implementation of the product. Uses of third party software was discussed during initial meetings, with some group members already having a good knowledge of possible email libraries and OAuth libraries to help with the connectivity to Fitbit and Jawbone services. Both of which will be discussed further in this report, with reasoning for why such libraries were chosen.

Chapter 2

Development Methodology

Chapter 3

Design

This section outlines the initial designs for the system to be produced. As this project is being developed using a Scrum based development methodology, there is no concrete up front design for the system. Much of the design presented here was created during the initial stages of the project and therefore has been subject to change throughout the implementation stages of the project. However, we were not going to start development on the project without any design whatsoever, especially since nearly all team members were unfamiliar the technologies we were going to be working with. Unless otherwise stated all designs are applicable to both the .NET and Java EE systems.

3.1 Use Case Diagrams

The first real piece of design to be undertaken was a detailed analysis of the requirements specification. This initial step aimed to tease out what was likely to be the most challenging, unintuitive elements of the project. Through this discussion we were able to draw out what we thought were the major use cases for the different system users.

Figure 3.1 shows the use cases for registering and logging in a user to the system. Note that all types of the system user can perform these actions regardless of their role. By "vanilla" login/registration we mean a custom login system specific to site that does not interact with another site via SSO etc.

Figure 3.2 shows the different actions that can be performed to administer a user account with the GoAber system. These are broadly pretty common sense and would be the sort of actions normally expected of system such as this. Note that in figure 3.2 administrators can do everything a participant can plus modifying their privileges. Changing user privileges is the only action that cannot be performed by anyone other than an administrator. "Deactivating" a user account will leave an audit record in the system but will remove all activity data as well.

The next use case diagram (figure 3.3) is arguably the most important in the series. This gives a loose summary of how the users will interact with their activity data in the system. It also shows which actors have permission to carry out particular actions. As mentioned before, administrators can carry out all actions that co-ordinators and participants can. Co-ordinators can only view information about themselves and others, much in the same way as users, but can obviously perform CRUD actions on their own data.

The diagram shown in figure 3.4 shows the interactions that can be carried out on groups of users. Administrators have the permission to perform CRUD operations on a group and have the ability to add participants to a group. Participants are only able to view a summary of other people in the group.

Use cases for challenges are shown in figure 3.5. Here, the major differences to be aware of is the difference between what actions a user and coordinator can perform. Users can only view information about challenges while coordinators can setup and edit challenges between groups and communities. Administrators will be able to perform all of these actions.

Figure 3.6 shows a couple of additional participant use cases which will be required in order to allow users to authorise their devices with our system. Participants should be able to authorise their devices another system via OAuth.

Finally, figure 3.7 shows some additional use cases for system administrators. This figure simply states some additional actions which were not included in the other use case diagrams. Administrators should be able to schedule there emails sent out from the system as well as edit and delete users.

3.2 System Architecture

This section discussed the high level system architecture that we conceived in the very early stages of the project. Figure 3.8 shows a formalised version of the early system architecture. We in no way expect the final design of the implemented system to accurately reflect this diagram. This diagram was a useful by product of early discussions to try and understand how the final system might hang to together. This was a key discussion that lead to us identifying parts of the system that we did not readily understand or that we found to be ambiguous.

This design was originally produced in rough on a white board, allowing us to shuffle key elements around until we were in agreement on how each part should work. Afterwards the diagram was formalised into figure 3.8 to be preserved as a design artefact. This overview is meant to be indepen-

dent of the technology used (either .NET or Java EE). This, combined which the fact that a Scrum based methodology can and will allow us to be flexible with the design are the major reasons why the final system will almost inevitably differ from this diagram. However, as mentioned, it played an important role in getting all team members on the same page before diving into implementation.

The system in diagram 3.8 can be split into two parts: "our system" on the left hand side and the "outside world" on the right. The outside world consists of regular human users (participants, coordinators, and administrators) but also includes other computer systems. For example, other GoAber systems need to communicate information about users and challenges between one another. Other systems that need to be communicated with are the Fitbit/Jawbone servers and with generic SOAP input.

The diagram shows several connections from our system to the outside world. The most obvious one, in the top centre of the diagram, shows that users can connect to a front end website. Conceptually this component is broken down into two parts: the views (what the page looks like) and the controllers (how the views get built displayed). The models sit further back in the system and are accessed by the controllers.

Below this component is the SOAP web API. This provides an access point for remote systems (i.e. non-human users) to communicate with our system. This includes both other community servers and potential other devices.

The third point of access in the bottom centre of the diagram is the most complicated part of the external communication systems. This shows a scheduling component, inside of which is nested a data collection component. The scheduling component will be responsible for firing off events both internally and externally. For example, internally this will be responsible for closing a challenge on time. Externally it will be used to periodically request data from the Fitbit and Jawbone APIs and as a timer to send out emails.

Sitting behind these front three layers is the business logic and data models for the system. These are shared by all three of the components described above. This part of the diagram is deliberately left more vague than the other parts of the diagram. As mentioned in the preceding chapter, we are using a Scrum based methodology and producing a big up front design would be going against its guidelines. Additionally this section of the system is highly likely to be specific to .NET and Java EE. For this reason we will choose to keep the low level design decisions of this part of the system up to the developer. Broadly speaking our approach in this project will be to keep the .NET and Java EE systems structured similarly. Both systems should share the same model structures, unless the specific implementation forces us to

change for some reason.

3.3 Database Design

The second major piece of design that we undertook in preparation for the implementation of this project was to come up with a entity relationship diagram which would form the basis for the model code in both versions of the application. Once again, in practice it is likely that this design will need to change once we become more familiar with the two different technologies we are using.

This, like the high system architecture in the previous section was very important to do early on in the project. It helped to solidify how we were going to represent objects on data in our system and gave good starting guidelines for the team members who implement these this design.

Starting to the right of the diagram is the *ActivityData* entity. This is perhaps the most important entity in the entire system. An *ActivityData* item is a single piece of activity data for a particular user. An *ActivityData* entity is associated with a particular system *User* and also has a reference to a *CategoryUnit* entity. A *CategoryUnit* is simply a linking table between the categories (such as "running" and "swimming") and units (such as "steps" or "strokes").

The system is designed in this way so that there is decoupling between the value of the data stored in the system the category or unit it belongs to. Every activity data item will simply be stored as a numerical value. The interpretation of that value is determined by what the associated category or unit is. This allows the database model to be flexible to the number of different types of activity data that the client may wish to store in the system.

This formulation has several distinct advantages. Firstly with this approach there is no need to introduce null entires into the table as you would have to do the type of the data was store alongside the value itself. Secondly, it means that it would be easy to add the ability to introduce new categories and units should the customer require. Thirdly this makes the system almost completely independent of the type of data that a user might want to store. As long as the activity data item is a numerical value, no changes to the database structure are required to introduce the new type. The only exception to this is if the new type of activity data to be introduced happened to be categorical instead of numerical. But even in this circumstance another table could be easily created (e.g. *CategoricalActivityData*) in order to support it.

Moving onto other parts of the model; in the centre of the diagram is

the *User* model. This will store almost all of the information about a user (name, email etc.). A user record also has a link to a user credentials table, which contains their password for the system and other authentication data. Additionally users also have a user role associated with them. The user role specifies what type of user they are and permissions they have (e.g. participant, coordinator or administrator).

Along side this user data a user may also register several devices. The devices entity stores the data about a specific device that a user has connected with the site. Each device has a device type. The device type stores the details for connecting to a specific third party site that can be polled by our code for activity data. In this project those sites will be limited to Fitbit and Jawbone.

The top of the diagram shows the tables that will be required to implement the challenges portion of the system. Each user can be associated with a group. A group conceptually a list of users. Each group belongs to a single community. Groups can have challenges associated with them. A challenge entity contains information such as the start and end date and the type of activity data that is associated with it. Participants in a challenge are linked using the *UserChallenge* entity.

3.4 Activity Diagrams

Alongside the other diagrams presented in this section we also found it useful to produce some state diagrams to try and get a better idea of how a user will transition from one state to another around the site.

Figure 3.10 shows a activity diagram for authenticating a user in the GoAber system. There are several different paths that a usr should be able to follow though the login procedure. If they are already registered they can directly login. If not they my first register with the system before proceeding. In either of these cases if their details fail to validate they are redirected to the appropriate page.

Figure 3.11 shows the workflow for both a participant and an administrator needs to pass through in order to delete activity data from the system. In the case of a participant they should be asked to confirm the deletion. If the user is an administrator they should be also asked to provide a reason why the data is being removed.

The next figure (3.12) shows the states the workflow for authorising a users device (e.g. a Fitbit or Jawbone device) for use with our system. Users should be able to view which devices are connected to the system and revoke access is desired. They should also be able to add a new device and any

errors in the case that the external site returns a failure.

Group management activities are shown in figure 3.13. Most of these actions are fairly self explanatory but there is a slightly non-trivial case where we wish to add a user to the group. In this case we must check make sure that they are removed from their old group as they can only ever be affiliated with one group at a time.

In figure 3.14 the activity flow for entering data into the system is shown. Activity data can be entered manually by participants in which case there data is validated on submission. With external systems there is the potential possibility of a connection failure and that we get bad data back. This could be either from a device API or manual input for a third party device via the SOAP API.

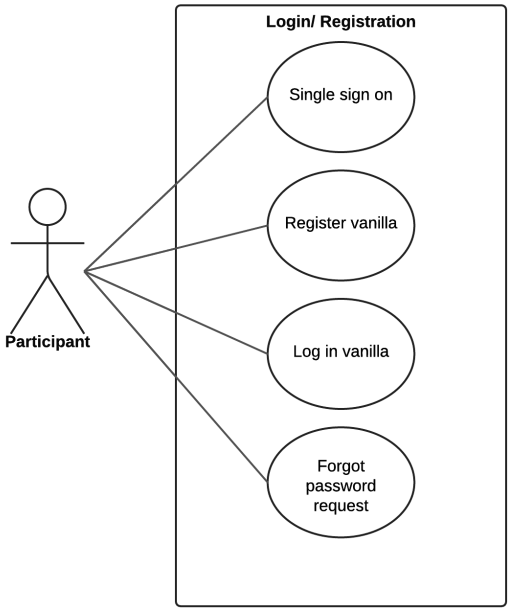

Figure 3.1: Shows the different login & registration actions that system user (participant, coordinator or administrator) can take. These actions are associated with D-FR11.

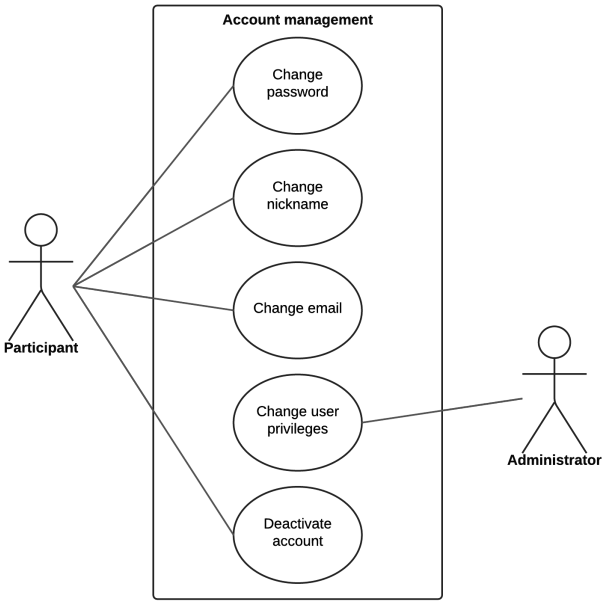

Figure 3.2: Shows the different account administration actions that can be performed by a participant of the system. This references requirements D-FR1 and D-FR10.

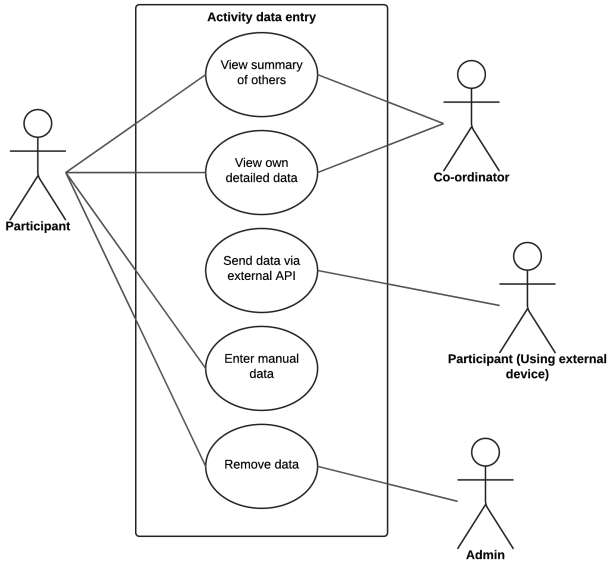

Figure 3.3: Shows the actions that can be performed by the three types on activity data. These were taken from requirements D-FR5, 7, 8, 9, 10

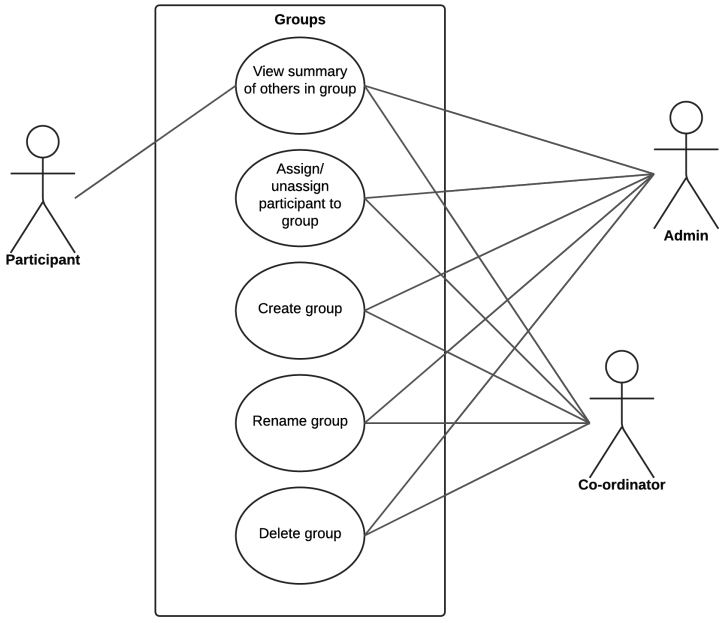

Figure 3.4: Shows how actors will interact with groups of users. This is in relation to requirements D-FR1, D-FR10

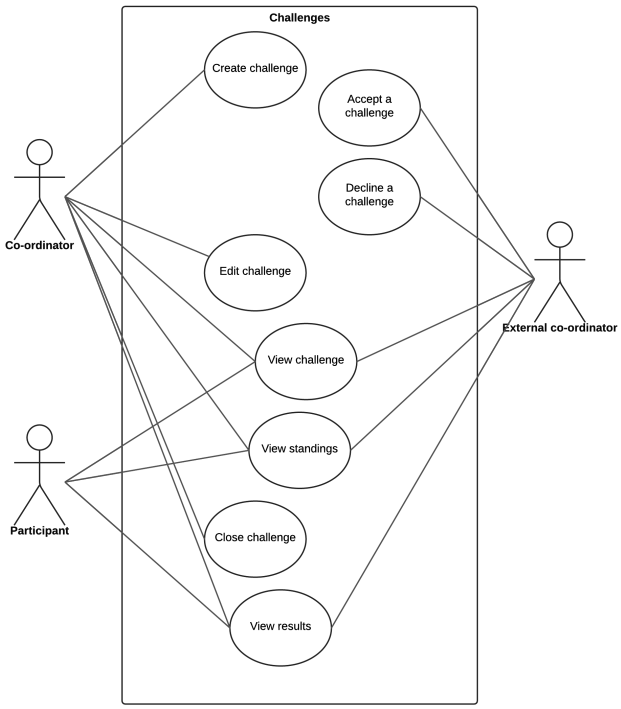

Figure 3.5: Shows how actors will interact with the system in terms of challenges. These reference requirements C-FR1-4 and E-FR1

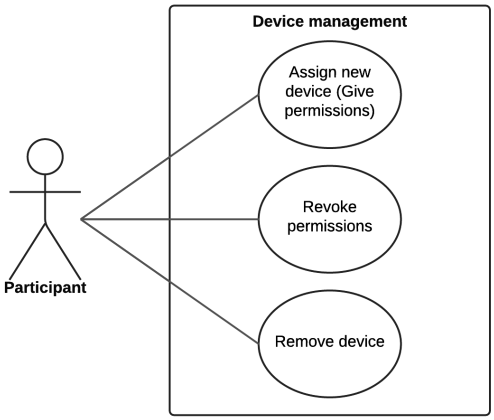

Figure 3.6: Shows how a user can authorise a device with an external system (e.g. Jawbone/Fitbit). This relates to requirement D-FR3.

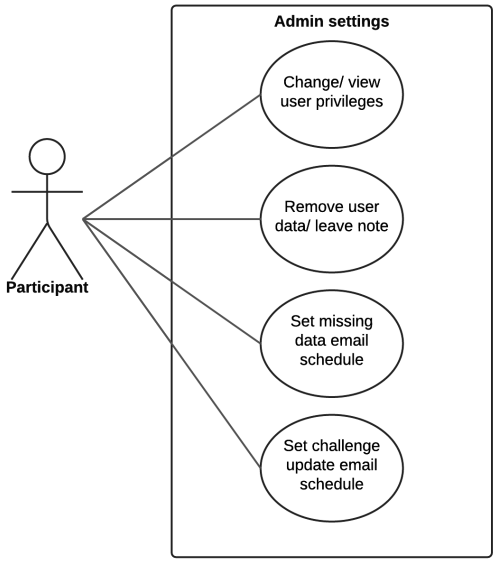

Figure 3.7: Shows some additional uses cases that can be performed by a system administrator. These action are all taken from requirements D-FR1, D-FR8, E-FR3 and E-FR4.

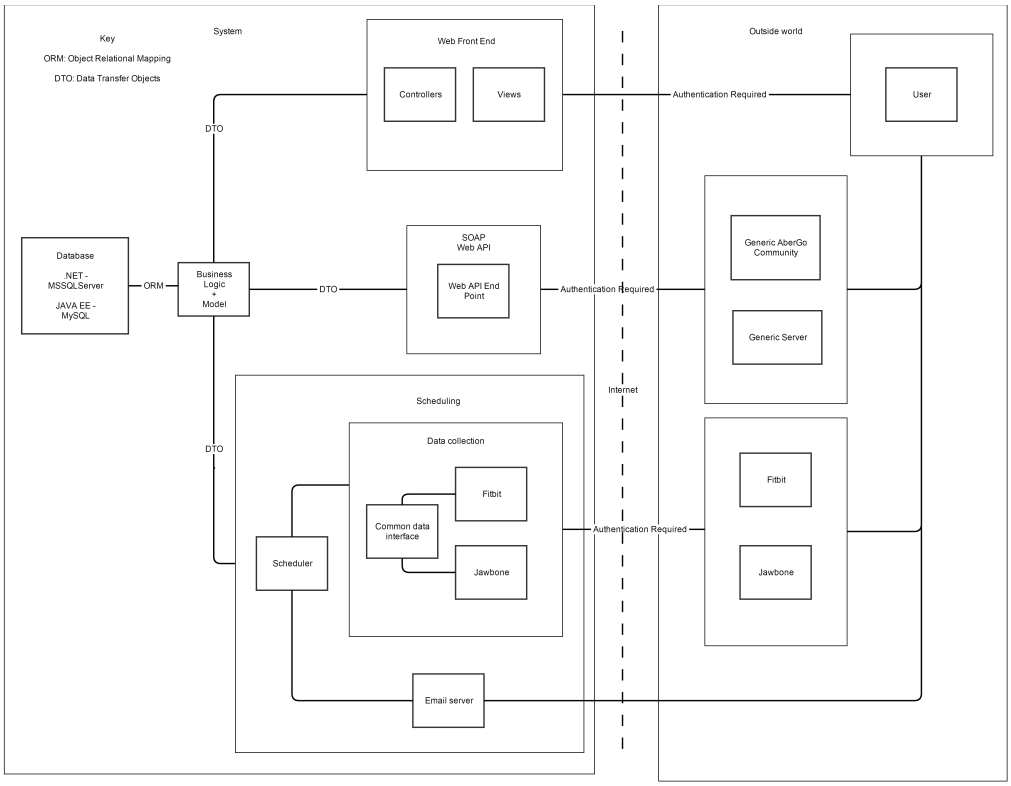

Figure 3.8: High level conceptual overview of the proposed system.

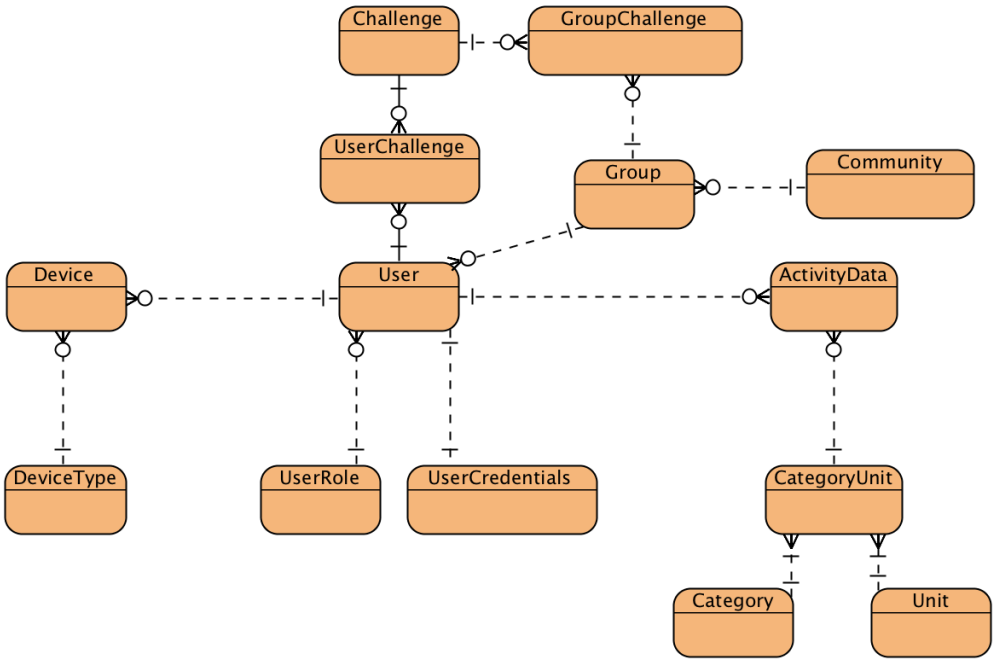

Figure 3.9: A entity relationship diagram showing how the data model in both of the applications interact with one another.

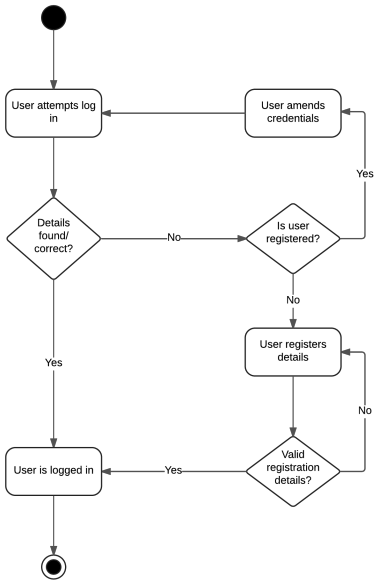

Figure 3.10: An activity diagram showing the states that a user can transition between when authenticating with the system.

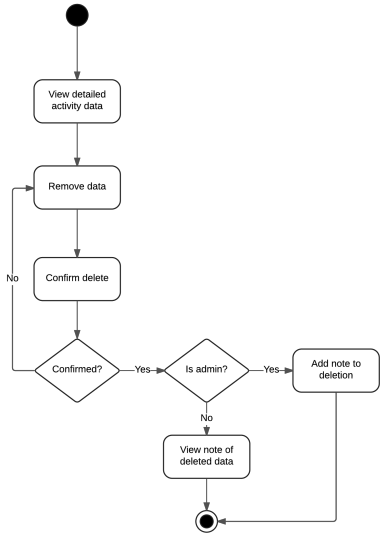

Figure 3.11: An activity diagram showing the states a participant/administrator passes through in order to delete activity data.

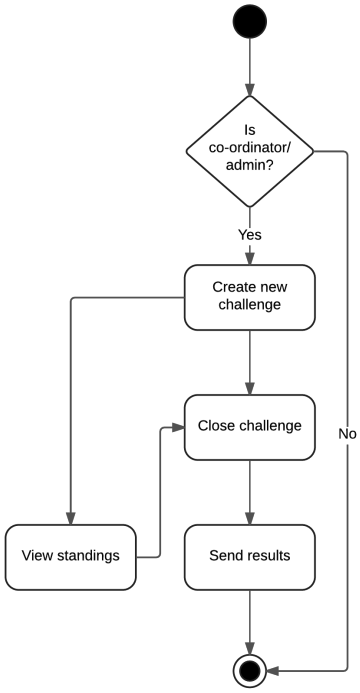

Figure 3.12: An activity diagram showing the workflow for device authorisation.

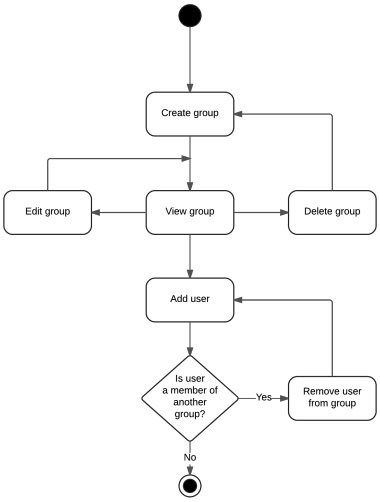

Figure 3.13: An activity diagram for the management of user groups in GoAber.

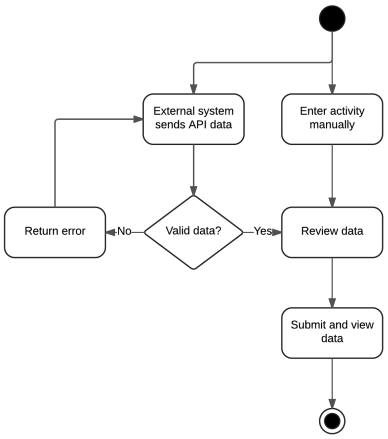

Figure 3.14: An activity diagram for manually inputting user data into the sytem.

Chapter 4

Implementation

Chapter 5

Testing

Chapter 6

Project Status

Chapter 7

Critical Evaluation

References

[1] Fitbit. Developer api. `http://dev.fitbit.com/uk/`, year =.

[2] Jawbone. Jawbone up api. `http://jawbone.com/up/developer`, year =.

[3] Nigel Hardy Neil Taylor. Go!aber requirements specification. *SEM5640 Group Project*, 2015.

[4] Twitter. Get bootstrap. `http://getbootstrap.com/`, year =.