

COMP 2000 – Software Engineering 2

Introduction.....	1
Background.....	1
Legal, Social, Ethical and Professional.....	2
Design	2
Implementation.....	5
User Testing.....	12
Evaluation	15
References	15

Introduction

This report will show the design process of the android application, that manages a series of employees by connecting to a RESTful API to access the data stored about the employees. The high-fidelity prototype used to design the interfaces and functionality of this application, the implementation of this program based on the design and the summative user testing study completed to evaluate the functionality of the system. The user testing plan will show the process of getting participants and their experiences of using the application, as well as the conclusions gained from this evaluation.

A video demonstrating the program can be accessed here:

<https://youtu.be/LTwTCxG3TH8>

The GitHub Repository holding the android application is found here:

<https://github.com/Plymouth-University/comp2000-main-assignment-Dan-Livermore.git>

The data used for this application can be found here:

<web.socem.plymouth.ac.uk/COMP2000/api/employees/index.html>

Background

The android app that was developed in Java is an employee management system, where the administrator logs into the system and can perform CRUD operations on their employees, whose data is accessed using a REST API. The employees can also access this system, allowing them to access and update their personal data, in addition to claiming days off from the company. Both user types of the system, log in by entering an email and a password, but the administrator can access and amend more data than the employee. The interfaces for this application will be based on the Google Pixel 4 android phone, so the screens will scale to any screen size and will run on any device that can use API 24 or above.

Legal, Social, Ethical and Professional

The data stored on the API must be very secure as otherwise there is a great number of threats to both the web system and the application that has been developed. Threats including human error, inconsistencies in the code, and cybersecurity breaches (Catherine Cote, 2021)ⁱ, could have a massive impact on the organizations due to the time and money taken to fix these issues – it now can cost up to \$10 million to fix a breach (IBM,2022)ⁱⁱ.

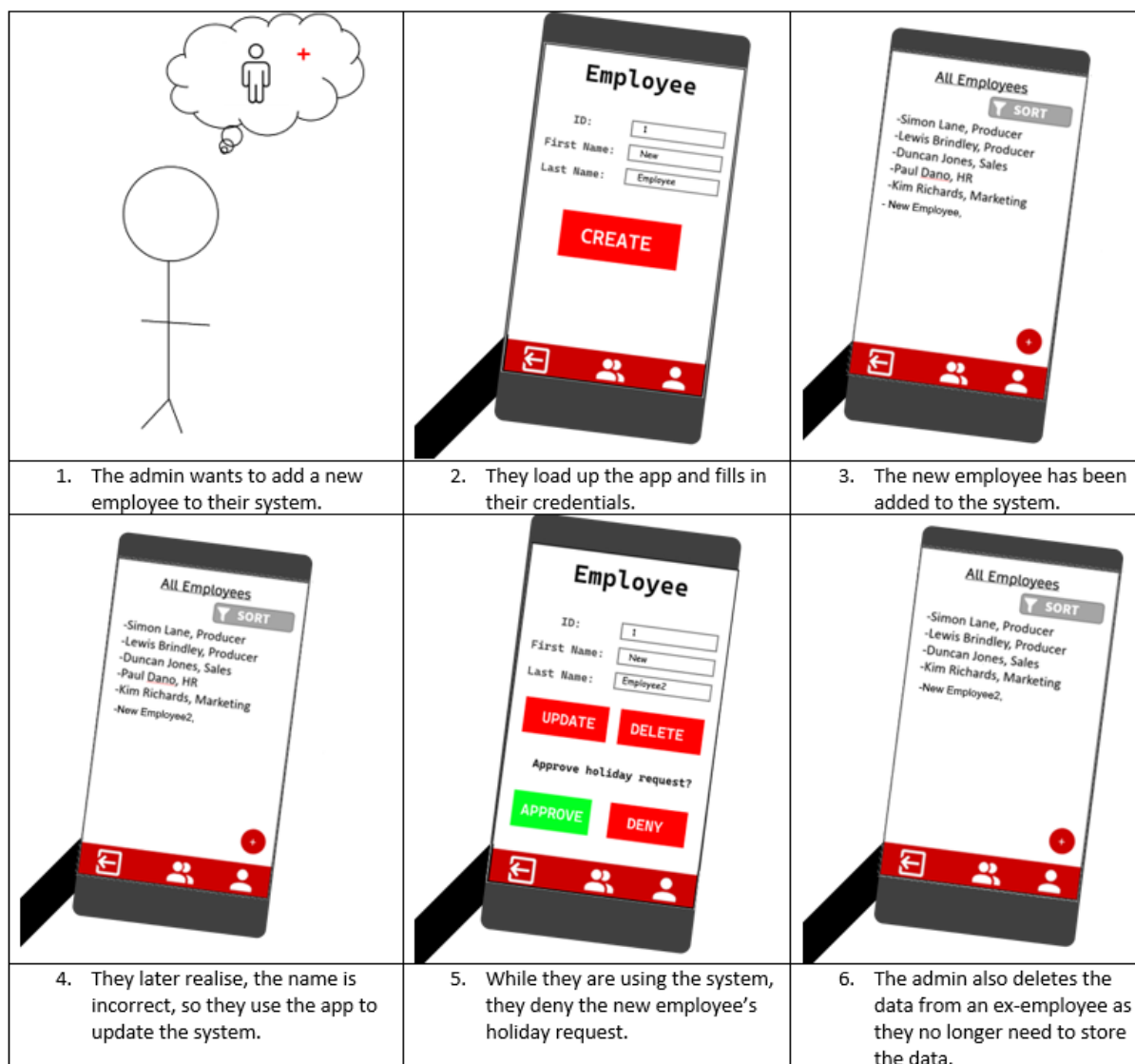
Additionally, the developers of the system should ensure that they are not misusing their access to personal data. The data should be considered confidential and unless given explicit consent,⁴ there is no time where the developers should view the personal data of any employee of their client. In any program developed, privacy should be a priority to protect every user of the system but in such an event when the system gets taken down, the developer would be breaking the GDPR due to compromising the personal data of others.

It is important that the third-party API used in this system is very secure before implementing, as once it has been breached, data from several other companies could be compromised, which is very expensive and time-consuming to fix. This could be a result of lacklustre access control as 94%(OWASP, 2021)ⁱⁱⁱ of applications have an access control weakness.

Another constraint of the API used to create this service is that the developers must all share the same space to save their users. This could lead to data being compromised or changed by other users when testing their program. If the API remained in this state during the release of the program, it is highly likely that the data of the employees could be overwritten easily as there is no security in place to protect the data that is being stored.

Design

Below is a storyboard used to explain the functionality of the admin side of this application. Where an admin can manage and maintain the information about a company's employees on an android app.



Following this, an improved high-fidelity prototype was created to demonstrate how the screens of the program would look as well as what functionality is required on each one of them. The screens are designed to conform to the same resolution as the Google Pixel 4 mobile device, which will be used to implement the program.

The interfaces are based on what had been previously developed but with some changes to focus on connecting to the API system. As they were a good base to work from, it was decided to just update them to fit the requirements; the changes are described below.

These diagrams also show which icons will be used; any images used for the buttons in this design were used from Google's Material Design Icons as these will be similar to the ones used in the development environment. This also makes sure that all the interfaces create a cohesive system, and that any user experience changes could be completed before implementing the design.

<div data-bbox="55 78 406 645"> <div>Employee Log In</div> <div>ADMIN LOG IN</div> <div> <div>Username :</div> <div></div> </div> <div> <div>Password :</div> <div></div> </div> <div>LOG IN</div> </div>	<div data-bbox="448 78 786 286"> <p>This screen is the same as initially planned for during the low-fidelity prototyping stage. As it is a simple log in screen, nothing needed to be improved.</p> </div>	<div data-bbox="826 78 1177 645"> <div>Admin Details</div> <div> <div>Name:</div> <div>Josh Turpin</div> </div> <div> <div>DOB:</div> <div>23/08/1969</div> </div> <div> <div>Email:</div> <div>jdturps@yahoo.com</div> </div> <div> <div>Address:</div> <div>25 Our Street London, LA2 4XP</div> </div> <div> <div>Position:</div> <div>CCO</div> </div> <div> <div>Start Date:</div> <div>12/11/2010</div> </div> <div>NOTIFICATION SETTINGS</div> <div>Last Updated: 05/11/22</div> </div>	<div data-bbox="1219 78 1557 497"> <p>This screen has the updated navbar on it. The employees and holiday claims have been consolidated into one section to streamline the user experience and remove clutter from the navbar. The user will use this screen to edit their notifications and to view their data.</p> </div>
<div data-bbox="55 667 406 1238"> <div>Admin Details</div> <div>Notification Settings</div> <div> <div>- Device Notifications</div> <div></div> </div> <div> <div>- Notify when user updates data</div> <div></div> </div> <div> <div>- Notify when holiday is claimed</div> <div></div> </div> <div>SAVE</div> <div>Last Updated: 05/11/22</div> </div>	<div data-bbox="448 667 786 907"> <p>This screen has only one change from the initial plan, the updated navbar. The user will use this screen to choose their notification preferences for the rest of the application.</p> </div>	<div data-bbox="826 667 1177 1238"> <div>All Employees</div> <div>SORT</div> <div> <div>-Simon Lane, Producer</div> <div>-Lewis Brindley, Producer</div> <div>-Duncan Jones, Sales</div> <div>-Paul Dano, HR</div> <div>-Kim Richards, Marketing</div> </div> <div></div> </div>	<div data-bbox="1219 667 1557 945"> <p>A button has been added at the bottom of this window to allow the admin to quickly add new employees to the database. The list of employees will be retrieved using the API.</p> </div>
<div data-bbox="55 1261 406 1854"> <div>Employee</div> <div> <div>ID:</div> <div></div> </div> <div> <div>First Name:</div> <div></div> </div> <div> <div>Last Name:</div> <div></div> </div> <div>CREATE</div> </div>	<div data-bbox="448 1261 786 1639"> <p>This screen allows the admin to create a new employee. By entering the employee's name and a random integer, the employee can be added to the system. When pressing the create button, a toast will appear telling the admin that the data has been stored.</p> </div>	<div data-bbox="826 1261 1177 1854"> <div>Employee</div> <div> <div>ID:</div> <div></div> </div> <div> <div>First Name:</div> <div></div> </div> <div> <div>Last Name:</div> <div></div> </div> <div>UPDATEDELETE</div> <div>Approve holiday request?</div> <div>APPROVEDENY</div> </div>	<div data-bbox="1219 1261 1557 1568"> <p>This screen will let the admin update, delete, and authorize holiday claims. As long as the ID is correct both of these actions will be completed. The holiday requests could inform the employee in a later version of the application.</p> </div>

Implementation

The interface designs as shown previously were implemented into the application largely as planned. A few unrequired elements were removed but the how the user interacts with the windows remains unchanged. Fonts and button sizes may have changed but this was due to the built-in fonts in Android Studio rather than a deliberate choice.

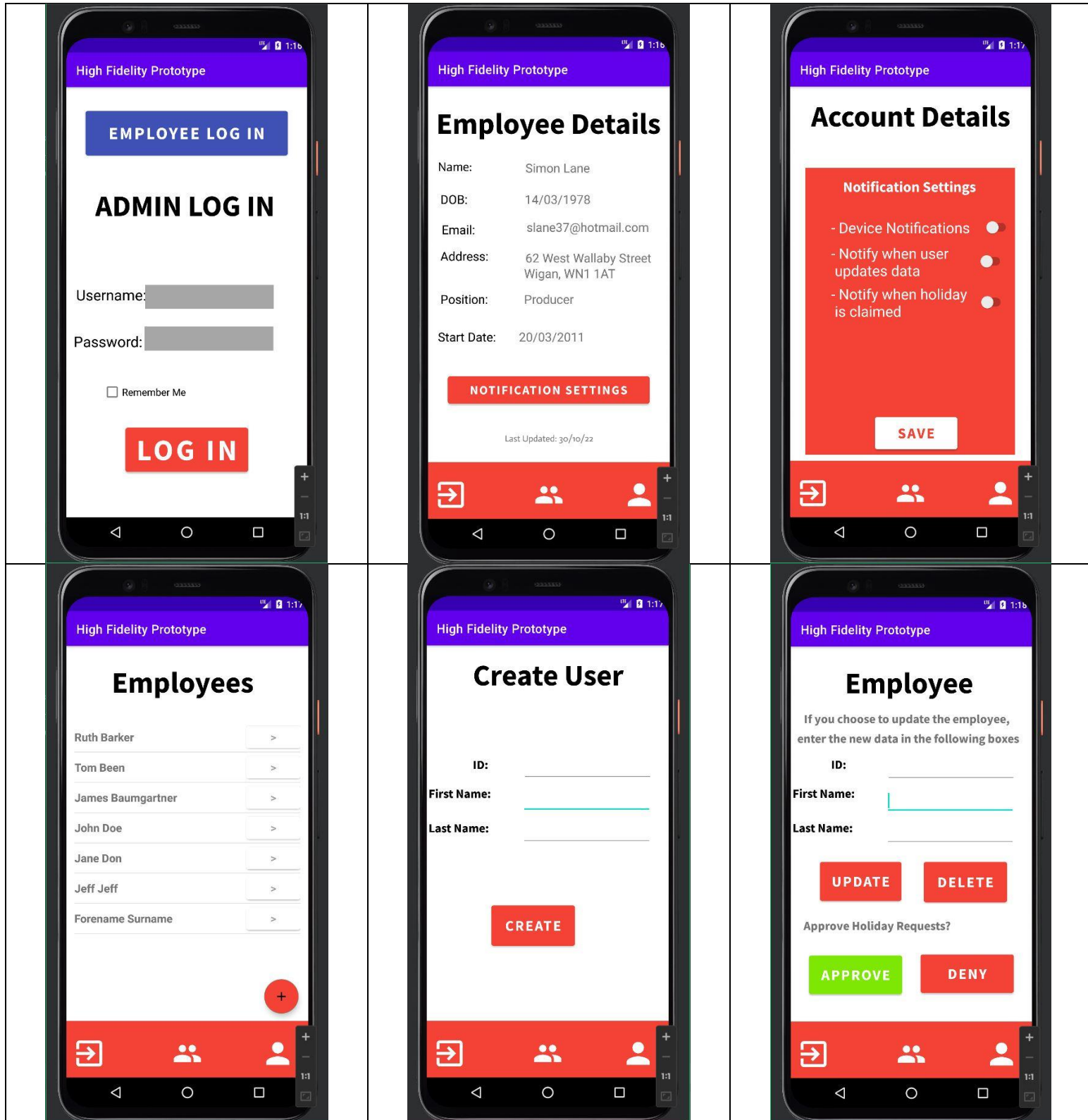


Figure 1 – Implemented interfaces

The application connects to the REST API by using the volley library and transferring data to the web system via JSON requests. To complete the CRUD functionality on the database (Create, Read, Update, Delete), a JSON array is sent including the employee's data, the URL of the webpage and the type of request intended - POST, GET, PUT and DELETE for the create, read, update and delete operations.

As shown in the design, the data stored in the database must be displayed in a list, allowing the admin to select the employee from the list of all employees. For the get requests, the data is returned in a list of JSON objects. The data is iterated through, formatted correctly and then displayed into the list. From there a button is added allowing the admin to edit the employee's data. If an error occurs when reading the data, a toast will tell the user that something goes wrong, but a more sophisticated could not be implemented due to time restraints. I had hoped to add additional try-catch statements to resolve the common errors that occur when trying to connect to a network (400, 404, 405) but this could not be completed during this time.

```
class GetEmployeesData extends AsyncTask<Void, Void, Void> {
    ▲ Dan-Livmore
    @Override
    protected Void doInBackground(Void... voids) {
        RequestQueue queue = Volley.newRequestQueue( context: AdminEmployeeList.this);
        //gets URL of webpage that contains the data
        String url = "http://web.socem.plymouth.ac.uk/COMP2000/api/employees/";
        //starts request for all the data
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            //GET = read
            ▲ Dan-Livmore
            new Response.Listener<String>() {
                ▲ Dan-Livmore
                @Override
                public void onResponse(String response) {
                    // Parse the response string and display the data in a list view
                    try {
                        //creates a JSON array in the form as the JSON data on the API
                        JSONArray jsonArray = new JSONArray(response);
                        ArrayList<DataModel> data = new ArrayList<>();
                        //iterates through all the data and adds it to the list
                        for (int i = 0; i < jsonArray.length(); i++) {
                            JSONObject jsonObject = jsonArray.getJSONObject(i);
                            DataModel dataModel = new DataModel(jsonObject);
                            data.add(dataModel);
                        }
                        //adds the list onto the list view to display the information
                        ListView listView = findViewById(R.id.list_view);
                        CustomAdapter adapter = new CustomAdapter( context: AdminEmployeeList.this, data);
                        listView.setAdapter(adapter);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
                ▲ Dan-Livmore *
            }, new Response.ErrorListener() {
                ▲ Dan-Livmore *
            }
        );
        @Override
        public void onErrorResponse(VolleyError error) {
            //basic error testing
            Toast.makeText( context: AdminEmployeeList.this, text: "Could not find employees", Toast.LENGTH_LONG).show();
            //could improve to handle network errors (400 / 404 / 405)
        }
    }
}

//calls worker thread to complete the async HTTP request
new GetEmployeesData().execute();
}
```

Figure 2 - Read request

The other JSON requests for inserting, updating, and deleting data are quite similar, only different based on their required inputs and how their outputs are handled. Shown below is the implementation of the update command, where after starting a new worker thread for the API requests the user-entered data is converted into a JSON object so it can be stored in the allocated location on the server. Following this, the URL is updated to lead directly to the object being updated, and the JSON object request gets completed.

After this, a notification is produced if notifications have been turned on and notifications for updating data have been selected. If one of these parameters is not met, no notification is displayed, leaving the user unsure of whether the data has been stored without checking the URL itself.

```

updatebtn.setOnClickListener(view -> {
    // Dan-Livemore
    class UpdateEmployeesData extends AsyncTask<Void, Void, Void> {

        @Override
        protected Void doInBackground(Void... voids) {
            RequestQueue queue = Volley.newRequestQueue(context: AdminEmployeeData.this);

            //creates input items
            EditText addID = findViewById(R.id.enterID2);
            TextView addFname = findViewById(R.id.EnterFname2);
            TextView addLname = findViewById(R.id.enterLname1);

            //get input data
            int newID = Integer.parseInt(addID.getText().toString());
            String newFname = addFname.getText().toString();
            String newLname = addLname.getText().toString();

            //creates input JSON
            JSONObject object = new JSONObject();
            try {
                object.put( name: "id", newID);
                object.put( name: "surname", newLname);
                object.put( name: "forename", newFname);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }

            //updates the URL to the specific user being updated
            String ID = addID.getText().toString();
            String url = "http://web.socem.plymouth.ac.uk/COMP2000/api/employees/" + ID;
            JSONObjectRequest jsonObjectRequest = new JSONObjectRequest(Request.Method.PUT, url, object,
            //PUT = update

            new Response.Listener<JSONObject>() {

                @Override
                public void onResponse(JSONObject response) {
                    Toast.makeText(context: AdminEmployeeData.this, text: "Updated Employee", Toast.LENGTH_SHORT).show();
                }

            }, new Response.ErrorListener() {

                @Override
                public void onErrorResponse(VolleyError error) {
                    Toast.makeText(context: AdminEmployeeData.this, text: "incorrect ID", Toast.LENGTH_SHORT).show();

                    //could improve to handle network errors (400 / 404 / 405)
                }

            });

            //Adds the new HTTP request to queue
            queue.add(jsonObjectRequest);

            //If notifications have been turned on correctly, push to the user that the data has been changed
            if (Notifications.devicenotifications == Boolean.TRUE && Notifications.afterupdate == Boolean.TRUE) {
                NotificationCompat.Builder builder = new NotificationCompat.Builder(context: AdminEmployeeData.this, channelId: "Notification 6");
                builder.setContentTitle("Employee Update");
                builder.setContentText("Employee Updated");
                builder.setSmallIcon(R.drawable.ic_baseline_adb_24);
                builder.setAutoCancel(true);

                NotificationManagerCompat managerCompat = NotificationManagerCompat.from(context: AdminEmployeeData.this);
                managerCompat.notify(id: 1, builder.build());

                return null;
            }

        }

        //calls worker thread to complete the async HTTP request
        new UpdateEmployeesData().execute();
    }
});

```

Figure 3 – Update request

Below is the code used for inserting and deleting data, but as they are similar to the other code previously shown, they will not have any explanations.

Insert

```
//when button is pressed create the queue for the API requests
createbtn.setOnClickListener(view -> {
    //starts worker thread
    class InsertEmployeesData extends AsyncTask<Void, Void, Void> {
        @Override
        protected Void doInBackground(Void... voids) {
            RequestQueue queue = Volley.newRequestQueue( context: AdminCreate.this);
            String url = "http://web.socem.plymouth.ac.uk/COMP2000/api/employees";

            //creates a new JSON object to store the data of the new employee in
            JSONObject object = new JSONObject();
            EditText addFname = findViewById(R.id.EnterFname2);
            EditText addLname = findViewById(R.id.enterLname1);
            EditText addID = findViewById(R.id.enterID2);

            //gets the data from the input boxes
            String newFname = addFname.getText().toString();
            String newLname = addLname.getText().toString();
            int newID = Integer.parseInt(addID.getText().toString());
            //stores the input data into the JSON object
            try {
                object.put( name: "id", newID);
                object.put( name: "forename", newFname);
                object.put( name: "surname", newLname);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }

            JSONObjectRequest jsonObjectRequest = new JSONObjectRequest
            //POST = insert
            (Request.Method.POST, url, object, new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                }
            }, new Response.ErrorListener() {
                2 usages
                @Override
                public void onErrorResponse(VolleyError error) {
                    //basic error testing
                    Toast.makeText( context: AdminCreate.this, text: "Could not create new employee, ID already used", Toast.LENGTH_LONG).show();
                    //could improve to handle network errors (400 / 404 / 405)
                }
            });

            //adds the insert to the request queue
            queue.add(jsonObjectRequest);
            //swaps screen back to employee list to show the user that the employee has been added
            Intent intent = new Intent( packageContext: AdminCreate.this, AdminEmployeeList.class);
            startActivity(intent);
            //if notifications have been correctly turned on, a push notification will be shown
            if (Notifications.deviceNotifications == Boolean.TRUE && Notifications.afterupdate == Boolean.TRUE) {
                NotificationCompat.Builder builder = new NotificationCompat.Builder( context: AdminCreate.this, channelId: "Notification 4");
                builder.setContentTitle("New Employee");
                builder.setContentText("Employee Added");
                builder.setSmallIcon(R.drawable.ic_baseline_adb_24);
                builder.setAutoCancel(true);

                NotificationManagerCompat managerCompat = NotificationManagerCompat.from( context: AdminCreate.this);
                managerCompat.notify( id: 1, builder.build());
            }
            return null;
        }
    }
    //calls worker thread to complete the async HTTP request
    new InsertEmployeesData().execute();
});
```

Figure 4 - Insert request

Delete

```
//create button to delete a user
Button deletebtn = findViewById(R.id.deletebutton);

deletebtn.setOnClickListener(view -> {

    class DeleteEmployeesData extends AsyncTask<Void, Void, Void> {

        @Override
        protected Void doInBackground(Void... voids) {
            //formats ID for use in URL
            EditText addID = findViewById(R.id.enterID2);
            String newID = addID.getText().toString();

            //starts queue for JSON requests
            RequestQueue queue = Volley.newRequestQueue( context: AdminEmployeeData.this);
            //changes URL to include ID for data being update
            String url ="http://web.socem.plymouth.ac.uk/COMP2000/api/employees/" + newID;
            JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
                (Request.Method.DELETE, url, jsonRequest: null, new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject response) {
                        Toast.makeText( context: AdminEmployeeData.this, text: "User Deleted", Toast.LENGTH_SHORT).show();
                    }
                }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        // TODO: Handle error
                        Toast.makeText( context: AdminEmployeeData.this, text: "ID not found", Toast.LENGTH_SHORT).show();
                    }
                })
            //could improve to handle network errors (408 / 404 / 405)
            {}
        };

        //adds request to queue
        queue.add(jsonObjectRequest);
        Intent intent = new Intent( packageContext: AdminEmployeeData.this, AdminEmployeeList.class);
        startActivity(intent);
        //if notifications are required, one is displayed
        if (Notifications.devicenotifications == Boolean.TRUE && Notifications.afterupdate == Boolean.TRUE) {
            NotificationCompat.Builder builder = new NotificationCompat.Builder( context: AdminEmployeeData.this, channelId: "Notification 5");
            builder.setContentTitle("Employee Update");
            builder.setContentText("Employee Deleted");
            builder.setSmallIcon(R.drawable.ic_baseline_adb_24);
            builder.setAutoCancel(true);

            NotificationManagerCompat managerCompat = NotificationManagerCompat.from( context: AdminEmployeeData.this);
            managerCompat.notify( id: 1, builder.build());
        }
        return null;
    }

    //calls worker thread to complete the async HTTP request
    new DeleteEmployeesData().execute();
});
```

Figure 5 - Delete request

The admin must also be allowed to authorize holiday requests from the employees. Unfortunately, I was not able to make a request from the employee side appear on the admin program, so the buttons used to approve or deny the request do not change any data, they only create a push notification if enabled.

```

//create holiday buttons
Button denybtn = findViewById(R.id.DenyButton);
Button approvebtn = findViewById(R.id.ApproveButton);

//if notifications are turned on, display to the user that the holiday has been confirmed or denied
denybtn.setOnClickListener(view -> {
    if (Notifications.devicenotifications == Boolean.TRUE && Notifications.isClaimed == Boolean.TRUE) {
        NotificationCompat.Builder builder = new NotificationCompat.Builder( context: AdminEmployeeData.this, channelId: "Notification 3");
        builder.setContentTitle("Holiday Claim");
        builder.setContentText("Holiday Claim Denied");
        builder.setSmallIcon(R.drawable.ic_baseline_adb_24);
        builder.setAutoCancel(true);

        NotificationManagerCompat managerCompat = NotificationManagerCompat.from( context: AdminEmployeeData.this);
        managerCompat.notify( id: 3, builder.build());
    }
});

approvebtn.setOnClickListener(view -> {
    if (Notifications.devicenotifications == Boolean.TRUE && Notifications.isClaimed == Boolean.TRUE) {
        NotificationCompat.Builder builder = new NotificationCompat.Builder( context: AdminEmployeeData.this, channelId: "Notification 4");
        builder.setContentTitle("Holiday Claim");
        builder.setContentText("Holiday Claim Approved");
        builder.setSmallIcon(R.drawable.ic_baseline_adb_24);
        builder.setAutoCancel(true);

        NotificationManagerCompat managerCompat = NotificationManagerCompat.from( context: AdminEmployeeData.this);
        managerCompat.notify( id: 4, builder.build());
    }
});

```

Figure 6 - Holiday claims

Push notifications were incorporated into this design to allow the user to customise their experience by choosing when they wish to receive notifications. I created a screen with switches that allowed the user to select whether they wanted notifications for the entire app, when an employee's data is updated or when a holiday claim is approved or denied. Once selected the outputs of the switches are given to a Booleans in a static class in a separate java file called 'Notifications', separating the data so it can be used but not edited throughout the program. The method used to update the variables could be improved as it requires the user to press the 'save' button before storing this data they could easily click off this without saving their changes. And the code could be improved if each method only updated one of the static variables (single-responsibility principle).

```

public class Notifications {
    //creates variables that can be accessed but not amended from anywhere in the program
    6 usages
    static Boolean devicenotifications;
    3 usages
    static Boolean isClaimed;
    4 usages
    static Boolean afterupdate;

    //updates the static classes based on the output of the AdminNotifications window
    1 usage 1 Dan-livemore
    public static void updateNotifications(Boolean x, Boolean y, Boolean z){
        if (x == Boolean.TRUE){
            devicenotifications = Boolean.TRUE;
        }
        if (y == Boolean.TRUE){
            isClaimed = Boolean.TRUE;
        }
        if (z == Boolean.TRUE){
            afterupdate = Boolean.TRUE;
        }
    }
}

```

Figure 7 - Notifications class

```

// initiate switches
Switch switch1 = (Switch) findViewById(R.id.switch4);
Switch switch2 = (Switch) findViewById(R.id.switch3);
Switch switch3 = (Switch) findViewById(R.id.switch2);

Button savebutton = findViewById(R.id.SaveNotify);
//save data button switches activity
savebutton.setOnClickListener(view ->{
    //after button is pressed, the values of the switches is stored and used to update the static class
    Boolean x = Boolean.FALSE;
    Boolean y = Boolean.FALSE;
    Boolean z = Boolean.FALSE;
    if (switch1.isChecked()){
        x = Boolean.TRUE;
    }
    if (switch2.isChecked()){
        y = Boolean.TRUE;
    }
    if (switch3.isChecked()){
        z = Boolean.TRUE;
    }
    Notifications.updateNotifications(x,y,z);
    //returns to previous screen
    Intent intent = new Intent( packageContext: AdminNotifications.this, AdminAccount.class);
    startActivity(intent);
});
}

```

Figure 8 - Update notifications procedure

User Testing

The participants are voluntary users that have a high level of technical ability, so they should have no issue using the prototype. The participants were selected for this reason as well as they were willing to take part in the testing. This will increase the speed of the testing; it should allow for more useful recommendations from the tests as to how to improve the system rather than issues someone less computer literate might have.

Test Plan:

The testing environment for this test will involve sitting next to the tester and instructing them to complete certain tasks on the application. After explaining, the procedure they will complete, the tester will simply observe the user as they attempt to complete the tasks. Only interfering if a fatal error occurs.

The two scenarios will be used to test the main functions of the application (CRUD operations on stored data, managing holiday requests and receiving notifications based on preferences).

1. Log in to the system, turn off notifications, select an employee and update their last name, and approve holiday requests.
2. Log in, turn on notifications, create a new employee, deny holiday requests, and then delete that employee.

The testing will be performed on the development machine using the development environment to ensure the application runs as expected. As the device has no touch capabilities, all the inputs onto the application will be completed using the trackpad on the device.

Below are images of the testers completing the test:



User Consent:

This testing is for a summative testing of a prototype employee management app developed for Android devices. As a user, you will be instructed on how to traverse the application and what tasks you will be expected to complete; any bugs encountered are not intended and should be noted after the testing. During the test, a photograph will be taken to prove your participation in this study. Following the test, a brief survey should be completed, where you can explain your opinions of the system.

I understand that my participation in this testing is voluntary, and I can withdraw at any time. I confirm I am 18 or over.

The data gained from this study will be anonymised and is for evaluation purposes only and only essential information will be recorded. The results of this user testing are for internal use only and personal data will not be stored after 01/03/2023.

Date: 23/1/23

Signature:



Date: 23/1/23

Signature:



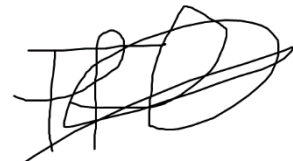
Date: 23/1/23

Signature:



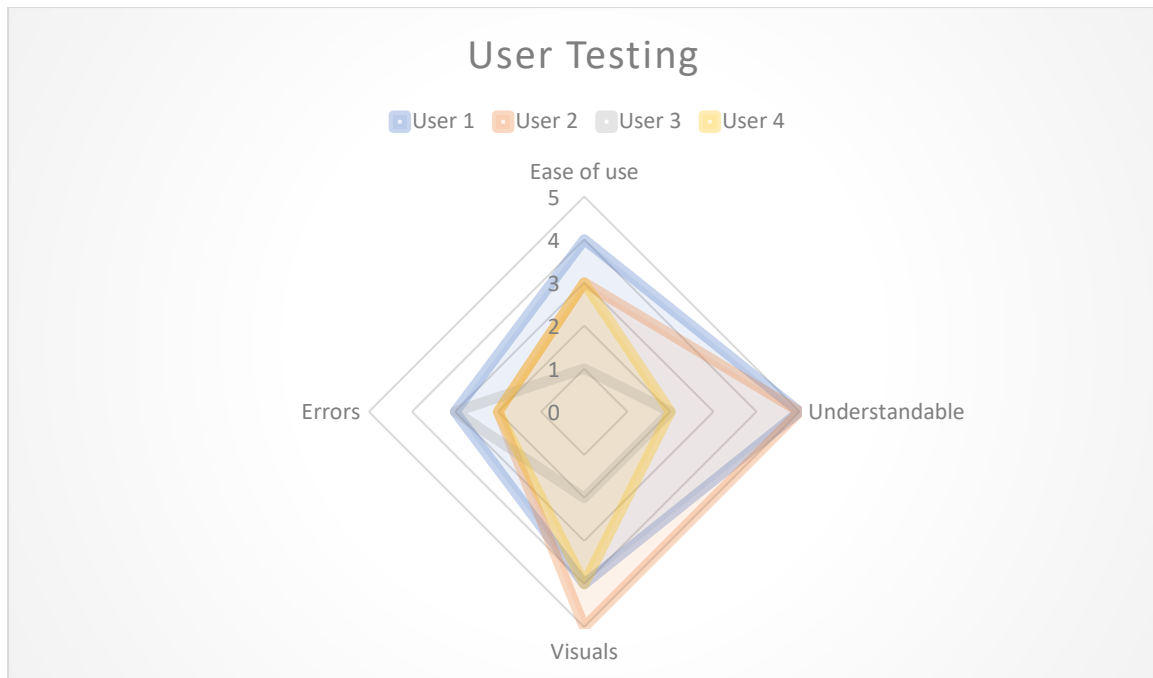
Date: 23/1/23

Signature:



Then their feedback and the tester's observations were combined to create the following findings.

Test Completed	Ease of use	Understandable	Visuals	Errors
1	4 Found most screens rather quickly	5 Notifications did not work (but no notifications were selected). Very clear employee adding	4 Clear screens	3 Shift button didn't work on keyboard.
2	3 Quickly found most windows	5 Correct notifications displayed, found all the buttons easily and understood their functions.	5 Simple colour scheme and all data found simply.	2 Deleted other employee's data rather than the one selected.
1	1 Required aid to find correct buttons, unclear which buttons had images on.	2 No notifications were changed (but they still appeared). Eventually found how to complete the tests.	2 If the buttons contained more colours, it could be simpler to show the different buttons.	3 Notifications appeared despite, selecting no notifications on device.
2	3 Easy to navigate	2 Somewhat but some text was blocked by keyboard. First use of android so personal inexperience for device.	4 Big buttons and easy to navigate. Approve button coloured clearly.	2 Keyboard was not responding correctly.



As shown above, the users clearly understand what the program is meant to do and generally easy to use. While being split between the design of the interfaces and the app contains several errors easily discovered during the user testing. These could be fixed in later versions as greater error detection could be incorporated and a more streamlined user interface could be designed.

Evaluation

Using this testing, a series of recommended changes could be made to the system including:

- Improving the error handling of the software to reduce the errors that occur when using the system.
- Allow the stored employees to be found by their name rather than just their ID, which the user of the system would be unlikely to know.
- Refine the user interfaces to incorporate back buttons rather than the one on the device.
- Improve log in functionality to check that the user is in the database.

Overall, the application has been completed and all the intended functionality has been incorporated. While this could simply be improved by having more security and error handling when attempting to access the API, it completes the required functionality. If there was more time, I would have hoped to fix some of these errors and tweak the interfaces a little more to improve the overall experience. Additionally, I had hoped to store more data than just the employee's ID and name in the database as well as combine the interactivity of the high-fidelity prototype to have the holiday requests come from a claim made on the employee side of the application. While a more structured development could have prevented some of these errors from occurring, it is acceptable that it has been completed in this way. Despite this, the application runs as expected and some of the proposed changes could be implemented to improve the quality of the program for a potential full release.

References

ⁱ What is data integrity? (2021) Catherine Cote, Harvard Business School Online <https://online.hbs.edu/blog/post/what-is-data-integrity> (Accessed: 22 January 2023).

ⁱⁱ Cost of a Data Breach 2022 (2022) IBM

<https://www.ibm.com/reports/data-breach> (Accessed: 22 January 2023).

ⁱⁱⁱ OWASP Top Ten (2021) OWASP Foundation.

<https://owasp.org/www-project-top-ten/> (Accessed: 22 January 2023).