Design Proposal
Cell Application Team
LFEV-Y5-2017

1. Introduction

This document will provide the necessary details about the Cell Phone Application development and how the application will work. The name of the application is currently "LFEV-Scada-Mobile".

We hope to build a observation tool that is mobile through tablet and phone environments. The users of LFEV-Scada-Mobile are people who are not necessarily knowledgeable about LFEV system itself. The users will also be provided a tutorial and so we aim to build a software that can be used by basically anyone.

Key aspects of the project to be implemented:

- Must be able to run both on tablets and cell phones
- Needs to use Java as the main language for easy accessibility for future teams
- Uses Gson parsing from a webserver for information gathering
- Read-only access to the data from the LFEV-SCADA system database
- Generates different types of displays for expanded functionality
- Be able to display the data required in Statement of Work R002k
- Follow the Formula EV rules

Current priorities:

- Preset Charts/Diagrams -- High Priority
- Number Display -- High Priority
- Speedometer/Odometer -- High Priority
- Notifications + Threshold -- High Priority
- Log Messages -- High Priority
- Custom Chart Generation -- Medium Priority
- Google Maps Coordinates -- Medium Priority
- Time adjustment to Charts -- Medium Priority
- Tutorial -- Medium Priority

2. Process View

Here we will present some activity diagrams that we have prepared for the application.

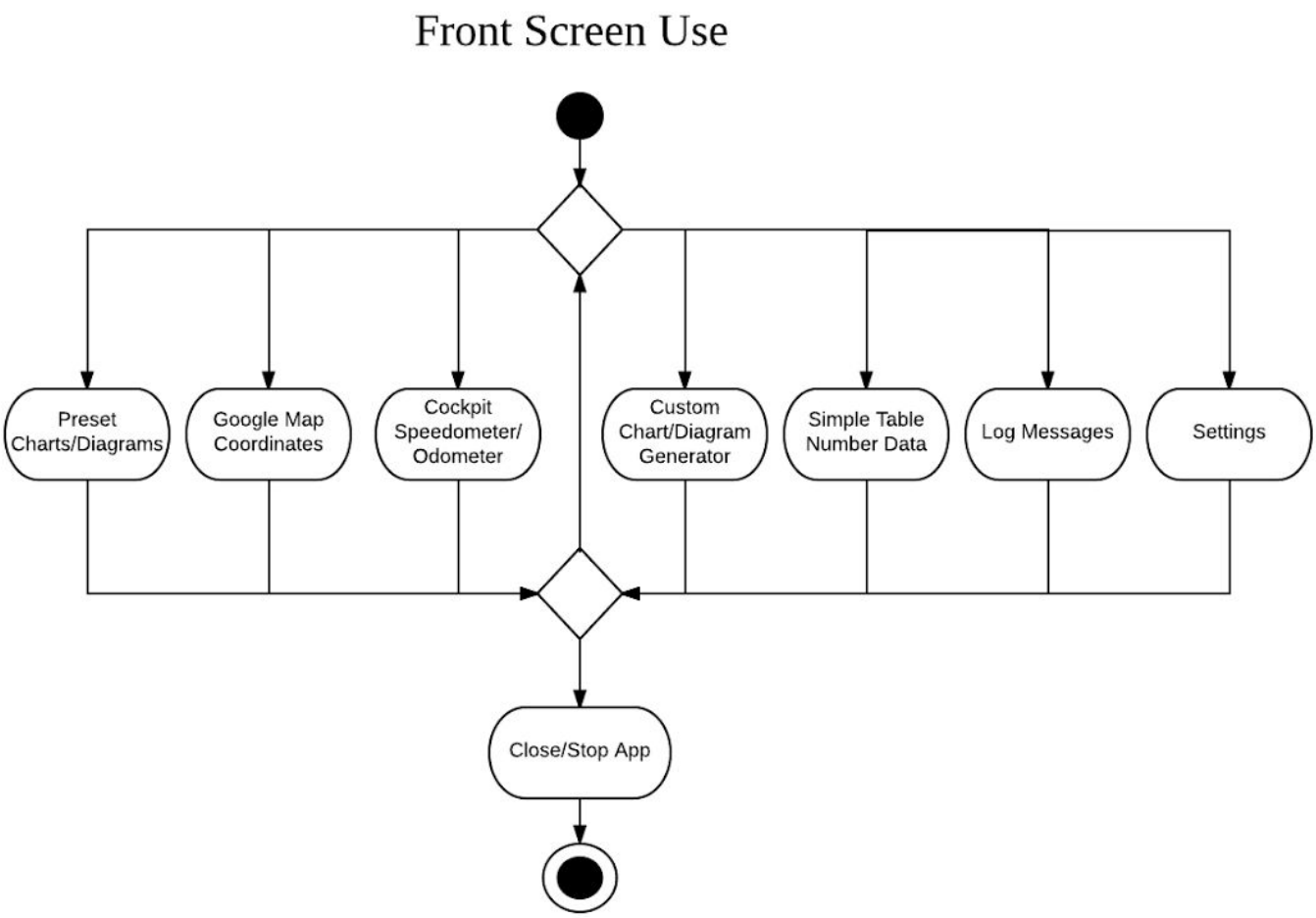


Figure 2.1: Front-Page Activity Diagram -- Shows what options the user will have access to and how the general application will update the display according to the choices of the user.

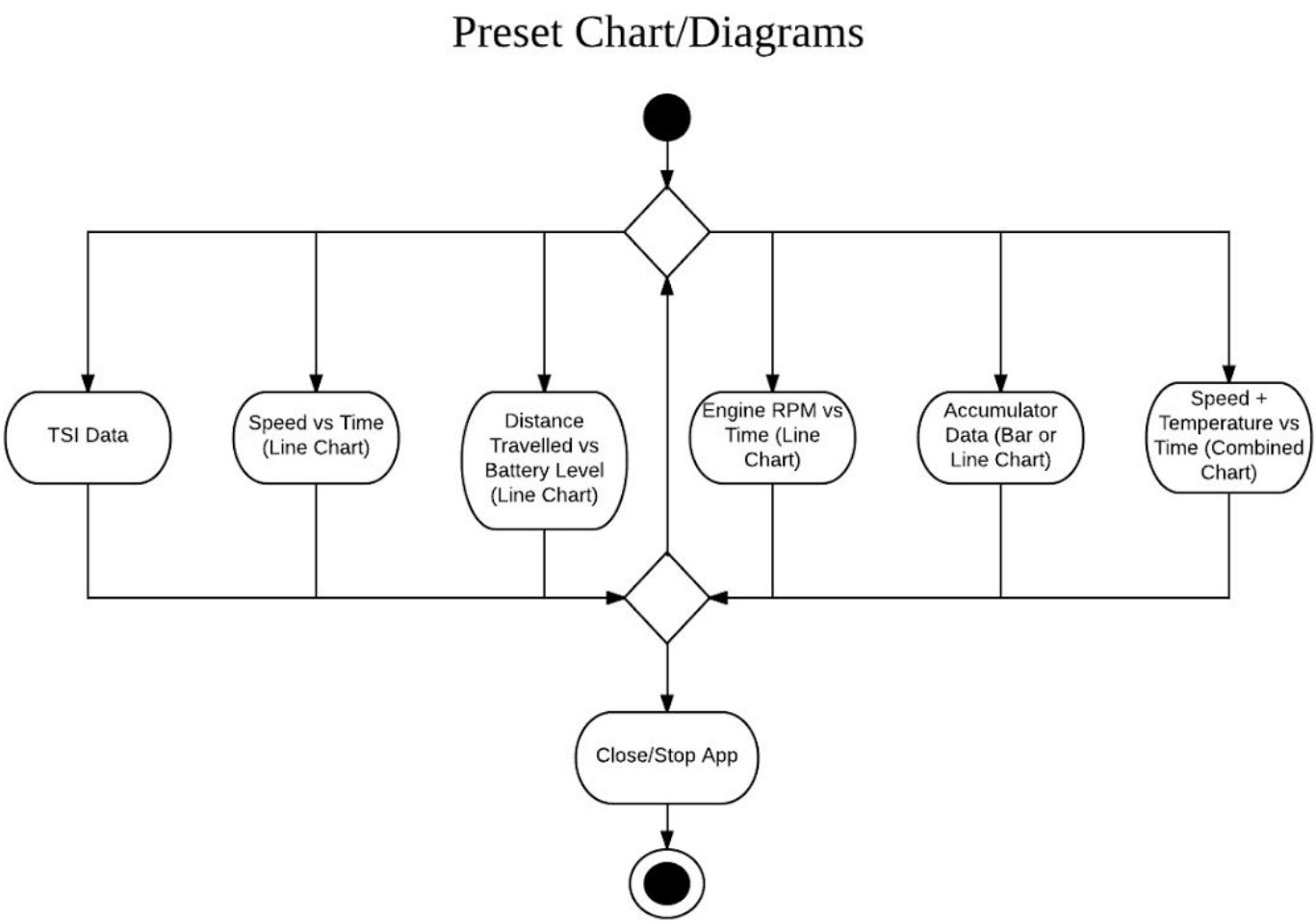


Figure 2.2: Preset Chars Activity Diagram -- Shows the charts that are offered by the application by default. These options does branch out more into other options as it can be seen in the other figures.

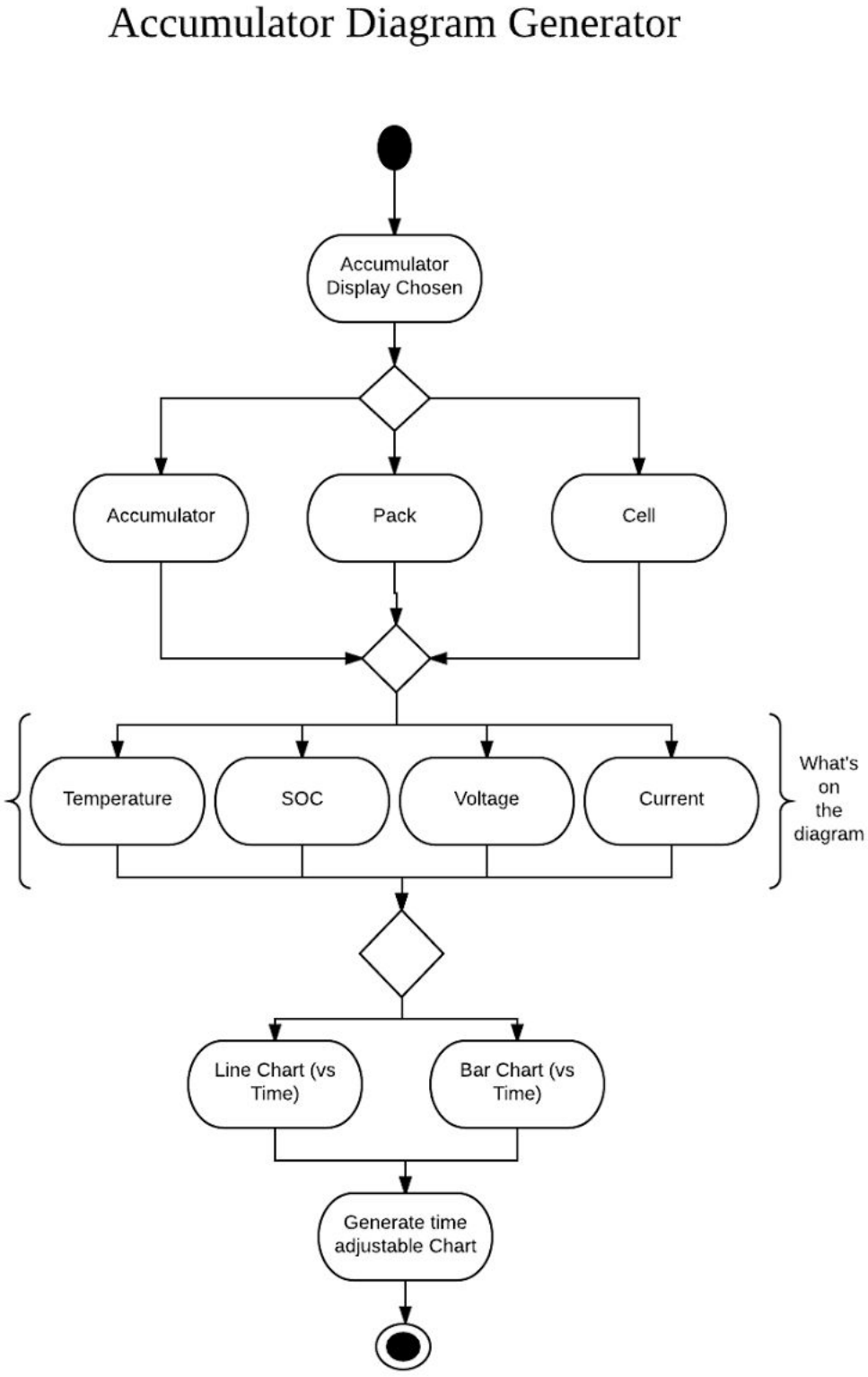


Figure 2.3: Accumulator Chart Generator Activity Diagram -- This shows how a chart will be generated according to the choices of the user. Examples of these are shown later in the document.

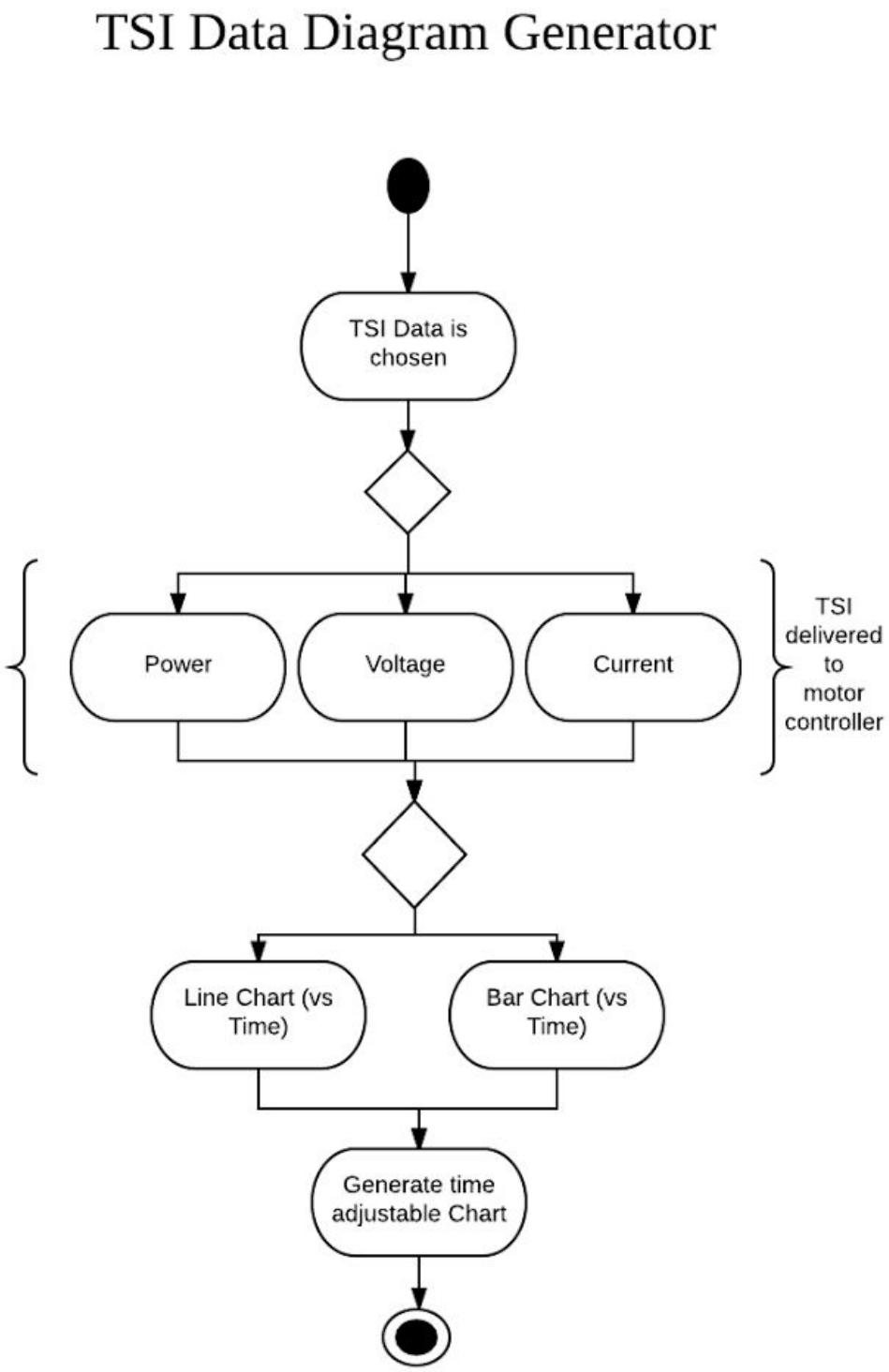


Figure 2.4: TSI Data Chart Generation Activity Diagram -- Similar to figure 2.3, shows the process for TSI Data charts

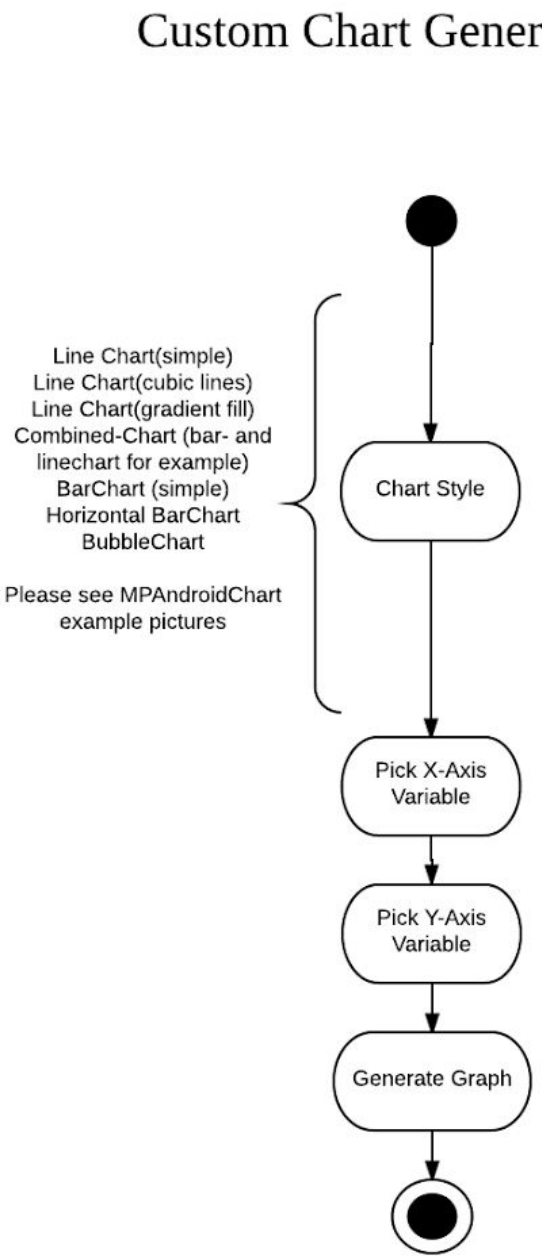


Figure 2.5: Custom Chart Generation Activity Diagram -- Custom chart generation will be fairly simple but the options for the custom generation are not yet determined. A further documentation will be provided with tutorial and later during development.

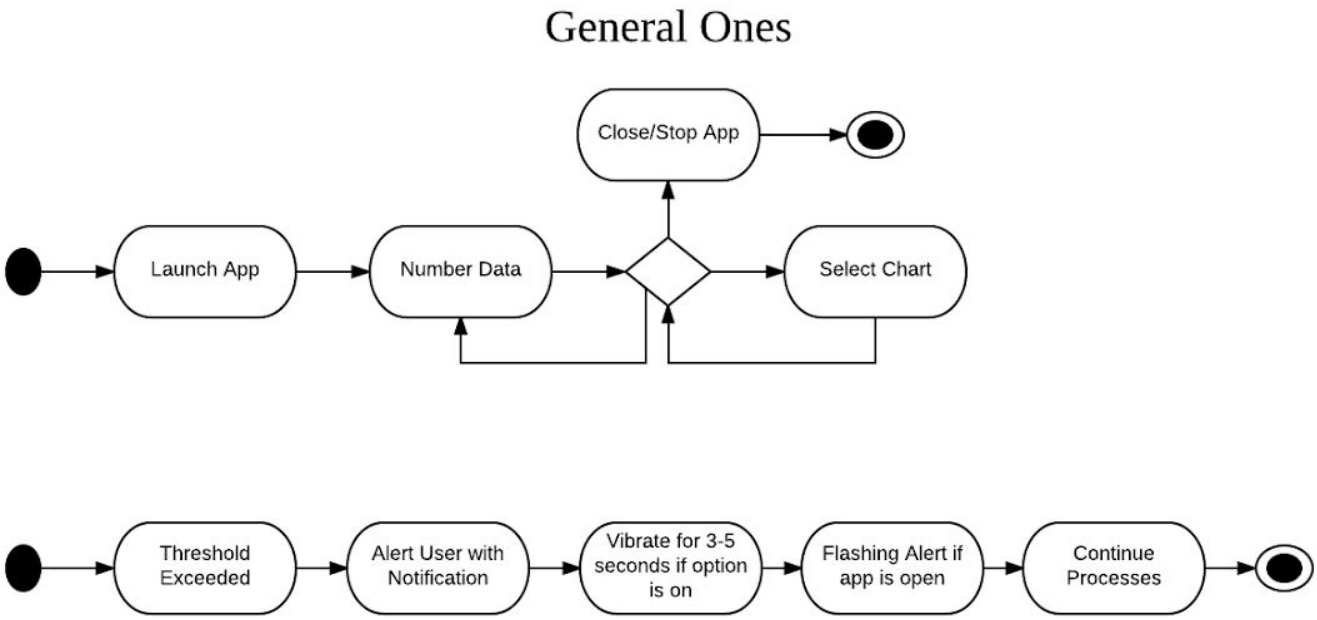


Figure 2.6: General Activity Diagrams: These diagrams show the notification process and the very simple activity diagram of the general application.

These activity diagrams provide a simple explanation for how the application is supposed to work once it is completed.

3. Development View

Another important aspect of the application is the classes that will be created for the program. Here we have created a Content UML diagram:

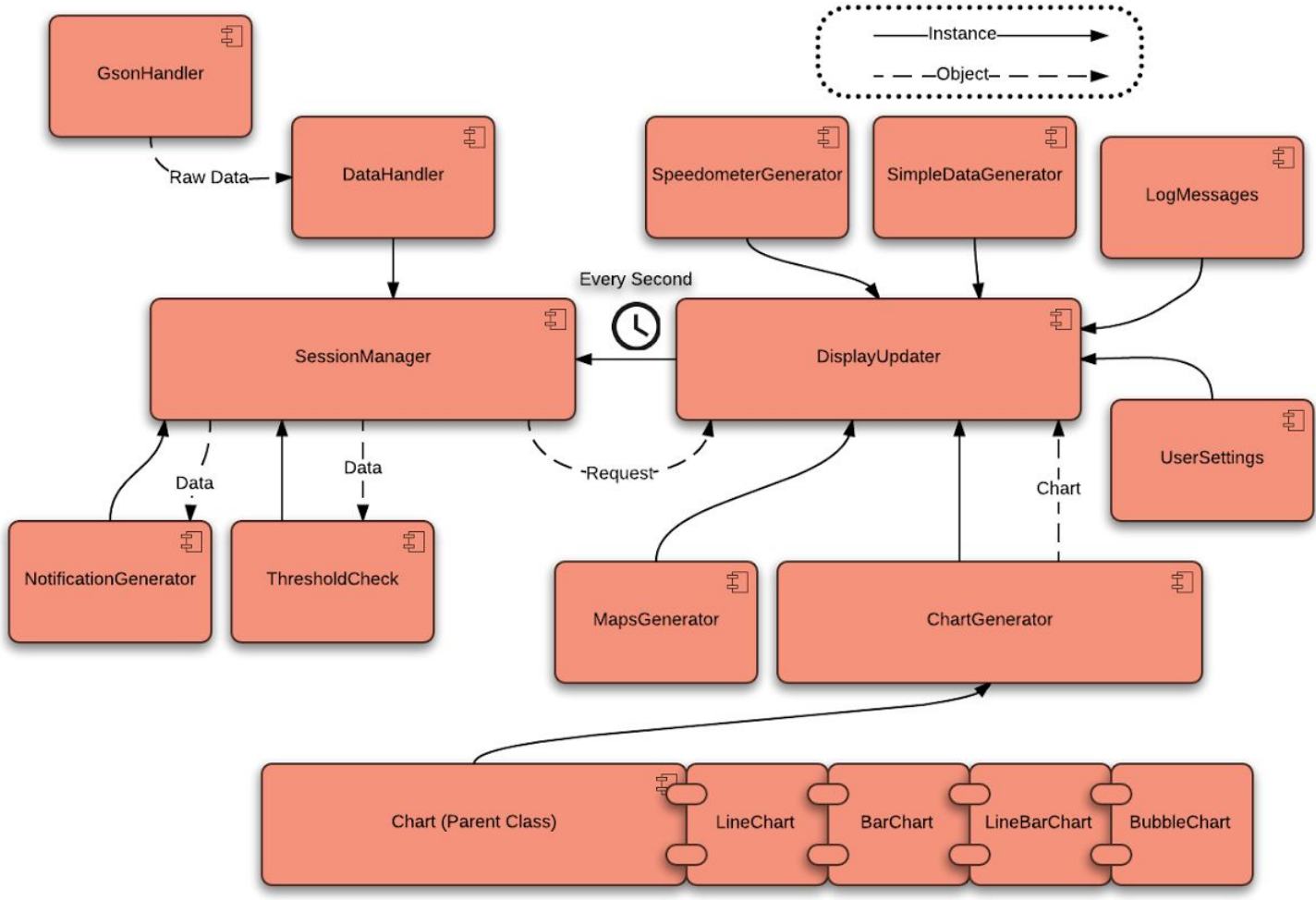


Figure 3.1: Development View -- Here you can see the classes and how they interact with each other in a very high level case. Each of these classes will be tested individually and the functionality will be demonstrated as the projects continues on. Two biggest components are DisplayUpdater and SessionManager.

A further UML diagram of the classes, a class diagram will be created and demonstrated.

4. Chart Examples

Figure 4.1: Speed vs Time Line Chart -- Will be improved and will have a time adjustability functionality

Figure 4.2: Single Pack vs Time display -- Here you can see that the time is adjusted to display the data from every hour. And this graph is generated through Accumulator Chart generation which is explained in figure 2.3. Only Pack 4 is shown here but it can be adjusted as the whole accumulator or even a single cell.

Figure 4.3: 7 Cells are displayed at the same time in a pack. Here we are looking inside the Pack 4 and each cell has a data that is updated every second from the database. This BarChart shows how the chart can be used easily to compare the cells to each other. Also if you compare this figure to Figure 4.2, you can see how Bar Chart can be used both against time for getting an update and also amongst the parts of the system for easy comparison.

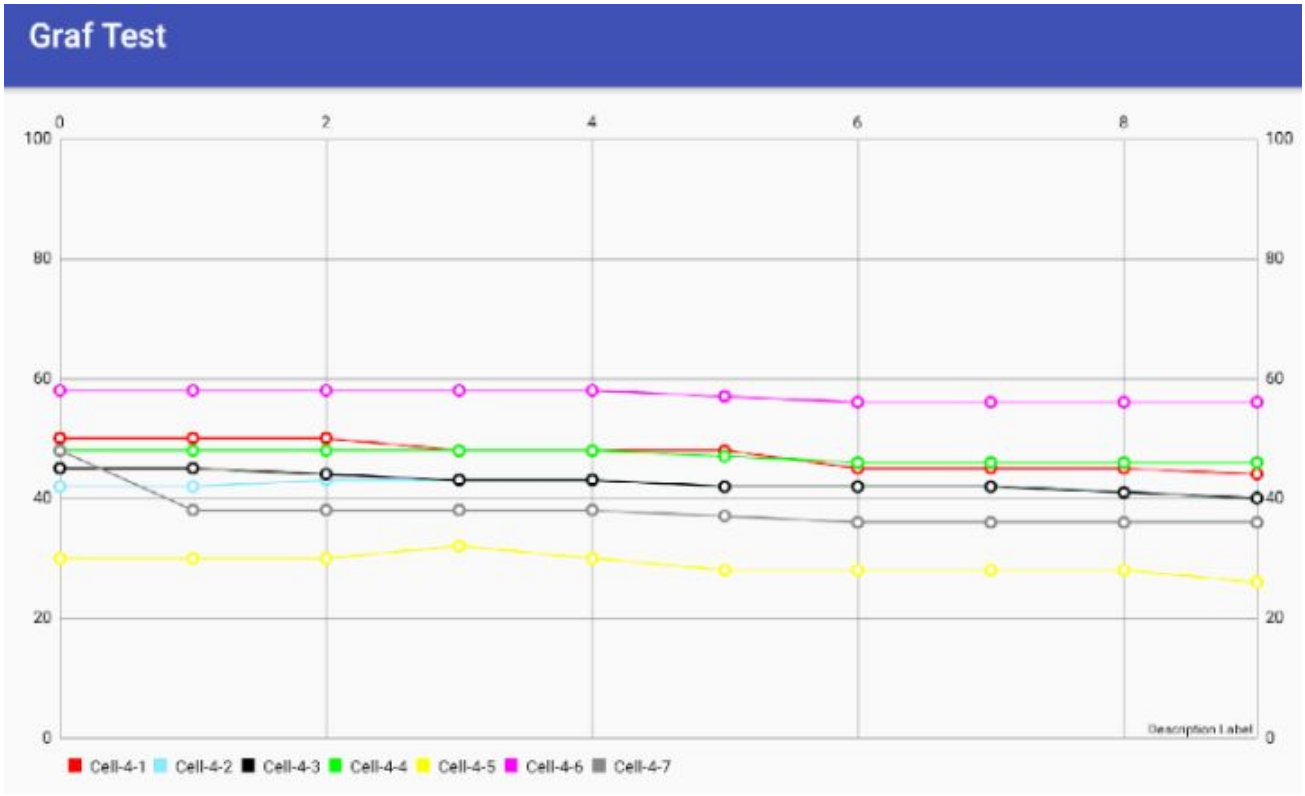


Figure 4.4: Cells Line Chart -- Again similar to figure 4.3, here we are looking at the seven cells from pack 4. However, this time we can see the data in a Vs time LineChart and combine the functionality of comparing cells with the functionality of seeing time update. This again is an example of accumulator chart generation from figure 2.3.

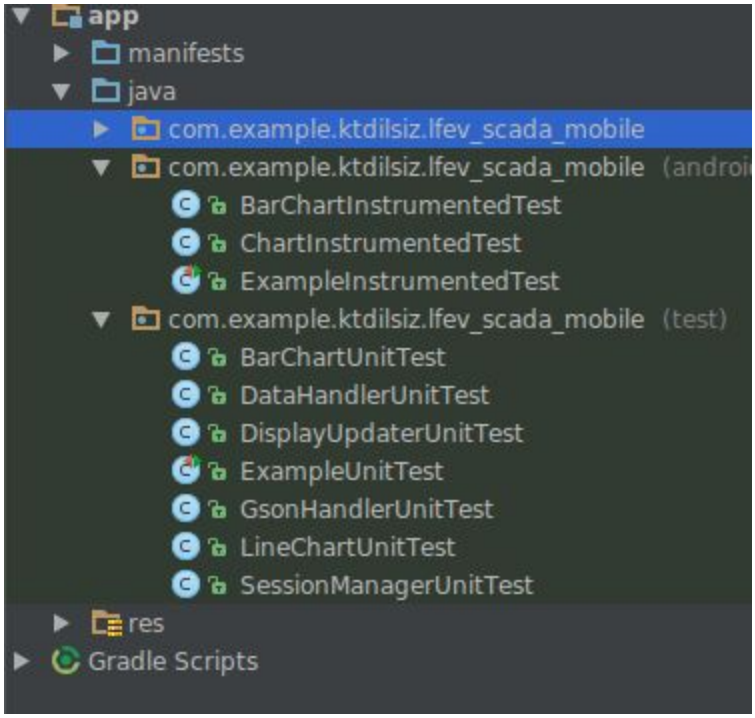
Figure 4.5: Emulator View Horizontal

Figure 4.6: Emulator View Vertical -- Currently there is only one chart is displayed but it will be updated.

The cell phone application will be similar in terms of display and zoom will be available.

5. Testing Protocol

Testing will be done through Android Studio standards.



Here you can see how we have created separate unit test files for each class. There will be two kinds of tests, “InstrumentedTests” and “UnitTests”.

Unit Tests run on the local JVM and Instrumented Tests run on the Android device. For more information please visit:

https://developer.android.com/training/testing/start/index.html

Example Unit Test Code:

```
/**
* Example local unit test, which will execute on the development machine (host).
*
* @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
*/
public class ExampleUnitTest {
  @Test
  public void addition_isCorrect() throws Exception {
     assertEquals(4, 2 + 2);
  }
}
```

6. Documentation Protocol

For documenting our methods and classes, we will use Javadoc structure. I will first give an example and then talk about it more specifically.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute{@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
 public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
 }
```

Example 6.1: Taken from: http://www.oracle.com/technetwork/articles/java/index-137868.html

Here as you can see javadoc requires the explanation for the method, the parameters, the return object and the references for the code. This generates a very nice javadoc documentation that can be easily converted into an API style HTML source.

This documentation will make the project sustainability much easier and also make the application much more accessible to the future teams for further expandability.

7. Conclusion

Our design is at a very primitive stage but we have a general idea of the application that we want to make following the procedures of the LFEV statement of work.

We hope to expand our Design Proposal and submit a second one in the upcoming weeks. However, with the current stage, this will give us a good idea if we are on the right track.