Overview

Client

Our client is University of Bristol Computer Science Society (CSS), a group at the University of Bristol that represents the computer science student body and arranges events such as socials and talks related to computer science for people at the University of Bristol. They also run a mentor scheme, advertise job openings exclusive to students that are interested in computer science. As our client is a representant of the student body, they take a lot of care when it comes to the welfare of their members. So a way to communicate with their members is important. CSS also host very large events where attendance is normally in the hundreds. So they have to make sure that events are organised very well and very efficiently. CSS is run by a committee of members who are elected by the student body, all committee members are themselves also computer science.

Application Domain

This app will be mainly used by students who are members of the Bristol Computer Science Society, although all University students will have access to it. It will be used as a social media type app where members of CSS will use it daily to view events and updates posted as well as engage with the new updates. Due to it being an app, it can be used anywhere, most of the time regular computer science students will use the app in their free time anywhere. Where as admins and committee members of CSS will use it as a means to organise and promote events, and so it can be used in meetings between CSS committee members. This application will exist in the quaternary sector as it provides information services to the people who use it.

Problem to solve

Currently, CSS are using facebook to organise events. The client has a preference to shift to their own platform to give them more control, ease and flexibility when planning events. The main problem that they put across to us is that, currently while using Facebook as a platform to organise events the events organised were not reaching every member of CSS. Due to many people not having Facebook. This severely affects the outreach of CSS. Another problem is that the Wiki that CSS developed is not known by many people and as such was not being used. We want to incorporate this feature into the app so users can view it, and engage with it by adding further information to the Wiki.

Vision for product

Our solution to the problem is to create a system in the form of a mobile app that is easy to navigate. Users will be able to interact with the app in a way that enables them to use all the features in the app. Users will be able to login using their “MyBristol” accounts and create their own personalised profile which will allow them to be identified when interacting or posting onto the app, by using their “MyBristol” accounts it means that many more people can access the information to events held by CSS. All profile details will be stored in our own databases that we create. By also adding the Wiki feature to the app, it will also become more accessible CSS members and will be used and engaged with more.

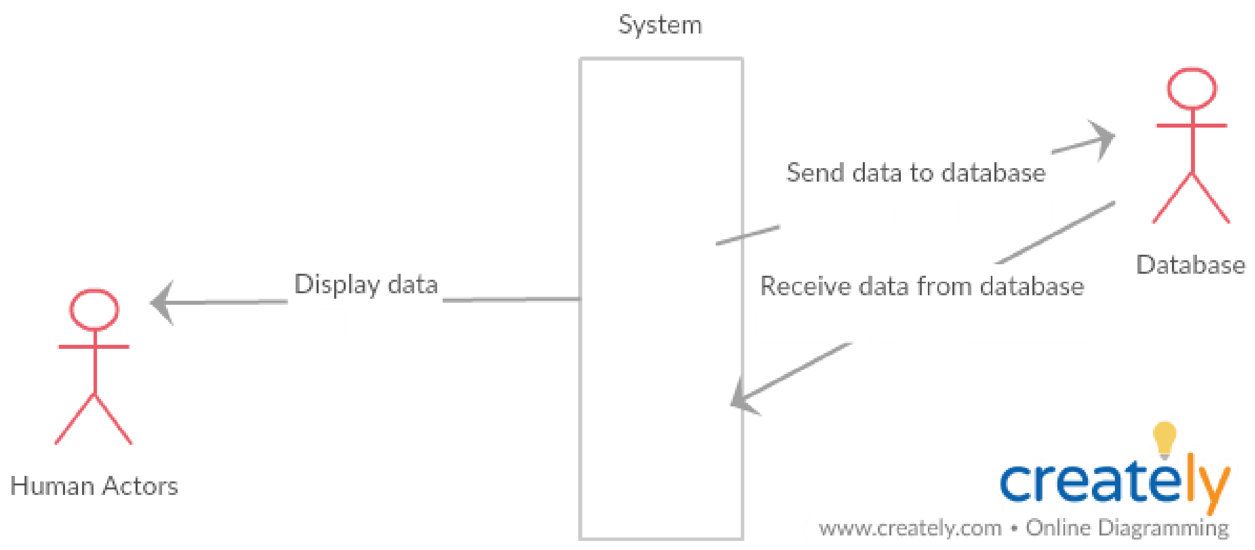
Requirements

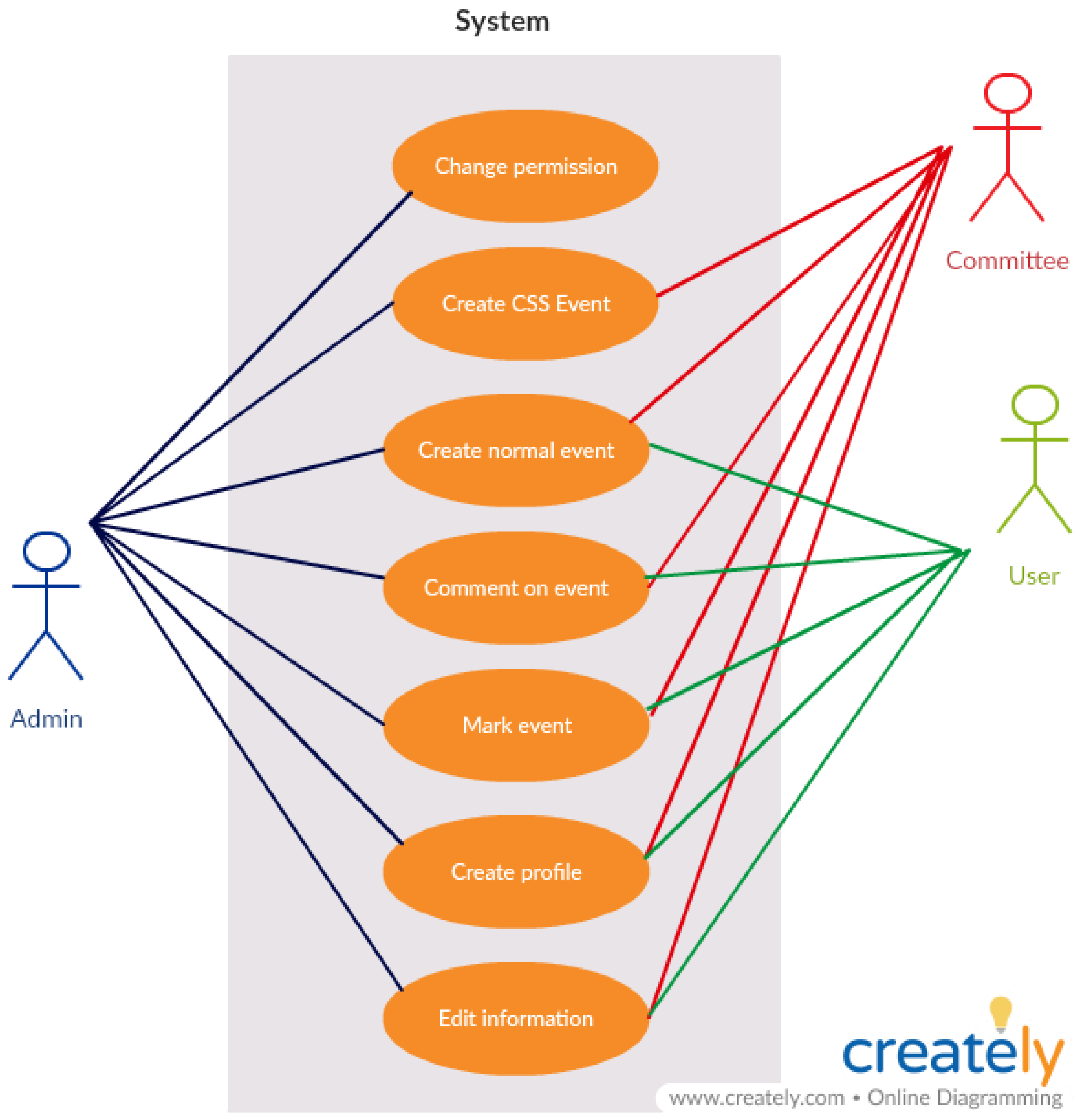
Key Human Actors:

Users : Everyone who has an University of Bristol account, but most of the users will probably be CSS members and Computer Science students, given the app's content.
Committee: All the members of the Computer Science Society committee. They will be able to create CSS event and CSS planning events. CSS planning events are hidden from users.
Admin: The person that has access to all of the app's functionalities. On top of having all the rights as a Committee, they could also change any user's permission, being the only person capable of making another Admin or other Committees.

Other stakeholders:

Developers: The only people that are allowed to change the app's code.
General Public: People that will be able to download and install the app, but would be unable to use it without an University of Bristol account.
Google: Because of the use of Google Sign-In and Firebase, Google could provide maintenance and assistance, if needed.
Database Maintenance: The people that will be assigned to maintain the database.
Ad Companies: The companies whose advertisements would appear on the app.
Search Engines: The app would appear if someone uses a search engine to look for it. In addition, people could also search for the Wiki.
Investors: People or companies that may want to invest or fund the app.




System
Change permission
Create CSS Event
Create normal event
Comment on event
Mark event
Create profile
Edit information
Admin
Committee
User
creately
www.creately.com • Online Diagramming

Functional Requirements

The main requirements the application must fulfill are for students to view upcoming CSS events, as well as planning and sharing their own events. Additionally, users must be able to add course notes on a shared wiki.

Regarding the users, each member must be able to log in with their MyBristol credentials. To do this we will use Google's sign-in method, as the MyBristol login is based on gmail. Once logged in, they must be able to set up a profile which contains their info, a profile picture and the tags for events they are interested in. These details will then be saved on a back-end database created with SQL to store the information of each user.

The application must have different permissions for members depending on their position within the society. The three different levels are Admin, Committee and Student. Students can create and comment events. Committee members, in addition to student permissions, will be allowed to use "CSS" and "CSS Planning" tags. Admins, in addition to committee permissions, will be able to change other user's permission level. This is done by configuring user's permissions based on their email.

Regarding the events, they must be able to be added, edited and viewed from the application. Events must show details including their title, an image, a description, a date, time and duration, a location, tags visibility and which users are going. For attendance, users must be able to indicate if they are going to or are interested in the event. Furthermore, users must be able to comment on events and give specific dietary requirements. All the information regarding events will be stored in tables in the database. The purpose of the tags are to filter events. There must also be a "CSS" and "CSS planning" tag which only the CSS committee should be able to add. The users must be able to filter the events by the tags they are interested in. Events tagged with the "CSS" tag must be visible by all users and must appear before all other events or on a separate tab. Only the CSS committee members should be able to view events tagged with the CSS planning tag.

Regarding the unit wiki, there must be different sections for each unit as well as a separation between each academic years. All of the users must be able to add and edit notes in the wiki. Again, the wiki will be stored in an SQL database.

The application must be usable at least on Android devices, this requirement is achieved by using Android Studio to build the application.

Further, optional requirements include custom sign-ups and data entries for event, such as creation of teams and joint sign-ups. Badges and achievements for users, a friends system with messaging, statistics, integration with google calendar, a dark theme, and a pizza equation which predicts the amount of pizza needed to be bought for an event.

The preferred assets to be used are the CSS logo, Bebas Neue font for titles and Open Sans font for text.

Many of the requirements are flexible.

1. Login

- 1.1. Display Google Sign-in.
- 1.2. Allow only University of Bristol accounts (@my.bristol.ac.uk).
- 1.3. Verify credentials through Google Sign-in.
- 1.4. Proceed to next activity if login is successful .
    - 1.4.1. Profile creation if new user. (**4**)
    - 1.4.2. Homepage if existing user. (**3**)
- 1.5. Omit login if user is already signed-in.

2. Profile creation

- 2.1. Allow users to select a profile picture, name and academic year.
    - 2.1.1. Set as mandatory fields.
- 2.2. Allow users to select interested tags.
- 2.3. Create a profile with the information given when clicking “Create” button.
- 2.4. Proceed to Home activity upon profile creation. (**3**)
- 2.5. Omit profile creation if user already has a profile.

3. Homepage

- 3.1. Display created events.
    - 3.1.1. Display name, description and author of event.
    - 3.1.2. Direct to event view when clicked. (**8**)
- 3.2. Display search bar.
    - 3.2.1. Return matching results if any are found.
- 3.3. Display filter button.
    - 3.3.1. Remove all filters.
    - 3.3.2. Filter by tags.
    - 3.3.3. Filter by ‘CSS’ tag.
- 3.4. Include toolbar. (**13**)

4. Profile view

- 4.1. Display member’s profile picture, name, academic year and interests.
- 4.2. Display events created by the member.
    - 4.2.1. Direct to event display when clicked. (**8**)
- 4.3. If the profile is the user’s, display button to edit profile.
    - 4.3.1. Direct to edit profile.
- 4.5. If user is an admin, display button to change permissions.
- 4.6. Include toolbar. (**13**)

5. Profile edit

- 5.1. Allow users to edit their profile picture, name, academic year and interests.
    - 5.1 Display current information in each field.
- 5.2. Edit user profile with the new provided information after clicking “Save” button.

6. Events

- 6.1. Include button for creating a new event. (**7**)
- 6.2. Display events created by the user.
    - 6.2.1. Direct to the event when clicked. (**8**)

6.3. Display events marked by the user.
    6.3.1. Direct to the event when clicked. (**8**)
6.4. Include toolbar. (**13**)

7. Event creation

7.1. Allow user to enter picture, name, date, time, location and description.
    7.1.1. Set fields as mandatory.
7.2. Allow users to select tags for the event.
7.3. Create an event with the information given when clicking “Create Event” button.
7.4. Return to homepage upon event creation. (**3**)

8. Event view

8.1. Display name, picture, date, time, location, description and tags of event.
8.2. Display number of users going to the event.
8.3. Allow user to mark the event.
8.4. Allow user to comment on the event.
8.5. Display comments for the event.
    8.5.1. Display author, body, time and date of comment.
    8.5.2. If the user is the author of a comment, allow them to delete it.
8.6. If the user is the creator of the event, include button to edit the event. (**9**)
8.7. If the user is the creator of the event, include button to delete the event.

9. Event edit

9.1. Allow edit of event picture, name, date, time, location, tags and description.
    9.1.1. Display current information for each field.
9.2. Edit event with the new information after clicking “Save” button.

10. Calendar

10.1. Display a calendar with the current day selected by default.
10.2. When selecting a day, show events for that day.
    10.2.1. Direct to event view when clicking on an event. (**8**)
10.3. Include toolbar. (**13**)

11. Members

11.1. Display list of members.
    11.1.1. Direct to profile view when clicking on a member. (**4**)
11.2. Include search bar.
    11.2.1. Return matching results if any are found.
11.3. Include toolbar. (**13**)

12. Permissions

Level	Change permissions	Use "CSS" tags	Add comments	Create events
Admin	✔	✔	✔	✔
Committee	✗	✔	✔	✔
Student	✗	✗	✔	✔

13. Toolbar

- 13.1. Display current activity title.
- 13.2. Include navigation drawer.
  - 13.2.1. Navigation to home activity. (**3**)
  - 13.2.2. Navigation to user profile activity. (**4**)
  - 13.2.3. Navigation to events activity. (**6**)
  - 13.2.4. Navigation to calendar activity. (**10**)
  - 13.2.5. Navigation to members activity. (**11**)
  - 13.2.6. Navigation to settings activity. (**12**)

14. App settings

- 14.1. Allow to switch between light and dark theme.
- 14.2. Allow to switch between public or private profile.
- 14.3. Include button to delete user profile.
- 14.4. Include button to log out.
- 14.5. Include toolbar. (**13**)

Use case: planning an event. Sequence of steps :

1. User 1 logs in to the app through Google Sign-in.
2. User 1 is presented with the profile creation screen and creates a profile.
   - 2.1. User 1 enters picture, name, academic year and interests.
   - 2.2. User 1’s profile is saved in the database.
3. User 1 navigates to “Events” through the toolbar’s navigation drawer.
4. User 1 creates an event.
   - 4.1. User 1 enters picture, title, date, time, location, tags, and description for event.
   - 4.1. Event is saved in the database.
5. User 2 logs in to the app through Google Sign-in.
6. User 2 is presented with the profile creation screen and creates a profile.
   - 6.1. User 1 enters picture, name, academic year and interests.
   - 6.2. User 2’s profile is saved in the database.
7. User 2 navigates to “Calendar” through the toolbar’s navigation drawer.
8. User 2 clicks on the day set for User 1’s event.
9. User 2 is presented with all the events happening that day, including User 1’s event.
10. User 2 clicks on User 1’s event.
    - 10.1. User 2 is directed to the event.
11. User 2 marks User 1’s event.
    - 11.1 The number of users going to the event is updated in the database.
12. User 2 logs out.
13. User 1 selects the previously created event.
14. User 1 sees User 2 is going to the event.
15. User 1 logs out.
16. User 1 and User 2 happily enjoy the event together at the date, time and location set.

Alternative Flows

1. Users already have a profile created.

When signing in, users will not be required to create a profile again. In this case, steps 2 and 6 (Profile creation) would be skipped for each user respectively.

2. Users already have an open session on their device.

If this is the case, users will be presented with the homepage right away. Here, steps 1 and 2 for User 1 are skipped and steps 5 and 6 are skipped for User 2.

3. User 2 sees User 1's event on the homepage.

When User 1 creates the event, it is shown for all users on the homepage. Through here, User 2 can access the event in the same way, skipping steps 7 to 9.

Exceptional Flows

1.User 1 deletes their profile before User 2 can see the event.

When deleting one's profile, all events created by that user will be eliminated alongside the profile. In this scenario, User 2 will never be able to see User 1's event since it has been deleted from the database.

2.User 1 loses connection while creating an event.

In this scenario, User 1 will still be able to create the event and it will display for the author in the homepage. However, User 2 will not be able to see the event until User 1 regains connection to the database.

Non-Functional Requirements

To ensure an adequate product quality for the application, it must meet a set of non-functional requirements, presented below.

1. Design

- 1.1. Achieve Google standards by referring to material.io guidelines.
- 1.2. Express UoB CS Society brand through the use of preferred assets.
- 1.3. Navigation drawer for easy use.

2. Optimization

- 2.1. Less than 200ms response time for actions.
- 2.2. Less than 1s response time between activities.

3. Security

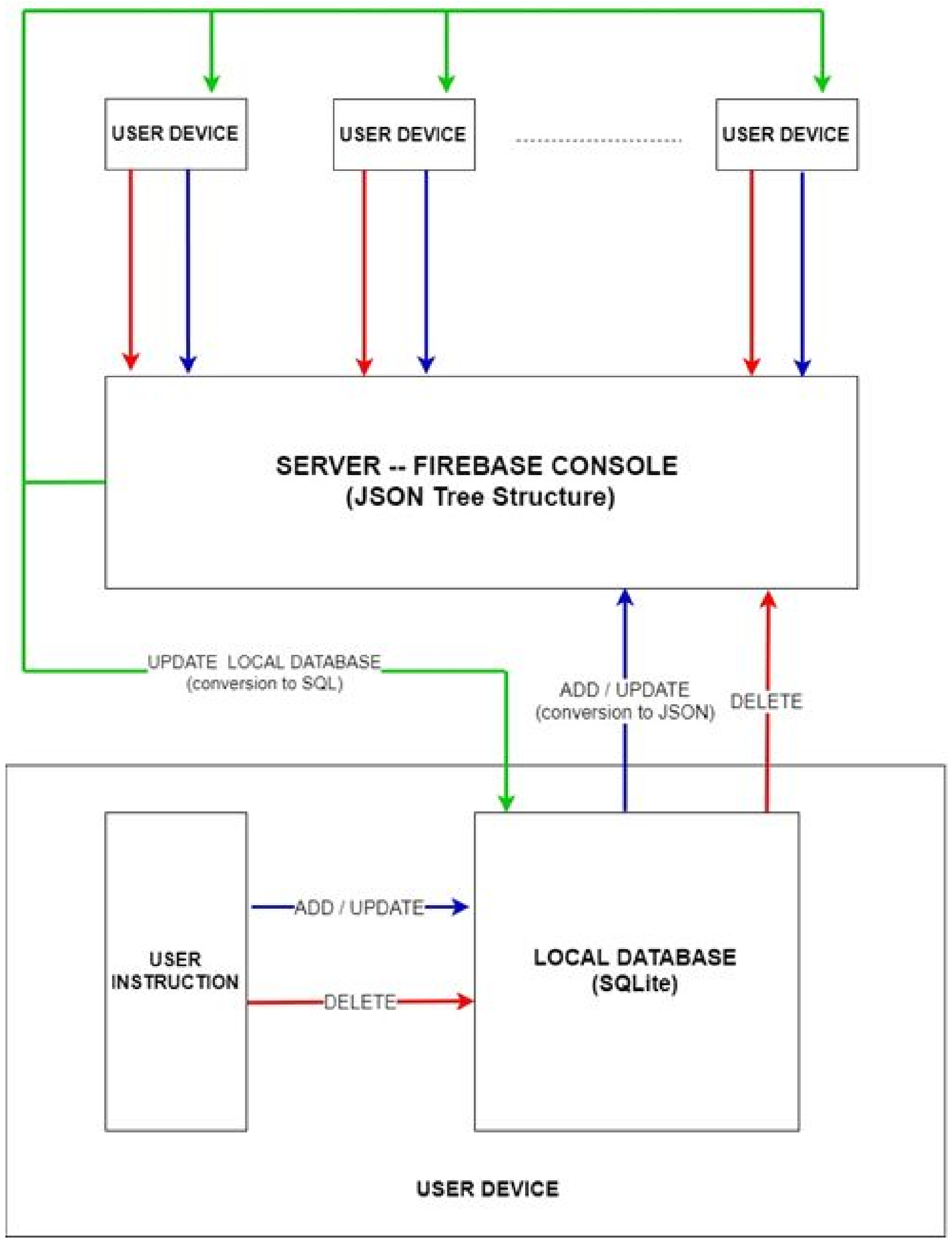
- 3.1. Google standards of security for user credentials through Google Sign-In.
- 3.2. Protection against SQL injection with prepared statements.

4. RAS

- 4.1. Reliability: the database must output the correct information within 3s.
- 4.2. Availability: the service must not exceed 50 hours of downtime per year.
- 4.3. Serviceability: if the service malfunctions, it must be repaired within 8 hours.

OO Design & UML

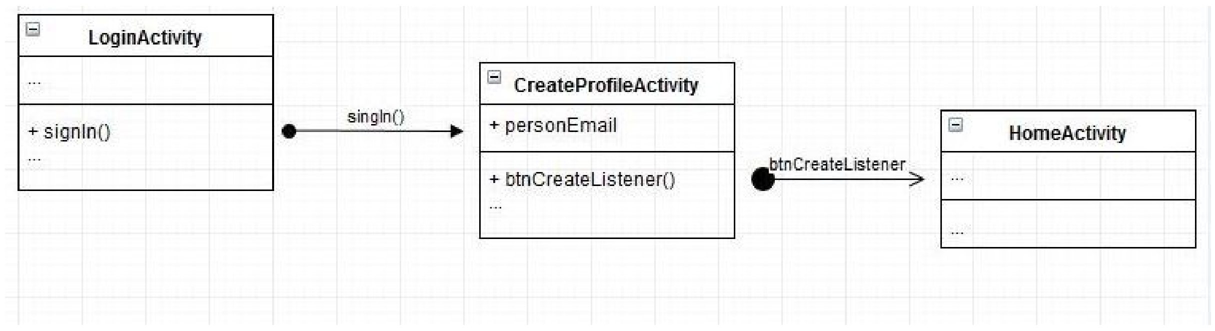
High-level architecture diagram


USER DEVICE
USER DEVICE
USER DEVICE
SERVER -- FIREBASE CONSOLE (JSON Tree Structure)
UPDATE LOCAL DATABASE (conversion to SQL)
ADD / UPDATE (conversion to JSON)
DELETE
USER INSTRUCTION
ADD / UPDATE
DELETE
LOCAL DATABASE (SQLite)
USER DEVICE

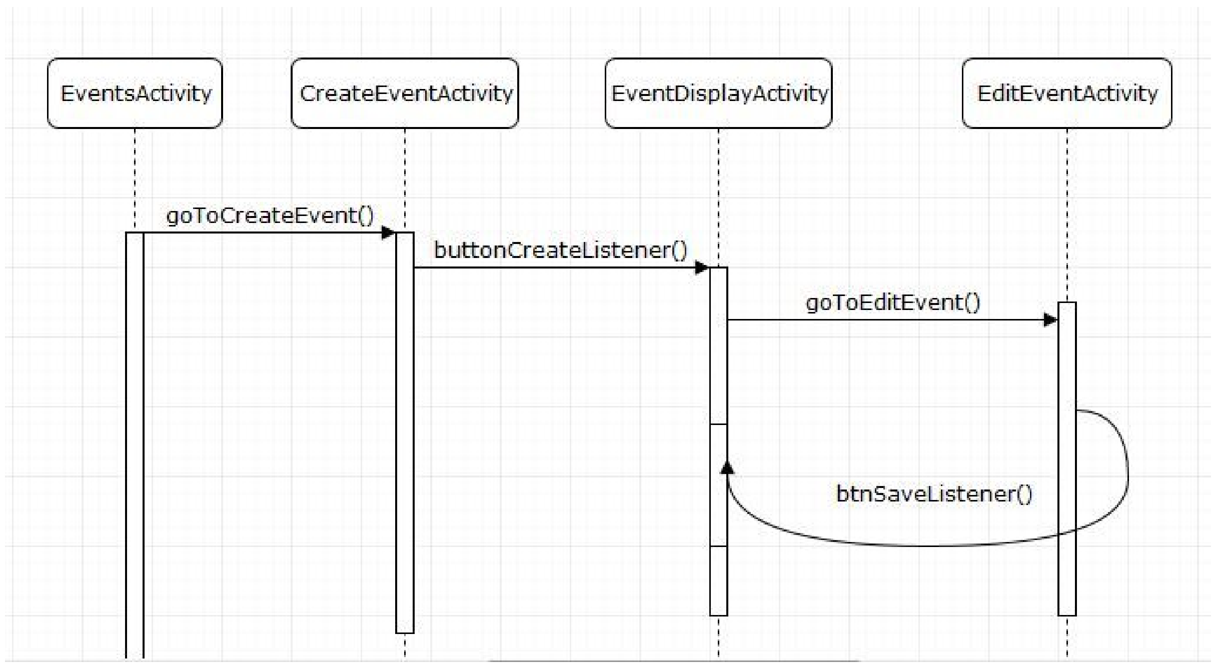
Components

The diagram shows the workflow of the backend processes. On one hand, there is a local database, created in each of the user's devices. It receives commands from the user and, at the same time, is listens for update instructions from the server. On the other hand, the server receives constant data from all users, converts the input into JSON object format and stores it in a tree structure.

Static UML modeling aspect



Dynamic UML modeling aspect



Model Context

The first challenge we faced when we began working on the back-end architecture was providing the functionality required by our client and, at the same time, allowing ourselves to develop clear code so we can tackle any issues we might face with ease. We tried to limit ourselves to pre-existent technologies that were, to some extent or another, either developed for use in an Android environment or frequently used in this particular context. In order for all data to be stored in one place, so that the app may function appropriately, an online form of hosting was needed.

The use of this is storing all the data needed to run the app. This includes information about the users, i.e. name, surname, academic year, interests, etc. Furthermore, the back-end stows details about created events, comments and lists of participants for each of the events. This collection of information needs to be malleable, so data may be added, viewed, removed or updated whenever.

Modeling Choice

Multiple attempts were undertaken at this stage, in order to find the best approach, some of which implied the use of either Oracle Cloud, MySQL or Google Firebase. During a few meetings with our mentor, we discussed the advantages and disadvantages of using each and every one of our methods. In the end, we decided the best way for us to achieve our task would be to keep going by the aforementioned principle. That is, using technologies that are highly compatible with the Android environment. Therefore, we chose to use Google Firebase as our means of online hosting.

Another advantage regarding this particular method is the sheer amount of provided documentation, both on the official website as well as in online forums. We also needed a means of storing the data locally, in order to assure easiness in fetching the appropriate information and the ability to view personal information even in an offline state. We decided to use the relational database management system called SQLite.

There are two primary reasons why we made this particular choice. One of them is that the SQL language is a familiar one, with a clear and straightforward syntax. The other one is that SQLite, particularly, is very frequently used in an Android environment, which would prove to be of great help when it comes to available online resources.

SQLite would allow us to construct a database which would be stored locally in each user's device. This database would constantly listen to updates from the server and would update itself accordingly. Whenever a change was made locally, that change would automatically update the server as well, creating a constant sync between the online hosting environment and all devices which use the app.

In order to do the sync, the data from the relational database would be reconfigured into JSON objects, subsequently distributed in the Firebase Database and arranged into tree structures. Any byte maps from the local environment, usually used for storing pictures, would be encoded into string formats, using the renowned Base64 algorithm.

What we learned

Probably the biggest learning outcome was that, when we first began working on this project, we had not considered how important and challenging the back-end architecture implementation could be, and how tough it would be to make a modeling choice. Another very problematic issue is that small errors and/or mistakes can create a domino effect in our model, essentially resulting in more voluminous issues when it comes to the functionality of our product.

Development Testing

Testing Strategy

The first stages of the development for our app was producing XML files and designing the layouts for each of our separate activities. Once we had finished creating them, we would write unit tests that verify each component works as intended as well as check the layout is consistent and stable. For example a test to make sure a text box works as intended would consist of finding the textbox view using a view matcher, performing an action (e.g. typing hello) and then asserting that the textbox displays the text “hello”. Each of the individual pages within the client component of our UML architecture model is an activity class within our project. For each activity we hve a corresponding testing class that was concerned with only testing the functionality of that activity.This meant we could keep our tests contained so when conducting integration tests each of the tests would not interfere with one another. An integration test would be conducted when a group of components in our activity and layout had been fully implemented. The writing of a test consisted of simulating use of those components and asserting the final outcome is as expected.

We tested the database by storing in the values and information from the “Create Profile” activity. For each value entered, we confirmed it properly enters the database in the appropriate section by looking into the database after introducing the value through the application. This was unique for each type of input field. Once the data were safely introduced into the database, we used the “Profile” activity to check it could all be obtained from the database and represented in the application. We have also ran our app through Firebase’s own “Test Lab” feature, which has emulated the use of our app on a series of different virtual devices. During the process, the “Robo test” undertook around 100 actions per activity and, at the end, all tests were marked as “passed”.

Testing frameworks

The main testing framework we are going to use is JUnit. We used the espresso framework as a way of implementing tests for our app. Espresso is a testing framework provided by Android Studio, and it provides us a reliable way of testing the apps front-end UI .To aid us in understanding the workflow, we designed a wireframe with pen and paper which included all the activities we had planned to implement, making it easier to formulate an idea of how the front-end of the application should work and how we should write the tests. These were used when writing Unit and Integration tests. Our testing strategy was to write tests along with the backend and frontend writing. As soon as we've implemented a function, we treated it as a black box, wrote a functional test for it and verified the function is working correctly by checking its output.

Challenges

One of the problems we encountered while testing was testing of some of the components such as ListViews. When checking if the ListView has contained the proper data, we were incapable of matching the data in the list for the adapter to its specific corresponding ChildView in the ListView. To get around this we had to write a test that manually scrolled through the ListView and matched the text of the data to anything within the View’s hierarchy. Whilst this achieves a similar goal, it is a far less elegant solution for testing the ListView. When testing the database and a problem arose, debugging our code remained a long process. It was hard to locate the problem, and we had to use logs on every function to test which ones were working as intended and which ones weren't. From there we could deduce the problem.

Product Evaluation

We presented our app to CSS members and to Computer Science students and we had them use it, in order to get feedback on our project. We believed that having them use the app for the first time, without any explanations or tutorials, would offer us the best insights and would increase the possibility of finding hidden bugs. We documented every observation by writing notes.

The overall response was positive: most users believed the app was well designed and it had all the required functionality to be useful. However, many argued that the switch to a different platform for events could be undesirable when it can already be done through Facebook.

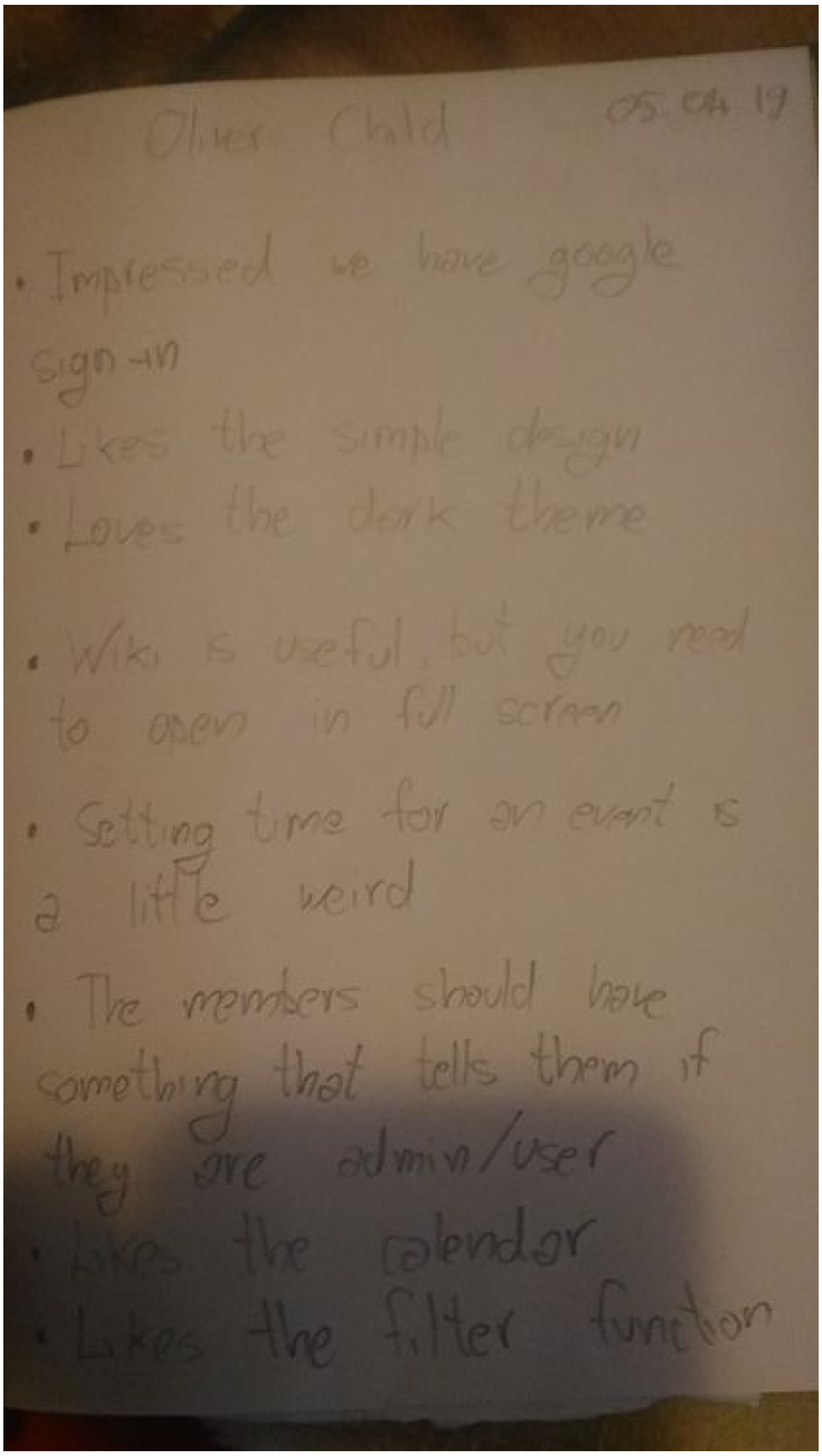
There were several aspects of the app which users were especially pleased with, such as the option of having a dark theme. Additionally, users were happy with the inclusion of the Wiki, and thought it could be a useful feature for CSS students.
On the other hand, we gathered observations towards other aspects of the app which users would like to be different. These are important to consider for future updates:

- The use of Google Sign-In instead of a University of Bristol SSO proved contradictory to some, as the log-in page requires users to enter their university credentials.
- Some users had trouble creating their profile or events due to not explicitly representing the images for these activities as mandatory.
- When presented with the Home screen, some users felt lost, as there are no guidelines as to what to do next, and the other activities are accessed through a drawer that is hidden unless opened. This was especially true when the Home page was empty.
- When choosing the academic year as “Postgraduate”, the profile page would instead display the academic year as “Year 5”.
- The “Date” field under events in Calendar is too far to the right.

The app was also presented to some committee members and the client, to confirm the validity of our app. It received approval from both sectors and they considered that the requirements were satisfied. Some of the most notable fixes suggested by the committee that were applied were:

- Having the start time and end time in of an event in separate fields, so that they can be changed individually
- Setting the Wiki page to open in desktop view, because users can't modify it in mobile view
- When writing something in the app, the first letter was always lowercase, which troubled a lot of the users. We added an attribute, so that the first letter is always uppercase

Notes from test session with Oliver Child, CSS committee member, in MVB, 05.04.2019:



Oliver Child 05.04.19

- Impressed we have google sign-in
- Likes the simple design
- Loves the dark theme
- Wiki is useful, but you need to open in full screen
- Setting time for an event is a little weird
- The members should have something that tells them if they are admin/user
- Likes the calendar
- Likes the filter function