**Lyle Simpson: S1436436**

**Mobile Platform Development - Test Report**

The application was rigorously tested to ensure there were no runtime errors and no logical errors were present, and that each action within the user interface produced the expected response.

The application was tested to ensure that each activity is represented correctly in both landscape and portrait mode, with alternative layouts created when necessary. For example, the earthquake item view displays a map representing the earthquake’s location, with some text underneath representing the data associated with that earthquake. This layout works well in portrait mode, however in landscape mode the text details disappear off the screen underneath the map. To remedy this, a new layout was created for landscape mode, where the map was placed on the left half of the screen, with the text details displayed on the right half. This allowed the user to avoid scrolling and keeps all the information displayable in a clean, intuitive manner for the user.

At the bottom of each activity in the application, there is a navigation bar which allows movement through different pages. Being the central navigation component, this had to be tested thoroughly. On each activity, every navigation item was clicked, to ensure it led to the correct page. This process was repeated on all activities in the application, to ensure the critical navigation actions worked correctly. In addition to this navigation bar, within the list view there also exists a toolbar, complete with a variety of options for the user to choose. These include options for searching the list of earthquakes, ordering the list according to some criteria, and filtering the list by some metric such as the magnitude or depth.

On the earthquake list activity, the recycler-view can be ordered according to an option the user selects on the menu. The list can be sorted according to the following options:

1. Sorted by date earthquake occurred
2. Sorted by alphabetical order of location
3. Sorted by the earthquake’s magnitude
4. Sorted by the earthquake’s depth
5. Sorted by the earthquake’s geographical coordinates (most northern, western, eastern, southern)

To facilitate these changes to the ordering of the list, Java Comparator objects were used to determine a variety of different sorting rules for each of the criteria. These objects were located within the Repository of the application, and were combined with the Java Collections library’s sort() method to produce new lists containing the items in the desired order. For naturally opposite orderings – for example, sorting by the largest magnitude, and sorting by the smallest magnitude – the results of the Comparator were simply reversed, using the Collections library’s reverse() method. These Collections static methods are built-in to the library, and are rigorously tested – and by using these methods, the ordering is guaranteed to be correct as long as the Comparator logic is bulletproof. To further test each of these re-orderings, their results were thoroughly analysed in the user interface, and were each determined to be correct.

In addition to ordering, the user has the ability to search through the list of earthquakes by entering text into a search field representing the location of the earthquake – for example, “Glasgow”. The list will filter in real-time to display only those earthquakes whose location contains the substring entered. The comparison between the user’s query and the location is done in a case insensitive manner, and this feature was thoroughly tested to ensure the correct results are returned to the list. The search function was tested with a variety of uppercase and lowercase characters to ensure the case did not affect the query results.

The user also has the ability to filter the list by all earthquakes that occurred on a particular date, with the date chosen by the user. This feature again was tested thoroughly, with the logic to perform this filtering located within the Repository class. The user selects a date using a Date Picker user interface widget, and this date is converted on the backend to a LocalDate object for easy comparison with earthquake dates within the repository’s List of earthquakes. Dates with known earthquakes were tested, with the expected results achieved. Dates with no earthquakes were also tested, and the interface correctly presented a Toast message to the user informing them that no earthquakes were discovered on that date.

For the statistics activity, the battle-tested Collections library method max() is used along with a comparator to display the maximum magnitude and the maximum depth for all the earthquakes gathered by the API. Java HashMap and Map.Entry objects are used to associated String keys with values to populate the statistics for where most earthquakes occurred, for which date had the greatest number of earthquakes, and for which hourly time period had the greatest frequency of earthquakes. These maps are constructed in the Repository class, again separating the logic of calculating these statistics from the activity itself, which is good programming practice.

All of the components within the application were debugged using the logcat command line tool. Within each component, Logging messages were generated upon important actions, and these logs were written to the logcat standard output. When debugging logical errors, the contents of Java data structures were often dumped to the logs in String format, in order to inspect whether the issue lay within the backend logical processing, or somewhere else. The application’s use of LiveData from the Android architecture components allowed ViewModels to observe changes to the data, and this was another important use of the Logging operations. Debug logs were printed to logcat whenever any changes to the underlying earthquake data occurred, allowing for introspection of whether the data refresh functionality was working. Logs confirmed that the data was being regathered every 5 minutes, as per the periodic asynchronous timer defined in the application. Logging statements were also vital in the creation of the XmlPullParser logic, as it took a little while to get this logic accurate. The logcat utility allowed debugging of XmlPullParser events, and ultimately led to the successful building of the domain models used in the application (Earthquake, Location, and Coordinates models). Logging was also useful within the WebService class, which is responsible for fetching the XML data on a separate thread. The network request class implements Java’s Callable interface, with the network work being done in its call() method. This means the work is done on a separate thread, and the callable can return the XML data upon completion of the request (which differentiates it from a Runnable, which cannot return data). When the user requests to refresh the data, the following logs confirm what’s happening:



The logcat utility also allowed system messages such as stack traces to be viewed, which allowed easier identification of where in the application errors were being thrown.

The application’s use of maps was another important area that needed thorough testing. There are two different uses of maps in the application. Firstly, a global map view of the location of every earthquake retrieved from the API. This had markers for each earthquake, along with a tooltip showing where it occurred and the date on which it occurred. Through rigorous testing, each marker’s tooltip was checked, to ensure it was associated with the correct earthquake.

The second use of maps was the detail view for each individual earthquake, which is accessible when clicking the list view item that holds the earthquake. When an item in the list view is clicked, it should load up a detail view for that earthquake, displaying more information about it, including the map. Two tests occurred for this particular process. Firstly, to ensure that the correct detail view is loaded when an item in the list is clicked, and that all the associated information such as magnitude, date and depth are correctly represented in the detail view. Secondly, to ensure that the map is displaying a marker representing the location of the earthquake, in the correct place.

A final component of the application that required testing was the database. The data, after being fetched from the API, was stored in a SQLite database, using the Room object-relational mapping library. When data was re-fetched via the user prompting this action, it would replace existing data within the database, and would not append duplicates to the Earthquake table. It was vital to test that duplicates were not created, as this would poison the data. It was also necessary to handle the conversion of Java LocalDateTime objects into native SQL date data-types. This was handled by creating a custom TypeConverter object, using the Room library. All these tools were tested thoroughly using logcat before deployment.

**Conclusion and Scope For Improvement**

To conclude, the application is thoroughly tested for the majority of the functional requirements implemented. There is, however, scope for improvement. The project did not ultimately have any unit testing, due to time constraints. A big goal of future iterations would be to implement a comprehensive suite of automated unit tests to make the application more bullet-proof to future changes.