# Assessment 2: Report

COMP2000 CW2, Corey Richardson

## Introduction

## Background

The produced application is a management tool for employees and admins within a company. It can be used by Employees to update their personal details, and submit Holiday (PTO) Requests, whereas administrative users can carry out a wider set of features in addition to those mentioned above. These features include viewing and managing employee records via CRUD operations and reviewing and actioning employee’s holiday requests.

When employee details or PTO request statuses are updated, a notification is pushed to the user. These notifications ensure users are informed of updates to their information, promoting communication.

The application relies on a backend RESTful API Web Service.

## Assessment Materials

GitHub Repository: [github.com/Plymouth-COMP2000/coursework-report-corey-richardson](https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_pto_menu.xml)

GitHub Release: [github.com/Plymouth-COMP2000/coursework-report-corey-richardson/releases/tag/cw2](https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/releases/tag/cw2)

Video Link (also include MP4 in submission!):

WakaTime Project: <wakatime.com/@coreyrichardson/projects/rwcthymumd?start=2024-12-10&end=2025-01-07>

## Design and Storyboarding

### Class Diagrams

|  |  |  |
| --- | --- | --- |
| Figure 1: UML Diagram for Employee Class | Figure 2: UML Diagram for PtoRequest Class | Figure 3: UML Diagram for UserSettings Class |

|  |
| --- |
| Figure 4: UML Diagram for EmployeeAdapter ListView Adapter |
| Figure 5: UML Diagram for PtoAdapter ListView Adapter |
| Figure 6: UML Diagram for aPtoAdapter ListView Adapter |

|  |
| --- |
| Figure 7: UML Diagram for DatabaseHelper Class |

|  |
| --- |
| Figure 8: UML Diagram for ApiService Class |

|  |
| --- |
| Figure 9: UML Diagram for NotificationHelper Class |

## Legal, Social, Ethical and Professional Considerations

Protecting user’s personal information would be an important aspect of the applications development. The local database stores private information such as the user’s password (in plaintext!) and both the local database and the web service API stores Personally Identifiable Information(PII) such as full names, email addresses and salaries. Passwords are stored in plaintext, as encryption falls out of scope of this module’s coursework, but this approach is not suitable for a production application.

SQL data transactions are used to ensure that data integrity is maintained; updates to the database are atomic, consistent, isolated and durable (ACID), mitigating the risk of a partial database update compromising data integrity.

The API Web service does not provide a Secure Sockets Layer (SSL) certificate and so communication with the API can only be done over an insecure and unencrypted HTTP connection, exposing the application to data interception attacks.

Ideally, all communication would be handled over HTTPS connections, and passwords would be encrypted using strong encryption algorithms such as SHA-256.

## Interface Design

This section displays the XML files, viewed in Android Studio, with a screenshot taken from the **Medium Phone API 35** emulator alongside it. Also included is a link to the XML file in the GitHub repository.

Consistent design was used across activities to improve the user experience of the application by ensuring that users can navigate the layout of the application intuitively. The uniformity of layouts was achieved using common elements like navigation buttons and input fields using a consistent size, style and positioning across all pages.

While not yet implemented, view-accessibility features such as a Dark Theme and a Red Green Colourblind theme were planned, accommodating users with disabilities and light sensitivities. The inclusion of these features in a future iteration would enhance the applications usability for a larger market audience.

|  |  |
| --- | --- |
| Figure 10: activity\_main.xml | Figure 11: MainActivity |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_main.xml> | |
| Figure 12: activity\_aadd\_employee.xml | Figure 13: aAddEmployee |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_aadd_employee.xml> | |
| Figure 14: activity\_aedit\_employee.xml | Figure 15: aEditEmployee |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_aedit_employee.xml> | |
| Figure 16: aemployee\_details.xml | Figure 17: aEmployeeDetails |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_aemployee_details.xml> | |
| Figure 18: activity\_apto\_requests.xml | Figure 19: aPtoRequests |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_apto_requests.xml> | |
| Figure 20: activity\_dashboard.xml | Figure 21: Dashboard (Employee)    Figure 22: Dashboard (Admin) |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_dashboard.xml> | |
| Figure 23: activity\_edit\_personal\_details.xml | Figure 24: EditPersonalDetails |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_edit_personal_details.xml> | |
| Figure 25: activity\_edit\_pto\_request.xml | Figure 26: EditPtoRequest |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_edit_pto_request.xml> | |
| Figure 27: activity\_pto\_menu.xml | Figure 28: PtoMenu |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_pto_menu.xml> | |
| Figure 29: activity\_request\_pto.xml | Figure 30: RequestPTO |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_request_pto.xml> | |
| Figure 31: activity\_settings.xml | Figure 32: UserSettings |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_settings.xml> | |
| Figure 33: activity\_view\_holiday.xml | Figure 34: ViewHoliday    Figure 35: ViewHoliday (Confirm cancellation?) |
| <https://github.com/Plymouth-COMP2000/coursework-report-corey-richardson/blob/main/SourceCode/app/src/main/res/layout/activity_view_holiday.xml> | |

## Implementation

### Functional Requirements of the Application

|  |  |
| --- | --- |
| **Admin Users** | |
| As an administrator I wish to add employees’ details. | Implemented |
| As an administrator I wish to check the employees’ details have been uploaded. | Implemented |
| As an administrator I wish to edit employees’ details. | Implemented |
| As an administrator I wish to delete an employee’s details. | Implemented |
| The app should also apply an automatic increment by 5% on the employee salary when they complete one year. | Not implemented |

|  |  |
| --- | --- |
| **Employee Users** | |
| As an employee I wish to view my details. | Implemented |
| As an employee I wish to edit my details. | Implemented |
| As an employee I wish to book/ manage my annual leave/ holiday (the allowance is 30 days per year). | Implemented |

### Activities

Activities are created with a combination of an XML file defining the layout of page to display, and a Java file which handles the dynamic content-generation. The application starts at the **MainActivity**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Activity** | **XML File** | **Java File** | **Explained** |
| MainActivity | activity\_main.xml | MainActivity.java | This activity holds the Sign In screen. The tester can use the two pre-defined user accounts (**admin@example.com**, **admin\_password** or **john.doe@example.com**, **employee\_password**) to test the applications functionality. |
| aAddEmployee | activity\_aadd\_employee.xml | aAddEmployee.java | This activity can be accessed by Admin users and contains a form to add new users into the local database and API service. |
| aEditEmployee | activity\_aedit\_employee.xml | aEditEmployee.java | This activity can be accessed by Admin users and contains a prefilled form with the selected user’s information. Here, the admin can edit the information and save the changes to the local database and API service. |
| aEmployeeDetails | activity\_aemployee\_details.xml | aEmployeeDetails.java | This activity holds a ListView item using the EmployeeAdapter which displays user information along with options for opening the aEditEmployee activity, and user record deletion. |
| aPtoRequests | activity\_apto\_requests.xml | aPtoRequests.java | This activity holds a ListView item using the aPtoAdapter which allows the Admin user to action employee’s holiday requests. |
| Dashboard | activity\_dashboard.xml | Dashboard.java | This is the main navigation hub of the application. |
| EditPersonalDetails | activity\_edit\_personal\_details.xml | EditPersonalDetails.java | This activity contains a prefilled form with the user’s information. The user can edit their information and save the changes to the local database and API service. |
| EditPtoRequests | activity\_edit\_pto\_request.xml | EditPtoRequests.java | This activity contains a prefilled form with the selected holiday requests information. The user can edit their requests details and save the changes to the local database. |
| ForgotPassword | activity\_forgot\_password.xml | ForgotPassword.java | Not functional. |
| ptoMenu | activity\_pto\_menu.xml | ptoMenu.java | This activity is a navigation page for selecting activities to Request PTO or Viewing Requests. |
| RequestPto | activity\_request\_pto.xml | RequestPTO.java | This activity contains a form where the user can submit new PTO requests. |
| Settings | activity\_settings.xml | Settings.java | This activity lets the user modify the setting Booleans stored in their userSettings attribute. |
| ViewHoliday | activity\_view\_holiday.xml | ViewHoliday.java | This activity holds a ListView item using the ptoAdapter which allows the user to see their requested and scheduled holiday. |

### ListView Items

|  |  |
| --- | --- |
| **Adapter Utilised By** | **XML File** |
| aPtoAdapter.java | admin\_pto\_list\_item.xml |
| EmployeeAdapter.java | employee\_list\_item.xml |
| PtoAdapter.java | pto\_list\_item.xml |

### Local Database Implementation

The application uses a local SQLite database for storing employee records, along with additional information not held by the API such as the Employee’s role and login information, as a short-term but persistent storage medium to minimise the number of GET requests sent to the API Web Service, reducing reliance for repetitive retrievals. This allows users read-access to their data even without an internet connection. SQLite was the chosen technology here due to its simple and lightweight implementation due to native support for Android.

The schema consists of three tables: User, UserSettings and PtoRequest.

|  |
| --- |
| Figure 36: SQLite Database Schema |

The SQLiteOpenHelper class is extended by DatabaseHelper to manage database creation, upgrade and functions.

|  |
| --- |
| Figure 37: DatabaseHelper extends SQLiteOpenHelper |

Data operations are handled through Data Access Object methods to Create, Read, Update or Delete records across the three tables.

As examples, here are the CRUD methods for the `Employee` table:

|  |
| --- |
| Figure 38: Example CREATE Operation - insertUser() |

|  |
| --- |
| Figure 39: Example READ Operation - getAllEmployees() |

|  |
| --- |
| Figure 40: Example UPDATE Operation - updateUserInDatabase() |

|  |
| --- |
| Figure 41: Example DELETE Operation - deleteEmployee() |

### API Integration

The integration of a RESTful API enables data management and synchronisation across devices and users. Periodic fetches from the API to the Local Database ensure the data is consistent and up to date.

On app startup, employee records are fetched from the API and stored locally. These records are then re-fetched, and new records are inserted into the local database each time the admin user adds a new user to the database. This uses lazy loading, meaning that new data is only loaded on demand to reduce the background memory usage of the application, at the cost of slowing the loading of this operation.

This approach comes with a few issues, namely, how to ensure a consistent data sync with the API when the user switches between online and offline web connections, how to manage data migrations between different schema versions and how to handle other users of the API deleting employee records.

|  |
| --- |
| Figure 42: fetchAndStoreEmployees() |

|  |  |  |
| --- | --- | --- |
| **Base URL:** | http://10.224.41.11/comp2000 | |
| **Endpoint** | **Method** | **Description** |
| /employees | GET | Retrieve a list of all employees in JSON format. |
| /employees/get/<int:id> | GET | Retrieve details of a specific employee by their ID. |
| /employees/add | POST | Add a new employee to the system. ‘firstname’, ‘lastname’, ‘email’, ‘department’, ‘salary’ and ‘joiningdate’ fields must be present in the request body. |
| /employees/edit/<int:id> | PUT | Updates an existing employee’s information. ‘firstname’, ‘lastname’, ‘email’, ‘department’, ‘salary’ and ‘joiningdate’ fields must be present in the request body. |
| /employees/delete/<int:id> | DELETE | Delete an employee by their ID. The ID of the employee to delete must be present in the request body. |
| /health | GET | Check if the API is running. On a 200 OK response, the API returns a status message “API is working”. |

## Key Design Features

### SOLID Principles

I used SOLID principles to ensure that my application was maintainable, scalable and robust.

Following Single Responsibility Principle, each class in the application is designed to have a single responsibility. The **DatabaseHelper** class handles all operations relating to the local SQLite database, such as the CRUD operations for employees and holiday requests. The **ApiHelper** class manages communication with the API and utilises methods from the DatabaseHelper class to ensure data synchronisation between the two stores. The **NotificationHelper** class handles the creation and display of notifications. This separation of concerns ensures loose coupling between classes, meaning that changes to one ‘responsibility’, does not have secondary impacts of unrelated areas of the codebase. This use of abstracted layers also means that the codebase follows the Open-Closed Principle, as extending these functionalities and classes does not change the core implementation of the application.

As an aspect of Liskov Substitution Principle, any object of a superclass can be replaced by an object of a subclass without impacting the program. This could be useful, especially during testing, to substitute the **ApiHelper** class with an interface mocking the implementation.

The Interface Segregation Principle ensures interfaces/classes are designed to be specific and only have methods relevant to their needs, in an effort to avoid bloated classes with unnecessary methods.

### Use of Worker Threads via AsyncTask

The application uses worker threads using **AsyncTasks** to handle API operations in the background, ensuring that the Android UI remains responsive even during intensive, long running operations. If intensive tasks are performed on the main thread, the UI can freeze, degrading user experience or leading to application crashes. AsyncTasks execute tasks in the background and updates the UI upon completion, enabling the application to handle these intensive tasks without affecting performance.

Fetching all the employee records stored on the API takes a longer time period, therefore an AsyncTask process was used to perform this network request on the background thread. Loading this dataset, or performing many CRUD operations at a similar time, could block the UI thread if it wasn’t executed in the background.

AsyncTask has been deprecated from the Android API and so ExecutorService should be preferred in future projects. Unlike AsyncTask, ExecutorService can handle multiple threads allowing it to handle multiple concurrent tasks.

## Summary

The project successfully delivered a functional, user-friendly application for employee and PTO management. The adoption of best practices ensured scalability and maintainability.

## References

* Include some Android documentation used here? Currently included as code comments
* Volley documentation
* AlertDialog