# Portfolio A

## Overview

Our project is aimed at helping the University of Bristol's Be:Active program by allowing students to have access to a phone app which will help them get more involved in sport activities at the University. The main goal of the app is to be an information point for students as well as a way for them to show interest and register attendance in sports activities.

The client of our project is the University of Bristol Sports, Exercise & Health department. Their main motivation for having us develop this app was the fact that a lot of sports activities happen outside of the main sport building which makes it hard for them to keep track of attendance. Furthermore they wanted to further increase engagement in sports by having the phone app be a point of centralised information. It would make the accessing of information (such as timings of activities) to be more user friendly due to the website being hard to navigate, especially on a mobile device. One of the potential additions is integrating the Legend API which allows members to book courts/classes, check their membership status, sign up to clubs on the app.

Our aim is to create an app that will allow students to access these features through their mobile phones. Moreover, we want to make an app that students will want to use instead of just getting all their information about sports from other social media apps such as Facebook. The app may potentially include social media features which will allow members to communicate with each other as well as the ability to comment on posts or photos posted by students. The app may also feature a learning platform that will allow students to check workout routines through videos and pictures, encouraging more students to be active at the university.

We have decided to base the application on Android since it is open-source and free to use. Android is also easy to setup and compatible on a wide range of devices which would make the app accessible to a larger student population. The programming language we decided to use was Java due to the fact that all of our group members are already familiar with Java to some extent as well as Java’s compatibility with Android development.

At the end of our project we hope to make an app that will be a central point of information for sports at the University of Bristol, eventually releasing it to the public and allowing the students to use the app themselves.

## Requirements

The first step in establishing the requirements of a project is to identify the stakeholders involved. They are divided into two categories: the people who are impacted by the system and the ones who impact the system. In our case, the people who will be impacted by the system are the users, the students with gym memberships and students who have an interest in sport events. The higher-ups in the University of Bristol's B:Active programme and the societies are the ones that will impact the system.

The next steps are to identify some "user stories" for each stakeholder and their flow steps. For our project we have identified the following ones:

**1. For students and external users:**

"As a user, I want to be able to view all available events in one place, so that it's easier to find what I'm interested in."

"As a user, I want to be able to view the details of the events, so that I can make an idea of what is going to happen in the event and decide whether I want to participate or not."

"As a user, I want to be able to book an event, so that I can let the staff know I want to attend it."

These user stories have the following basic flow:

1. Download the app
2. Browse through the events
3. Select an event that you may be interested in
4. Read the details of that event
5. Book the event

An alternative flow may be:

1. Download the app
2. Filter the events
3. Browse through them
4. Select an event
5. Read the description
6. The user aborts and the event is not booked.

**2. For the higher-ups in the B:Active programme:**

“As the health and wellbeing department staff, we want to be able to register attendees for events, so that we can keep track of them easier. ”

1. User attends event
2. They mark the event as attended on the app
3. The staff is able to see who has attended the event

“As the health and wellbeing department staff, we want to be able to manage events inside the application, so that we can add and delete events depending on user privileges (administrator/user).”

1. User books event
2. Event gets cancelled
3. Gym staff deletes the event from the app
4. The user receives a notification about the cancellation

“As the health and wellbeing department staff, we want to be able to receive feedback for our events, so that we know what to improve.”

1. Student attends an event
2. After the event is finished, they write a feedback for that event on the app

Exceptional flow:

1. Student books an event
2. They attend the event
3. They don’t want to write a feedback, so they ignore it.

The story we have chosen to be the main one is: “As a user, I want to be able to view all available events in one place, so that it’s easier to find what I’m interested in.”
The next step is to specify the atomic requirements, which are then divided into two categories: functional and non-functional requirements.

For the main story, the atomic requirements are:

“The user must be able to download the app.”
“The user must be able to browse through all the available events.”
“The user should be able to filter the available events.”
“The user shall be able to read the description of the available events.”
“The user must be able to book an event.”
“The user shall be able to cancel the booking for any event.”
“The user must be notified in case an event gets cancelled.”
“The staff should be able to receive feedback after each event.”

Lastly, we need to identify the functional and non-functional requirements, written in natural language. The functional requirements are:

BOOKING-1.0 : The user shall be able to download an application in order to allow them to use all the facilities provided by the system.

BOOKING-2.0 : The user must be able to browse through all available events from the timetable, in order to allow them to decide which events they want to attend.

BOOKING-2.1 : The user should be able to filter the events by their interest, in order to be easier for them to find something that they would like.

BOOKING-2.2 : The user shall be able to read a description of any of the available events in order to help them decide if they want to attend the event or not.

BOOKING-3.0 : The user must be able to book any available event in order to let the staff know they want to attend it.

BOOKING-3.1 : The user shall be able to cancel a booking for an event in order to let the staff know they are no longer able to attend it.

CANCELLATION-1.0 : The staff must be able to cancel any event, in case the event won't take place anymore.

CANCELLATION-1.1 : The user must be notified in case the staff cancels an event that the user was going to attend, in order to let them know they are not able to host it anymore.

FEEDBACK-1.0 : The staff should be able to receive feedback from the attendees of their events, in order to allow them to improve their services.

The non-functional requirements are:

EFFICIENCY-1.0 : The application shall meet the agreed response time and capacity requirements.

USABILITY-1.0 : The application should be capable enough to support 100 000 users without affecting its performance.

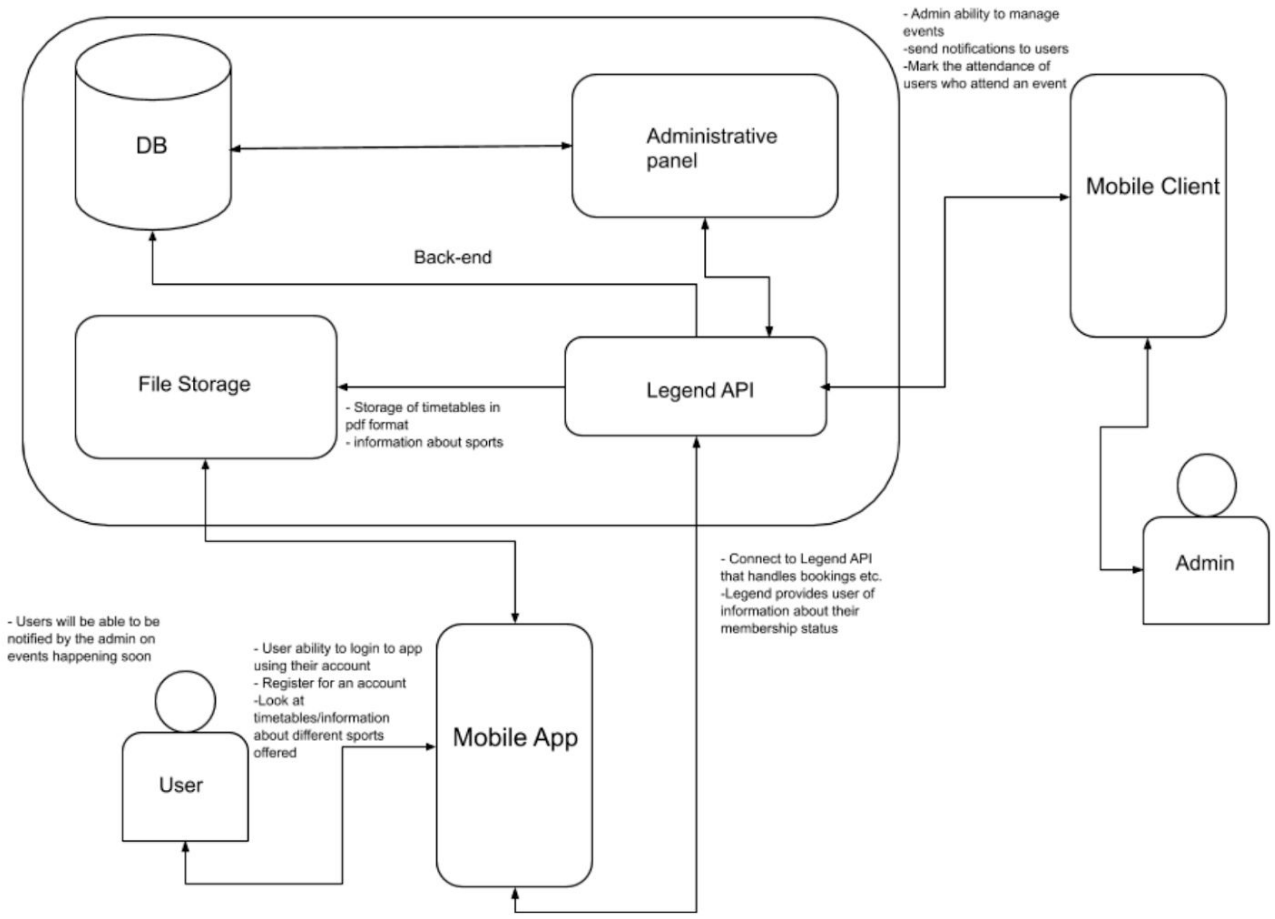
USABILITY-1.1 : The application shall be able to accommodate increased users and volumes.

SECURITY-1.0 : The application must provide access only to legitimate users.

## Architecture

The architecture models the basic structure of all processes that will be occurring in the background of the application. After meeting with the client, we have outlined the basic processes that are needed to take place in order to make the app sustainable. The app will be available for everyone to download. The average user (i.e. Student) should be able to create an account and use that account to sign in with. This means that the account information will need to be stored in a database. The API will allow communication between the user and the server/database. The user will also be able to look for events, sign up for events and mark attendance when going to events/classes. The mobile app will continuously communicate with the API when the user carries out any of these actions. There will also be admin accounts which have the ability to manage events i.e. create, cancel, etc. and send notifications to users about upcoming events. The processes occurring here are dynamic i.e. sending notifications to users. The architecture will include a file storage system. This is because, as mentioned in the client meetings, the user should be able to view timetables of fitness classes. These timetables are currently stored as pdf files. Therefore we would require some sort of file system in the back end. We have also discussed the possibility of a “social media/learning platform” that will mean users can store pictures and videos. This again will require the use of the file storage system.

The main API currently being used by the University’s sport services is known as “Legend”. Essentially, Legend allows users to view their membership status, make bookings, cancel bookings, etc. The API will allow admins to push notifications to all users about upcoming events. The API will be used to communicate with the file storage system and the database holding information about users.


- Admin ability to manage events
-send notifications to users
-Mark the attendance of users who attend an event
DB
Administrative panel
Mobile Client
Back-end
File Storage
Legend API
- Storage of timetables in pdf format
- information about sports
Admin
- Connect to Legend API that handles bookings etc.
-Legend provides user of information about their membership status
- Users will be able to be notified by the admin on events happening soon
- User ability to login to app using their account
- Register for an account
-Look at timetables/information about different sports offered
Mobile App
User

## Development Testing

Our development testing strategy follows the principle that testing is used to show the presence of bugs and not the absence of bugs. We will go about adhering to this principal by following the protocol of first writing the tests before writing the code, making sure these tests revolve around identifying if the code is conforming to the specification and if the product is actually being built correctly. After doing this, we will continuously re-run, refactor and optimize the tests. The correctness of these tests will be ensured by using synthetic data that is chosen using methods such as equivalence partitioning to provide complete coverage of test cases without needing to try all possibilities. All automated tests will be implemented using JUnit.

A core function of the project is to have an application that allows users to find, create, delete, update and register for events that are organised by the university Health and Wellbeing Department that aren't currently integrated into the university Legend system. A core component of the application is to have different levels of user privileges and that users who had the administrator privilege had the ability to create, delete and update these events while more importantly be able to register attendees. This was identified as a core component by establishing that this was the main aspect of the clients vision of a 'minimum viable product' which aimed to solve the problem of not being able to efficiently track analytics of events that aren't currently integrated into the system.

| Test | Type | Input | Reason | Expected Outcome |
|---|---|---|---|---|
| Event creation | Unit Test:<br>- Automatic<br>- In-memory database and JUnit | User with admin privileges creates an event using application interface.<br>- Date<br>- Location<br>- Time<br>- Capacity | Required in client specification. Testing for input validation and prevention of duplication of events. | If event doesn't already exist/overlap with events at that time and that location then an event is created and displayed on application and imported into database. |
| Event deletion | Unit Test:<br>- Automatic<br>- In-memory database and JUnit | User with admin privileges deletes an event using application interface or event is deleted after it expires. | Required in client specification. Making sure application doesn't continue to display out of date events but information of past events remain. | If event exists, when manually/automatically deleted it is removed from application interface but not from database. |
| Event attendee registration. | Unit Test:<br>- Automatic<br>- In-memory database and JUnit | User with admin privileges registers attendees using application interface. | Required in client specification. Making sure number of attendees doesn't exceed capacity. | After registering attendees, information of event is stored in database. |
| Separation of user privileges and restrictions on user access. | Unit Test/Security Test<br>- Automatic<br>- MockWeb Service | User interfaces and privileges are displayed differently depending on account. | Required in client specification. Making sure non-admin users can modify events without permission. | Different application interfaces and actions are displayed depending on account privilege. |
| Check all user inputs/event data is reflected in database. | Integration Test:<br>- Automatic<br>- In-memory database and JUnit | Integration of all unit tests. | Making sure all unit tests /inputs are reflected in database records. | Database records are updated according to user input. |
| Application event limit handling. | Stress/Load Test:<br>- Automatic<br>- In-memory database and JUnit | N/A | Making sure database can synchronize with multiple event creation/deletion/registration at the same time. | Information is synchronized in the database. |

## Release Testing

Our release testing strategy follows the principle that a test should identify if the current product/a new feature is meeting a user story or client requirement and ultimately that the product is the correct solution to the client's problem and meets all client requirements and user needs. We will go about adhering to this principal by emphasising user experience testing, constantly getting feedback from the client and running both automatic and manual tests using methods such as acceptance testing and automated API calls. After doing this, we will continuously re-run, refactor and optimize the tests based on real user assessment and the products reliability, performance, usability and robustness.

A core user story of the product is for students to be able to access an application that compiles all university wide and society events that are available to them and act as a focal point of information regarding the societies they are interested in/apart of, their upcoming events and a way to navigate and register to such events. This is as a core user story as we identified along with the client that there is currently no coherent way for students to find, register, show interest and get details of university and society wide events other than navigating Facebook and searching for these events and specific societies.

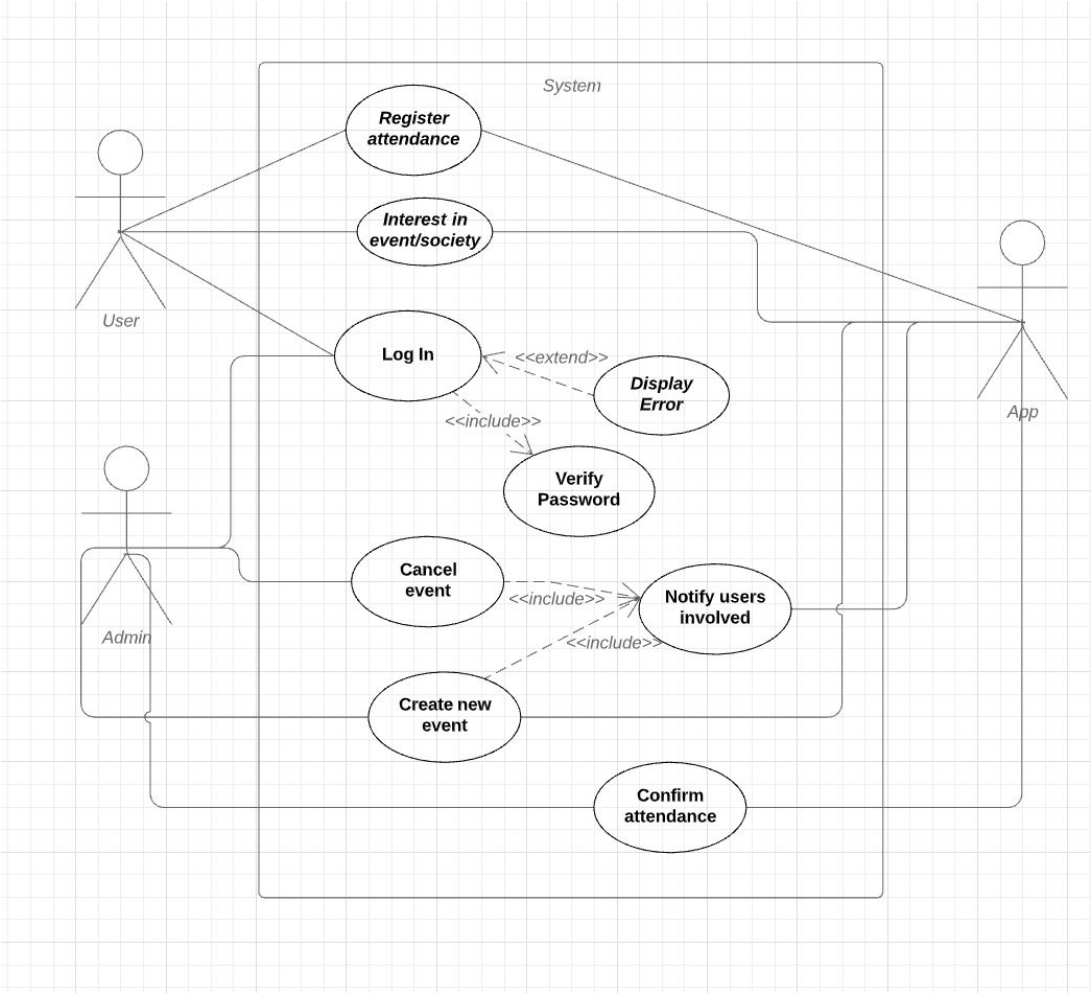
| Test | Type | Input | Reason | Expected Outcome |
|---|---|---|---|---|
| Application works on different environments and displays properly on all Android devices. | Production/System Testing<br>- Manual | N/A | Important to make sure the application works on all relevant devices so all users can access it. | Application works coherently across all relevant devices. |
| Automated API/CRUD Calls | Production/System Testing:<br>- Automatic | Automated calls using JUnit. | To make sure system and features work as they should when handling api/crud calls. | API/CRUD calls are handled correctly. |
| Implementation of additional features and release verification. | Regression/System Testing<br>- Automatic and Manual | MockMVC | Making sure that additional features interact correctly with system even if the system isn't fully functional/developed. | New features don't cause errors in fundamental features. |
| Basic Release Test | Smoke Test<br>- Manual | Manual user input/interaction | Making sure most important functions work and allows us to verify the stability of product and allows us to continue testing other components. Allows us to monitor server and application metrics. | Product complies with most basic user requirements and meets user story. |
| User/Client Interaction | Deployment test | | Provides testing environment that can't be replicated in an automated/lab environment. | Reliability, usability and robustness of application is confirmed.<br>- Feedback is taken from user. |
| Client/User Audit | QA Test:<br>- Manual | Client/User manual interaction with application. | To make sure user/client needs are satisfied and that the correct processes are being taken to fulfill the client/user story. | Product complies with most basic user requirements and meets user story. Feedback is given for continuous improvement. |

## OO-Design and UML

This UML case diagram describes the process our minimum viable product should contain. After the meetings with our client we came to the conclusion that these features were the most important and should be the ones initially implemented to reach the goals of the project.

There are two types of users in our app, a regular user (students) and admins. Both of these types of users will have to sign in when using the app and our app will always check if the password is correct. If the password is incorrect, they will be notified as shown on the case diagram with the <<extend>> arrow.

The main functionality the regular users are meant to have is the ability to register interest in societies and sports events as well as registering their attendance if they have attended a specific event. The main functionality of the admin is the ability to create and cancel events while being able to notify all interested users about a new event or a cancelled event. The <<include>> tag specifies that each time an event is created or cancelled, the admin will automatically notify everyone by mass email or a notification on the app. They are also able to confirm attendance of the students to send to the sports center.



The UML diagram shows the classes we intend to use to complete the MVP. We can see a User class which is the parent class of the Student and the Admin class, which are the only types of users using the app. Each user has a login ID as well as a password which will be checked during the login.

We use composition to describe the Event and Society class in relation to the Admin. Events and societies cannot exist without the admin creating them initially and performing administrative roles such as creating or cancelling events. This is also true for users with their interested events as well as the respective societies they follow.

We chose to use this model as it shows the core functionality of the app, everything implemented after this will rely on this main functionality so it is important that we implement it properly. Furthermore, it is the main functionality asked by the client to be implemented.