**Prior to development:**

I conducted an in-depth technology scan and feasibility analysis prior to commencing development.

<https://github.com/Danielle-Heinrich/ICT-Project/raw/master/Technology%20Scan%20and%20Feasibility%20Analysis.docx>

**Traccar Client development:**

For the Android application, I chose to build upon Traccar Client. The current release of Traccar Client utilises OsmAnd (OpenStreetMap Automated Navigation Directions) protocol for sending messages to a server:

*http://demo.traccar.org:5055/?id=123456&lat={0}&lon={1}&timestamp={2}&hdop={3}&altitude={4}&speed={5}*

The OsmAnd protocol sends messages appended to a URL query string.

However, the client requested that messages be sent in a specific three-line format, known as TLT-2H protocol, as a TCP payload.

*#868323028789359#MT600#0000#AUTO#1*

*#07d8cd5198$GPRMC,164934.00,A,1814.4854,N,09926.0566,E,0.03,,240417,,,A\*4A*

*##*

I had a look at previous releases of the application and found that prior to implementing the OsmAnd protocol, Traccar Client used a proprietary, extensible message format. The first two lines were mandatory, while the third line could be appended to the message through an option on the application interface:

*$PGID,123456789012345\*0F*

*$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E\*68*

*$TRCCR,20140111000000.000,A,60.000000,60.000000,0.00,0.00,0.00,50,\*3a*

As shown, this message format better resembles the format required by the sponsor, and also utilises TCP as a transport layer.

Therefore, I made the decision to build upon an old release of the application, which was available on GitHub.

I used Android Studio as my development environment, which I have never used before, but as it is based on IntelliJ the learning curve was not too steep. The most difficult part was understanding the associated Android libraries and configuring event handlers. As I was using an old release, there was also an issue of the build tool (Gradle) being out of date and some of the properties being deprecated. I had to fix these issues before I could even build the project without error. Overall, I found modifying the interface through the Layout Editor to be intuitive and fun. I removed the ability to toggle message extensibility and added a button to automatically set the device identifier to the IMEI (International Mobile Equipment Identity) number of the device. I also set the default values for server and port. Most importantly, I modified the string builder for the message format so that the output string matched the format required by the client.

**Traccar Server development:**

For the server-side application, I chose to build upon Traccar Server, using a second-party modified variant of the web interface which provided additional features. I stood the server up on a cloud-based EC2-VPC platform as it required a public facing IP address. Initially I chose to use Nectar Cloud, which provides scalable compute infrastructure, software and services for free to Australian researchers and students. I was able to install Traccar Server and the base web interface without issue. However, after a week or so, my EC2 instance experienced a major fault and was stuck in “Hard Reboot” mode. It took several days of communication with Nectar Support to resolve this issue and everything appeared to be working correctly again. A few days later, I began running into issues in connecting to particular ports on my instance. The Traccar Server provides a range of ports for devices with differing message protocols to forward their messages to. For example, devices using the OsmAnd protocol connect to port 5055 by default, while devices using the TLT-2H protocol connect to port 5030 by default. I spent a week attempting to resolve this issue, including researching the problem and building my understanding of firewall rules, contacting Nectar Support, and speaking to lecturers at University. I used a number of tools to try and resolve the issue, including nmap, netcat and packetsender. The port would appear open on the localhost, but attempting to connect from an external host would fail. In the end, I stood up a EC2-VPC instance through Amazon Web Services and had no networking issues. This leads me to believe that the port issue was in fact on Nectars side, and that my access and security groups were not being applied to my instance correctly.

The biggest issue I faced with installing Traccar Server was configuring a database. The software comes bundled with a H2 database which operates embedded in the application and is suitable for development but not for release. I chose to create a MySQL database instead. I have never done this before, especially through the command line interface, and found it quite challenging. My EC2 instance does not offer a GUI so I used a SQL client called Squirrel, which allows you to view the structure and tables of a compliant database, to visually confirm my database was correct. Once I had configured a MySQL database and user with permissions to access the database, I had to modify the configuration file bundled with Traccar Server with the driver, URL, user and password of my database.

Once the Traccar Server was installed and functioning correctly, I started work on further developing the web interface. The web interface was written in Java, so I initially chose Eclipse as my IDE. However, the project required Maven, Google and Git plugins to build correctly on Eclipse and I could not get it to configure correctly. I decided to use IntelliJ instead and downloaded and installed the community version. Once it was installed, I attempted to build the project but discovered that the community version did not support some of the required frameworks, such as JPA (Java Persistence API). This meant I had to sign up and install the Ultimate version of IntelliJ, which is fortunately free for a year for students. In addition to JPA, I needed to configure GWT for debugging in browser, Git for checking out the project (although I could have easily downloaded the source code manually), and Maven for building the war artefact. As I was developing on my own personal computer but standing up a server on a Cloud based machine, I needed to use FileZilla to transfer my war file between computers every time it was built. This was a bit time consuming and I needed to stop and restart the Traccar service each time. Installing the modified web UI for the first time also involved dropping the existing database and heavily modifying the configuration files bundled with the server application. This caused a lot of grief as if the configuration wasn’t carried out in a specific order, the server would fail to start or the database would not connect.

During development, GWT suddenly stopped compiling the modules required to access the application in a browser. The code server was being created, but the modules were not. I researched the issue extensively and even got in contact with the official GWT community. They attempted to resolve my problem, but in the end, after trying many solutions, I found a fix on my own. The answer was to delete the local instance of the H2 database, which would be recreated upon a new build. I suspect my database became corrupted or out of sync when I began adding image resources to the project (the military symbology).

Overall, I found the biggest challenge was a lack of documentation. The source code was not commented and did not come with Javadocs. Configuring the client or server to suit the sponsors needs was difficult as it meant I needed to have a firm understanding of data flows and dependencies between software classes. There were no system design diagrams available so I could not readily tell how the applications communicated between one another or served information to a browser. The installation and build instructions were minimal and it took me a few weeks to configure my development and production environment correctly.

**Before and After images:**

<https://github.com/Danielle-Heinrich/ICT-Project/raw/master/Before%20and%20After.docx>