Flight Tracker – GreenFlights – Phase 3

Team 4

Members:

Griffin Baxter

Swapnil Bhagat

Kye Oldham

Darryl Alang

Noah Irving

Submission Date:

5/10/2020 – 5pm

Table of Contents

Executive summary

In the modern era, climate change has become a significant issue. It is observed to have a detrimental impact on the environment. One factor that contributes to this impact is carbon emissions from aircraft. This project aims to develop an easy-to-use flight tracking application that provides reliable information about flight carbon emissions, alongside general data and map visualisations on flight routes, paths, airports, and airlines. This application is developed using Java and can be run on a local PC with Java runtime environment using the Linux operating system.

This application provides basic functionalities such as viewing, searching, filtering, and managing airport, airline, and route data. Additionally, it provides a close estimate of the amount of carbon emissions of a given route. It also presents ways for individuals to offset emissions such as environmental donations and the equivalent number of trees to plant.

The main stakeholders of this application include flight passengers, environmentalists, flight enthusiasts, the SENG 202 course staff and development team. The main risks identified are team conflict, COVID-19, and project ambition; however, the team has outlined in-detail the methods for preventing and mitigating such risks.

Overall, unlike other flight tracking applications, this does not only provide the end-user information related to flights, such as airports, airlines, routes and flight paths, but also information about the amount of carbon emitted by a given flight or set of flights and how to offset them.

Change log

Phase 2 Changes

Section numbers	Section titles	Change	Who changed
• 8	• Current product version	Started and finished most of the "current product version" section of the design document, which discussed various features of our current application.	Griffin Baxter
• 8 • 1 (1.1, 1.2)	• Current product version • Business and system context (Relevant business information and System context)	Added more details and paragraphs to the "Current product version" and "Business and system context" sections of the design document, including a system context diagram. Also, added numbered subheadings to the sections that require them.	Griffin Baxter
• 1 (1.1, 1.2)	• Relevant business information (Relevant business information and System context)	Completed the new business and system context section of our design document. This included adding more specific details on our flight tracking application, alongside significantly expanding our system context section.	Griffin Baxter
• 9	• Risk assessment	Updated the risk assessment section with better planned and detailed mitigation and prevention strategies. Also, updated the impact and likelihood ratings to reflect this, with a low to high rating rather than the use of unexplained numbers as before.	Griffin Baxter
• 2 (2.1)	• Stakeholders	Removed some unrealistic stakeholders and adjusted priorities. Sorted table by priorities	Swapnil Bhagat
• 2 (2.1)	• Stakeholders	Re-did stakeholders and added more comprehensive paragraphs under the 'concerns' column and, using team feedback, did some renaming/reprioritisation.	Swapnil Bhagat
• 2 (2.2.1)	• Use Cases (Diagram)	Re-did use case diagram based on Matthias' feedback session as well as a discussion with the team on which use cases to keep/remove/add. Inheritance between actors was removed as per Matthias' recommendations and use cases were more condensed but elaborated on in the textual use cases.	Swapnil Bhagat

• 1 (1.2), 2 (2.1)	• Systems Context, Stakeholders	Changed instances of ‘Environmentalist’ to ‘Environmentalist’ as the latter is more widely used and specific.	Swapnil Bhagat
• 2 (2.2.1)	• Use Cases (Textual Representation of Use Cases)	Re-did all of the textual descriptions to relate exactly to the use case diagram and have blue sky flow, alternate flow, and exceptional flow. This was done due to the revisions suggested by Matthias during the feedback session.	Swapnil Bhagat
• 2 (2.1)	• Stakeholders	Added Team Trees and Carbon Fund to stakeholders as we use their services for charity donations. Priority was set to low as these are easily changeable.	Swapnil Bhagat
• 2 (2.2.1)	• Use Case (Diagram)	Added description of actors above the use case diagram to clarify their roles with the application.	Swapnil Bhagat
• 2 (2.2.2)	• Functional Requirements	Re-did all of the functional requirements (about doubling) such that they applied the principles of SMART and overall were more specific than before. Feedback from Matthias was implemented as well (e.g. every use case now has associated functional requirements).	Swapnil Bhagat
• 8	• Current Project Version	Created a table of use cases and their related functional requirements, discussing which of these had been implemented in Phase 2 and which needed to be implemented in Phase 3.	Swapnil Bhagat
• 7 (7.1)	• Functional testing procedures	Reformatted the table to be sorted by the UC column. Also, more functional testing procedures such that every use case had an associated (manual) functional test.	Swapnil Bhagat
• 7 (7.2)	• Quality testing procedures	Created this table to outline which procedures were implemented when testing the quality of the application. The same format as the functional testing table was used.	Swapnil Bhagat

• 6	• UML Class Diagram	Re designed the UML class diagram based on the current state of the application. We now utilize an MVC design pattern. The Model, View and Controller packages are shown separately on this document. Each package only shows intra-package dependencies. This process was especially helpful in identifying redundant pieces of code. It also led to the discovery of repetition in the code that can be solved through abstraction	Kye Oldham
• 6	• UML Class Diagram	Inter-package dependency diagram completed. As whole class diagram is too large to be included in this document, I created a small compact class diagram. The packages are still displayed but, the attributes and methods of each class are omitted	Kye Oldham
• 2 (2.2.4, 2.2.5)	• Stakeholders and requirements (quality requirements, key drivers)	Updated and added more details to the quality requirements and key drivers sections of the design document. These changes reflected the feedback given for phase 1 of the project.	Griffin Baxter
• 7	• Testing procedures	Added the manual tests done to test the functionalities of the application, including their description, results, and criticality.	Darryl Alang
• 10	• Project Plan	Added a Gantt chart for the visualisation of the project plan. Updated the tasks and milestones for phase 1 and 2.	Darryl Alang
• 1 (1.2) • 10	• Business and System context (system context) • Project plan	Updated the system context diagram to include details about systems used by the application. Also updated the executive summary to better reflect the current state of the project and the design document alongside updating the project plan with more details.	Griffin Baxter
• 6	• UML Class Diagram	Added comprehensive explanations for each packages UML class diagram along with the inter-package relationship diagram	Kye Oldham

• 10	• Project Plan	Rewrote paragraph about lockdown restrictions contingency plan. Refactor milestones table into a major milestones table and 3 more tables for the milestones of each phase. Added additional milestones and tasks.	Noah Irving

Phase 3 Changes

Section numbers	**Section titles**	**Change**	**Who changed**
• 5	• Deployment Model	Updated the deployment model of our application. This includes our updated file structure for the persistent database storage.	Griffin Baxter
• 4	• GUI Prototypes	Updated the GUI Prototypes section of the design document. This included removing some features that aren't present in our phase 3 application.	Griffin Baxter
• 7 • 8	• Testing procedures • Current project version	Updated the Testing procedures and current project version of the design document. This included updating the testing procedures and current project version with our advancements made with the project throughout phase 3.	Griffin Baxter
• 6	• UML Class Diagram	Updated the UML class changes to reflect changes made in phase 3 and end of phase 2. E.g. AutoCompleteComboBoxListener class was removed.	Swapnil Bhagat
• 11	• Lessons Learned	Added the lessons that I have learned throughout this semester.	Swapnil Bhagat
• 2.2.3/2.2.4	• Quality Requirements/Key Driver Analysis	Removed legality as a quality requirement as it did not make much sense, especially as a key driver. Updated key driver values – now usability is at the top with the other following closely behind.	Swapnil Bhagat
• 11	• Lessons Learned	Added three lessons learned over the course of the project. Tasks take longer than expected to complete, plan for the worse and ask for help if stuck.	Kye Oldham
• 11	• Lessons Learned	Added three of my key lessons learned over the course of this SENG 202 project with the team.	Griffin Baxter

• 7 • 10 • 11	• Testing procedures • Project plan • Lessons learned	Added testing procedures on new functionalities (mainly CRUD). Added tasks for phase 3 on project plan and gantt chart for visualisation. Written the lessons learned when making the project.	Darryl Alang
• 3 • 11	• Acceptance testing • Lessons learnt	Completely rewrote table for acceptance testing table. Wrote personal section for lessons learnt.	Noah Irving

1. Business and System Context

1.1. Business Context

The services and features provided by our application GreenFlights include the ability to view data about airports, airlines, and aeroplane routes, such as airline names, source and destination airports, distances of flight routes. This data can be searched, sorted, filtered, and viewed through a table and an interactive Google maps interface. Users can upload data to the application and, furthermore, can add new entries to the existing sets of airports, airlines and routes, and update or delete existing records. The application will also be open source, allowing for anyone to modify and even add features to the application. Finally, and most importantly, the application provides information on the carbon emissions generated by individual flight routes. The user can rank these routes by carbon emissions generated (kilograms carbon dioxide) and additionally compare these visually through an intuitive bar chart. The amount of money to donate to CarbonFund.org in order to offset the total carbon emissions of the selected routes is shown to the user. Alternatively, the number of trees to plant to offset the carbon emissions is also shown in a similar way.

Given the public's growing concerns about the impacts of climate change, one of the most prominent contributors to climate change must be highlighted, air travel. Carbon emissions from aircrafts have a significant effect on climate change (Capoccitti, S., Khare, A., & Mildenberger, 2010). Currently, there exists a lack of easily accessible information on the true impacts of these emissions. The vision for this product is to create an easy-to-use application that can be used to track flight route, airline and airport data as well as provide information about carbon emissions produced by chosen routes and ways to counteract these emissions. While GreenFlights is not as complex nor have as many in-depth features of popular current flight tracker products such as FlightRadar24, the team believes that the carbon emission features, while being easy-to-use, open source, and portable, creates a distinct product that is useful in this market on its own.

The target demographic for this application are environmentalists, (amateur) flight enthusiasts and environmentally conscious flight passengers who would benefit from the data provided by this application. Current flight tracking software on the market does not provide data on carbon emissions of flights and aims their product towards flight enthusiasts (Flightradar24, 2020). Existing software that does provide information on flight carbon emissions does not provide a way of tracking large sets of flights. This application capitalises on this gap in the market by creating an easy to use application that can track large sets of flights and provide information on their carbon emissions and ways to counteract these emissions. Most flight tracking applications, being browser-based, also rely on an internet connection to analyse data. GreenFlights, apart from the Google Maps feature, has no such limitation and also has the potential in the mobile market (IOS App Store, Android Play Store).

Due to this gap in the market of flight tracking application relating to the calculations of carbon emissions, whether it be for personal flights or a set of multiple flights from uploaded data sets, this application is worth spending the time to develop.

1.2. Systems Context

The flight tracking application system would be able to be used by multiple types of people for various purposes, as illustrated with its context in Figure 1. Additionally, a service provided to the system is Google Maps, which is used for visualizing routes and airports within the application. This application would appeal to flight passengers trying to find the most environmentally friendly route to get to their destination, using the ranking system on finding the best route. From this, the passenger can upload their own flights to the application for viewing, filtering, and searching for data on. Environmentalists would be appealed to use this application due to its ability to view the carbon emissions of a selected set of routes in order to offset or reduce their emissions, based on either an environmental donation equivalent or a certain number of trees to plant for offsetting CO2. The application would also appeal to flight enthusiasts, in which data sets can be uploaded, modified, filtered, and browsed through for airports, airlines and routes of flights, alongside being able to visualise this information on a map.

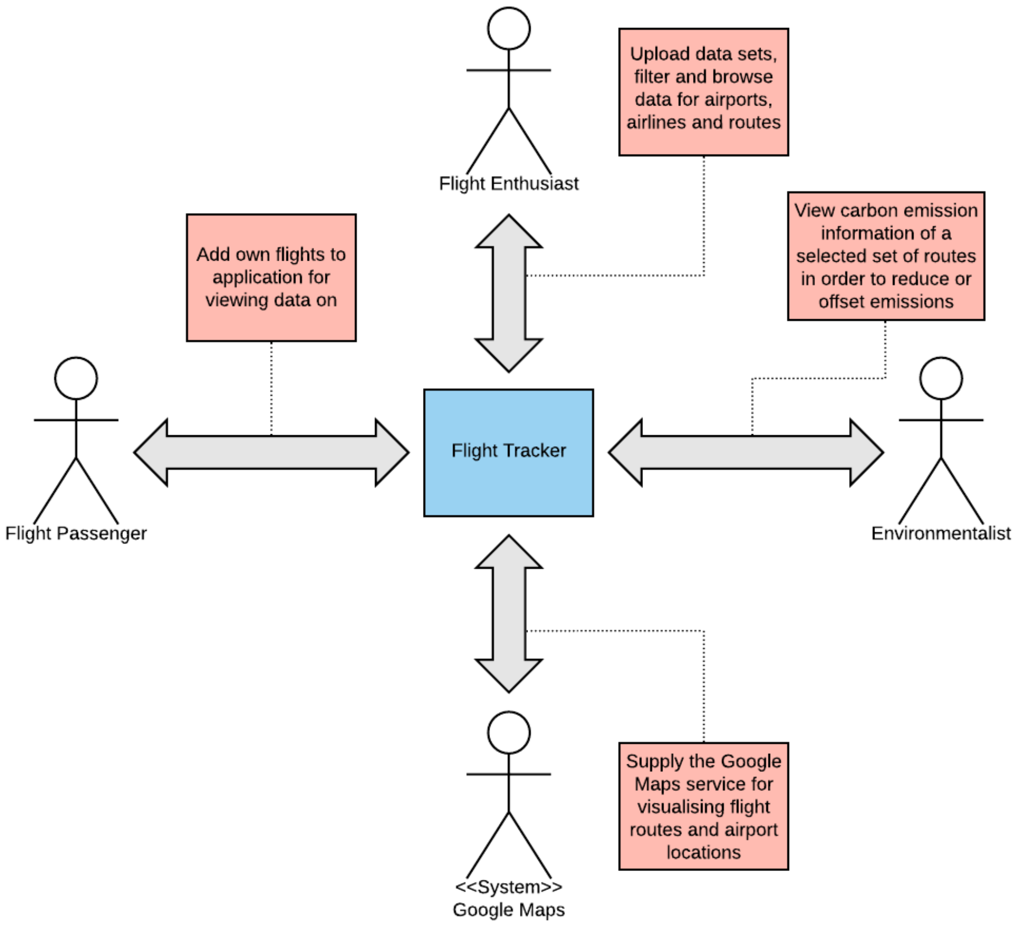
Flight Enthusiast
Upload data sets, filter and browse data for airports, airlines and routes
Add own flights to application for viewing data on
View carbon emission information of a selected set of routes in order to reduce or offset emissions
Flight Passenger
Flight Tracker
Environmentalist
<<System>> Google Maps
Supply the Google Maps service for visualising flight routes and airport locations

Figure 1 – System context diagram of the flight tracking application.

2. Stakeholders and Requirements

2.1. Stakeholders

Due to the large amount of features this application offers, there is an equally large range of possible stakeholders who may impact or be impacted by this project. The priority stakeholders however are casual flight passengers or individuals who have a vested interest in the environment. The secondary stakeholders for the application largely include amateur flight enthusiasts who may need a free lightweight and portable flight tracking tool/database system to aid in their flight management, or a skeletal software to add their own analysis. The full details of the project's stakeholders are listed below in Table 1.

Table 1 – Analysis of stakeholders for the flight tracking application.

ID	Stakeholder (+ description)	Concerns	Priority
SH-1	Flight passengers/travellers	Flight passengers are within the project's target demographic as our application provides an easy way to view their flight's carbon footprint and offset this through environmental donations. It is likely that these passengers may also have interests in seeing routes/airports that emit the most carbon – analysis that the application will provide. Passengers' main concerns lie with an easy-to-use application that provides quick analysis for their flight route.	High
SH-2	Development team (team developing the application)	The development team heavily impacts the project and are heavily impacted by it as it affects our academic transcript and programming experience. Our main concerns lie with code readability, maintainability, and modularity such that the project is continuously upgradable, and any bugs can be fixed with relative ease. Our application should also meet the functional requirements outlined in 2.2.2 as to satisfy our stakeholders.	High
SH-3	SENG202 course staff	The course staff heavily impact the development process through their feedback and giving the development team the facilities to aid in development (e.g. Eng-Git). They also provide the criteria on which the application is marked. Their main concerns are the project's end-user experience, feature quality, and low-level implementation of software design principles (e.g. readability, maintainability).	High

SH-4	Environmentalist	Environmentalists would have a vested interest in this application due to it providing plane carbon emission calculations and analysis based on routes and airports. Their main concern would be the accuracy of the data provided such that it can be used in a factual way.	High
SH-5	Flight enthusiast (people who are interested in flights, routes, aircraft)	Amateur flight enthusiasts may require a lightweight, portable flight tracker that provides a way to upload and analyse (calculating distances, ranking, and filtering) custom flight data. The application is open source and so enthusiasts may add their own features – in this case, the code provided should be readable and modular.	Medium
SH-6	Google Maps	The application relies on the Google Maps Application Programming Interface (API) to provide a visualisation on the uploaded route and airport data. The concern for the developers/users is that Google Maps is a reliable software such that any changes made by Google will scantly affect the application.	Medium
SH-7	Flight Plan Database	The application relies on the Flight Plan Database API (flightplandatabase.com) to provide a detailed route from the source and destination provided by the user. Like Google, the concerns lie with the reliability of the service as any changes Flight Plan Database makes could affect the application.	Medium
SH-8	Carbon emissions researchers (carbonindepenedent.org)	The project's flight carbon emissions calculations are derived from researchers and so they are stakeholders who impact the system. The core functionality of the app does not however depend on this (searching, filtering, maps) and carbon emissions research, as long as it is reliable, is interchangeable, hence the medium priority.	Medium
SH-9	OpenFlights (openflights.org/data)	Raw data tables in the application use the format provided by OpenFlights and so users uploading data must also adhere to this format. This can be modified/swapped out with relative ease, hence the medium priority.	Medium
SH-10	Environment advocacy groups	Environmentalist groups may use this application for research purposes and an initial assessment for flight carbon emissions. However, they are more likely to conduct studies of their own and so have low priority.	Low

SH-11	Development team's immediate social circle (family and friends)	The development team's immediate social circle impacts the project indirectly through impacting the developers' schedule, productivity, and availability to work. This is largely a low impact so therefore has a low priority.	Low
SH-12	Team Trees (teamtrees.org)	The application allows the user to plant trees based on the carbon emissions for the selected routes. Team Trees is the most popular platform to donate trees-equivalent-money and will be used for the application. The major concern is their continued/unchanging operation, but many other similar platforms exist that we could support instead so priority is low.	Low
SH-13	Carbon Fund (carbonfund.org)	Carbon Fund was chosen as the organisation to donate to due to their ZeroCarbon initiative and them being carbon emission focused. Concerns are their continued/unchanging operation, but many other similar organisations exist that we could support instead so priority is low.	Low

2.2. Requirements

2.2.1 Use Case Diagram

Figure 2 showcases the main use cases and actors of the flight tracking application. Additionally, the major use cases are textualized with their descriptions, pre-conditions, flows and post-conditions in Table 2.

Actors

Flight enthusiast (SH-5): An individual who informs themself on all-things aeroplane and so would likely use this application as a portable and easy-to-use visualiser/data viewer.

Flight passenger (SH-1): A regular curious consumer looking to use this application to upload their own flight routes in order to analyse their carbon footprint. The easy-to-use nature of the application also makes it possible for normal consumers to look further into flight data analysis.

Environmentalist (SH-4): An individual looking to analyse the carbon footprint of a range of flight routes or airports. The data obtained from the application may be used for public information or personal projects.

Google Maps System (SH-6): Map visualisation is a core component of the application and as such an established map provider like Google Maps is required. Users will be able to visualise routes and airports.

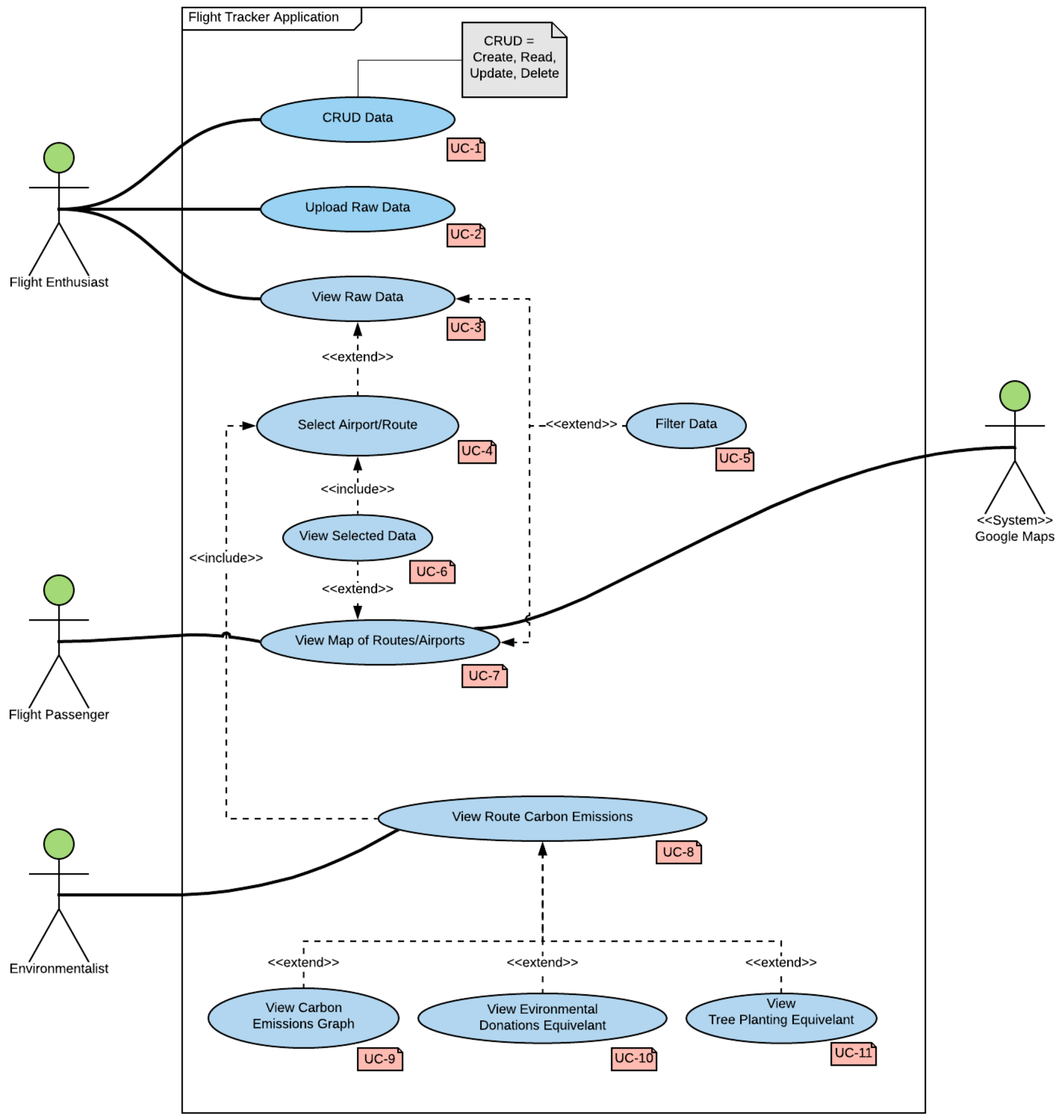

Figure 2 – Use case diagram of the flight tracking application.

Table 2 – Textual description of use cases.

ID	Use case	Actor(s)	Pre-condition(s)	Description (flows)
UC-1	CRUD (Create, Read, Update, Delete) data	Flight Enthusiast (SH-5)	The user must be on the ‘Airports’, ‘Airlines’, ‘Routes’, or ‘Flight Paths’ tab of the application	Blue sky scenario – Read, Update, Delete: 1. User selects a row from the raw data table and chooses to read, update, or delete this data record 2. *On read*: additional information of this data record is shown 3. *On update*: user modifies data values and commits changes 4. *On delete:* user removes record from table Post-conditions: *On read*: no changes are made to the record, additional information is shown to the user *On update*: record is shown to the user with modified values *On delete*: the record is no longer visible in the data table Alternate flow - Create: 1. User chooses to create a new data record 2. User enters their chosen attributes and commits changes Post-condition: Table has one extra record containing user submitted data Exceptional flow: ✗ *On update*: User enters invalid values ✗ *On create*: User enters invalid values Post-condition: *On update*: error message is shown to user and data records remain unmodified *On create*: error message is shown to user, who can subsequently change invalid values

UC-2	Upload raw data	Flight Enthusiast (SH-5)	The user must be on the 'Airports', 'Airlines', 'Routes', or 'Flight Paths' tab of the application	<u>Blue sky scenario:</u> 1. User choose to import Airport/Airline/Route/Flight Path Data 2. User selects a .dat, .txt, or .csv file with correct (OpenFlights) format and selects 'Upload' <u>Post-condition:</u> The uploaded data is displayed accurately in the corresponding data's raw data table <u>Exceptional flow:</u> × User uploads invalid file type or data within files is invalid (e.g. wrong format). <u>Post-condition:</u> Error message containing exception is shown to the user. Lines that are valid in the file are loaded into the data type's raw data table; invalid lines are skipped (no data is loaded if invalid file type)
UC-3	View raw data	Flight Enthusiast (SH-5)	The user has data loaded onto the application	<u>Blue sky scenario:</u> 1. User uploads data in the required format 2. User navigates to the tab where data has been loaded <u>Post-condition:</u> A raw data table showing a subset of the columns in the uploaded data is shown to the user.
UC-4	Select airport(s)/route(s)	Flight Enthusiast (SH-5), Environmentalist (SH-4)	The user is on either the 'Airports' or 'Routes' tab and has data loaded	<u>Blue sky scenario:</u> 1. User selects one or more rows <u>Post-condition:</u> Route/airport is selected and can therefore be viewed in the 'Map' and 'Emissions' tab <u>Exceptional flow:</u> × *On route:* selected route has missing source/destination airport fields × *On airport:* Selected airport has no associated routes or is missing co-ordinates

				<u>Post-condition:</u> Error is shown to the user and the route/airport is not selected
UC-5	Filter data	Flight Enthusiast (SH-5), Environmentalist (SH-4)	The user is on the ‘Airlines’, ‘Airports’, ‘Routes’, or ‘Map’ tab of the application	<u>Blue sky scenario:</u> 1. User selects filters from drop-down menu, sliders, or search bar <u>Post-condition:</u> Raw data table shows filtered results (zero or more rows) <u>Alternate flow:</u> 1. User clicks on column they wish to sort by <u>Post-condition:</u> Raw data table rows are ordered alphanumerically by the selected column
UC-6	View selected data [on map/emissions tab]	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1) Google Maps (SH-6)	The user has selected routes or airports and is currently on the ‘Maps’ or ‘Emissions’ tab	<u>Blue sky scenario:</u> *2. On maps*: user chooses to view selected routes or view selected airports on the map (not both) 3. *On emissions*: user chooses to load selected routes <u>Post-conditions:</u> *On maps:* selected routes/airports are shown on the Google Map as routes/points *On emissions*: selected routes are shown in the emissions data record table alongside the distance and emissions calculation for each route <u>Exceptional flow:</u> × *On emissions*: invalid coordinates or a calculation resulting in an invalid number are shown as errors to the user and are not shown in the table
UC-7	View (filtered) map of routes/airports	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1), Google Maps (SH-6)	The user has loaded in airport (and route) raw data (or default data)	<u>Blue sky scenario:</u> 1. User selects ‘Map’ tab 2. User chooses (with or without filters) to display routes or airports. <u>Post-condition:</u> If no filters are used, all up to a set limit (to avoid clutter) routes/airport are shown,

				otherwise only data adhering to the filters are shown <u>Exceptional flow:</u> × The user is not connected to the internet <u>Post-condition:</u> User is shown warning when loading application about the lack of internet connection. Every element of the application except the map view still works
UC-8	View route/airport carbon emissions and distance (for route)	Environmentalist (SH-4), Flight Passenger (SH-1)	User must have routes/airports selected from the 'Routes' or 'Airports' tab and is now on the 'Emissions' tab	<u>Blue sky scenario:</u> 1. User chooses to load selected routes/airports 2. *For routes:* for each route selected, distance between endpoints and associated carbon emissions are calculated 3. *For airports*: for each airport, the carbon emissions based on the flights coming from this airport are calculated <u>Post-condition:</u> For each route/airport selected, its distance (if route) and amount of carbon dioxide emitted is shown. A sum of carbon dioxide emitted is shown as well <u>Exceptional flow:</u> × The routes/airports selected have invalid values for calculations, so error is thrown when calculating <u>Post-conditions:</u> User is shown error message and those whose data is invalid have no distance/carbon calculations shown
UC-9	View Carbon Emissions Graph	Environmentalist (SH-4), Flight Passenger (SH-1)	User has valid airports/routes selected and are on the 'Emissions' tab	<u>Blue sky scenario:</u> 1. User chooses to view graph containing a comparison of carbon emissions of the routes/airports selected 2. Carbon emissions for each route/airport is evaluated 3. Graph is generated comparing these values

				<u>Post-conditions:</u> A graph representation of route/airport carbon emissions is presented to the user <u>Exceptional flow:</u> ✗ The routes/airports selected have invalid values for calculations, so error is thrown when calculating <u>Post-conditions:</u> User is shown error that one or more entity has invalid values and graph is not shown
UC-10	View Environmental Donations Equivalent	Environmentalist (SH-4), Flight Passenger (SH-1)	User is on the 'Emissions' tab	<u>Blue sky scenario:</u> 1. User chooses the option to view environmental donation equivalent 2. Amount to donate relative to carbon emissions calculated <u>Post-conditions:</u> The amount to donate to a charity is shown to the user in dollars as well as a link of the charity to donate to (Carbon Fund) <u>Exceptional flow:</u> ✗ User attempts to calculate donation equivalent with no routes selected <u>Post-conditions:</u> Error is shown to the user, prompting them to choose some routes first
UC-11	View Tree Planting Equivalent	Environmentalist (SH-4), Flight Passenger (SH-1)	User is on the 'Emissions' tab	<u>Blue sky scenario:</u> 1. User chooses option to view planting trees equivalent 2. Number of trees to plant to offset routes' carbon emissions is calculated <u>Post-conditions:</u> The number of trees to plant is shown to the user as well as a link to teamtrees.org – who will plant those trees for you based on your donation

2.2.2 Functional Requirements

The functional requirements of the flight tracking application are shown in Table 3, alongside their accompanying stakeholders, priority and use cases.

Table 3 – Functional requirements analysis of the application.

ID	Description	Stakeholders	Priority	Use case(s)	Implemented
FR-1	A user must be able to create a new data record for an airline/airport/route on this data type's tab by clicking the appropriate 'add data record' button.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 2
FR-2	On invalid input when adding a new record an error message saying which input is invalid is shown to the user.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 2
FR-3	A user must be able to select an airline/airport/route and view additional information about it.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 3
FR-4	A user must be able to select an airline/airport/route in the raw data table and modify its attributes.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 3
FR-5	On invalid input when modifying data, an error is shown to the user of which input line is invalid.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 3
FR-6	A user must be able to select an airline/airport/route and delete it from the raw data table.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-1	Phase 2
FR-7	Using an 'upload' button a user must be able to upload (from their own file system) a .dat/.txt/.csv of raw airport, airline, or route data in the format	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	High	UC-2	Phase 2

	specified for each data type by *OpenFlights.org*. When a new file is uploaded, a new data set is created containing this data.				
FR-8	When uploading data, it is not possible to choose file type other than those suffixed with .dat/.txt/.csv.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	High	UC-2	Phase 2
FR-9	There must be a validity check for all data components when new data is uploaded. This validity check is done by comparing each column of the input to a regular expression. If the data does not pass the validity check an error is shown to the user and the line is skipped.	Environmentalist (SH-4) Flight Enthusiast (SH-5) Development Team (SH-2)	High	UC-2	Phase 2
FR-10	If duplicate (valid) lines are present in the upload file, then only one of the lines are added to the raw data table – hence there are no duplicates in the table.	Development Team (SH-2)	Low	UC-2	✗
FR-11	The user must be able to view a subset of the uploaded data (airline, airport, route) in a table format with each column relating to a distinct comma-separated-value in the uploaded file.	Environmentalist (SH-4) Flight Enthusiast (SH-5)	High	UC-3	Phase 2
FR-12	A user must be able to select an airport or route on its respective raw data table through a checkbox column.	Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-4	Phase 2
FR-13	Using a series of drop-down menus and/or sliders, a user must be able to filter airports by country and city.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-5	Phase 2
FR-14	Using a series of drop-down menus and/or	Flight Passengers (SH- 1)	Medium	UC-5	Phase 2

	sliders, a user must be able to filter routes by airline, departure/destination country, number of stops, and plane type.	Environmentalist (SH-4) Flight Enthusiast (SH-5)			
FR-15	Using a series of drop-down menus and/or sliders, a user must be able to filter airlines by country.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-5	Phase 2
FR-16	Using a search bar, the user must be able to search a raw data table (airlines, airports, routes) for rows containing attributes that contain the searched letters.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-5	Phase 2
FR-17	On each of the routes, airlines, and airports raw data tables, the user must be able to sort alphanumerically by each column of the table by clicking on the head of the column.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-5	Phase 2
FR-18	The user must be able to select/deselect multiple routes/airports in the routes/airports table using a checkbox column.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast	High	UC-6	Phase 2
FR-19	After having selected airport(s) on the airport raw data table, the user must be able to visualise these *selected* airports as points on the map.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5) Google Maps (SH-6)	Medium	UC-6	Phase 2
FR-20	After having selected route(s) on the routes raw data table, the user must be able to visualise these *selected* routes as points and lines on the map.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5) Google Maps (SH-6)	High	UC-6	Phase 2

FR-21	After having selected route(s) on the routes raw data table, the user must be able to view the emissions of each route on the emissions tab.	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)	High	UC-6	Phase 2
FR-22	A total carbon emission (carbon dioxide kilograms) calculation must be shown at the top of the page to the user based on the routes selected.	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)	Medium	UC-6	Phase 2
FR-23	The user must be able to view routes as points connected by lines on the map. Filters relating to route attributes may be applied via drop-down menus and only those routes that adhere to the filters are then shown. Filters include airline, airport, and plane type. Note: this is different from FR-14 as it is not only *selected* items.	Flight Passengers (SH- 1) Flight Enthusiast (SH-5)	High	UC-7	Phase 2
FR-24	The user must be able to view airports as points on the map. A filter for the airport country may be applied via drop-down menus and only those airports that adhere to this filter are then shown.	Flight Passengers (SH- 1) Flight Enthusiast (SH-5)	High	UC-7	Phase 2
FR-25	A limit of the number of airports/routes shown on the map must be in place to avoid clutter and increase the readability of the data.	Development Team (SH-2)	Low	UC-7	Phase 2
FR-26	The user must be able to view the distance in kilometres of a route for each route selected from the routes tab in the emissions tab.	Flight Passengers (SH- 1) Environmentalist (SH-4) Flight Enthusiast (SH-5)	Medium	UC-8	Phase 2
FR-27	The user must be able to view the carbon dioxide (per person in kilograms)	Flight Passengers (SH- 1)	High	UC-8	Phase 2

	emitted from a flight. Calculations are derived from carbonindependant.org with assumptions.	Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)			
FR-28	The user must be able to view the carbon emissions for each selected airport based on the number of outgoing routes (sum of all routes’ carbon emissions).	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)	Medium	UC-8	✗
FR-29	The user must be able to view a bar graph comparing the carbon dioxide emissions (kg per person) selected from the routes tab in the emissions tab.	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)	Low	UC-9	Phase 3
FR-30	The user must be able to view a bar graph comparing the carbon dioxide emissions of airports (based on the sum of the carbon dioxide emissions of the outgoing routes).	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10)	Low	UC-9	✗
FR-31	The user must be able to view how much, in New Zealand Dollars, money is required to offset the carbon emissions previously calculated. The chosen organisation to donate to is Carbon Fund. The carbon dioxide to offset amount in dollars is calculated using mercycorps.org research.	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10) Carbon Fund (SH-13) Carbon Emissions Researchers (SH-8)	High	UC-10	Phase 2
FR-32	The user must be able to view how many trees must be planted to offset the previously calculated carbon emissions. Teamtrees.org is the charity chosen to donate to and a link taking the user to their website is shown.	Flight Passengers (SH- 1) Environmentalist (SH-4) Environmental Advocacy Groups (SH-10) Carbon Fund (SH-13) Team Trees (SH-12)	High	UC-11	Phase 2

FR-33	The user must be able to switch between datasets (different sets of uploaded data) using a drop-down menu on the 'Airports', 'Airlines', 'Routes', 'Flight Paths' tabs. The corresponding raw data tables are then updated with the newly selected data.	Flight Enthusiast (SH-5)	Medium	N/A	Phase 2
FR-34	The user must be able to store their data, closes the application. Re-open the application and still have their stored data visible.	Flight Enthusiast (SH-5)	Medium	N/A	Phase 2

2.2.3 Quality Requirements

The quality requirements of the flight tracking application are shown in Table 4, alongside their accompanying stakeholders and priority.

Table 4 – Quality requirements analysis of the application.

ID	Description	Stakeholder	Priority
QR-1	Performance and responsiveness. The application must be able to perform all the required tasks in a speedy manner. This means that uploading data to the application should not take more than 10 seconds for a large set including 100,000 lines. Additionally, clicking on menus and tabs in the application should be perceivably instantaneous (less than one second).	Flight passengers (SH-1), Flight Enthusiasts (SH-5), Environmentalists (SH-4)	High
QR-2	Maintainability. The application must be designed with principles such as low coupling, high cohesion, to have a modular codebase. This allows the ability for the development team of the application (and additionally, other developers or flight enthusiasts as it will be an open-source application) to modify, fix and add new features or updates to the application, without or with minimal aid of the development team.	Development team (SH-2) Flight Enthusiasts (SH-5)	High
QR-3	Usability. The GUI should be simple enough for the general users to easily learn how to use the application, without excess reading of documentation. This means adding instructions throughout the program and making navigation intuitive for a non-technical person. The application should be able to detect user input errors and provide warning messages.	Flight passengers (SH-1)	High

QR-4	Compatibility. The application should be able to be used on different operating systems aside from the standard Mint Linux OS installed on the UC lab computers.	Flight passengers (SH-1), Flight Enthusiasts (SH-5), Environmentalists (SH-4)	Medium
QR-5	Reliability. The application should be able to provide reliable and reproducible calculations of flight route carbon emissions and their respective equivalents to offsets based on amount of donation and trees.	Development team (SH-2), Flight passengers (SH-1), Environmentalists (SH-4)	High
QR-6	Suitability. The application should meet all stated requirements and use cases, wherein it can perform all the high priority specified tasks and includes all the specified features.	Development team (SH-2), Flight passengers (SH-1), Flight Enthusiasts (SH-5), Environmentalists (SH-4)	Medium

2.2.4 Key Driver Analysis

The key driver analysis in Table 5 shows which stakeholder and quality requirements, based on the high/medium priority stakeholders, should be prioritised. Quality requirements listed above were used with these weighted stakeholders to calculate which requirement/s would be the key driver/s for this application. The requirement with the highest weighted average was Usability (QR-3) and so will be focused on in more depth throughout the project. However, most other requirements fell in a 2-3-point range which leads the team to believe that the app should be well rounded due to the broad demographic to which the team offers the software.

Table 5 – Key driver analysis to determine the priority of main quality requirements.

Stakeholders	**Weight**	**Performance**	**Maintainability**	**Usability**	**Compatibility**	**Reliability**	**Suitability**
Flight passengers	0.25	20	1.25	22.5	7.5	20	17.5
Development team	0.25	15	20	18.75	15	17.5	17.5
SENG202 course staff	0.15	9	13.5	12	7.5	9.75	11.25
Environmentalists	0.20	9	1	12	2	14	12

Flight enthusiasts	0.15	7.5	7.5	10.5	4.5	12	12
Total	**1**	**12.1**	**8.65**	**15.15**	**7.3**	**14.65**	**14.05**

3. Acceptance Tests

Table 6 shows the black-box tests to be done by specific end users to test the functionalities of the application. It describes the test scenario, who is responsible for testing it and how to determine whether the application passed or failed the test. Also stated is the criticality of a certain test, with high criticality being closely linked to the core functionality of the application.

Table 6 – The black-box acceptance testing for the flight tracking application.

ID	Description	Responsibility	Acceptance Criteria	Criticality	Use Case
AT-1	*Given* that the user is viewing the airline/airport/route/flight path tab. *When* the user double clicks a record in the table. *Then* a new window will be launched showing the user the record that was clicked.	Flight enthusiast (SH-5)	When the record is clicked the user should be shown the full contents of that record.	Medium	UC-1
AT-2	*Given* the user has double clicked a record in airline/airport/route/flight path tab and the contents of the record is shown in a new window. *When* an attribute of the record is edited, a data set is chosen and confirm is clicked. *Then* if the edited record is valid the record will be updated in the database and the record should be updated in the table.	Flight enthusiast (SH-5)	When a record is edited such that it is valid and confirm is clicked the record should be updated in the database and in the table.	Medium	UC-1
AT-3	*Given* the user has double clicked a record in airline/airport/route/flight path tab and the contents of the record is shown in a new window. *When* an attribute of the record is edited, a data set is chosen and confirm is clicked. *Then* if the edited record is invalid the user would be	Flight enthusiast (SH-5)	When a record is edited such that it is invalid and confirm is clicked the user will be notified as to which field is invalid through an error message.	Medium	UC-1

	notified to which field was invalid in the edited record.				
AT-4	*Given* that the user is viewing the airline/airport/route/flight path tab. *When* the user clicks one or more records and click the “x”. *Then* the records that were selected will be deleted from the database and removed from the table.	Flight enthusiast (SH-5)	When the user has chosen one or more records the “x” is clicked the records should be deleted from the database and from the table.	Medium	UC-1
AT-5	*Given* that the user is viewing the airline/airport/route/flight path tab. *When* ‘+’ is clicked. *Then* a new window appears showing empty attributes to fill for a new record of the tabs data type.	Flight enthusiast (SH-5)	When “+” has been clicked a new window should launch showing empty attributes to fill of the tabs data type.	Medium	UC-1
AT-6	*Given* that the user is viewing the airline/airport/route/flight path tab and has clicked ‘+’. *When* the user inputs data for the attributes of the new record, chooses a data type and clicks confirm. *Then* if the attributes are valid the new record will be added to the table and the database will be updated with the new record.	Flight enthusiast (SH-5)	When the input data the user gave is valid for all the attributes of the new record and a data set is chosen the new record should be added to the table and the database.	Medium	UC-1
AT-7	*Given* that the user is viewing the airline/airport/route/flight path tab and has clicked ‘+’. *When* the user inputs data for the attributes of the new record, chooses a data type and clicks confirm. *Then* if an attribute is invalid the user is notified of the invalid attribute.	Flight enthusiast (SH-5)	When at least one of the attributes is invalid then the user should be notified of the invalid record.	Medium	UC-1
AT-8	*Given* that the user is viewing the airline/airport/route/flight path tab.	Flight Enthusiast (SH-5), Environmentalist (SH-4),	When the user clicks ‘Upload Data’ they should be prompted with a new window asking	High	UC-2

	When the user clicks 'Upload data'. *Then* a new window will open asking the name of the new data set and the file that's being uploaded.	Flight Passenger (SH-1)	the user to input a name of the data set and choose a file.		
AT-9	*Given* that the user is viewing the airline/airport/route/flight path tab and the user has clicked 'Upload data'. *When* the user inputs a name of the new data set, chooses a file and clicks confirm. *Then* the valid records are uploaded to the database and added to the table and the users is shown a prompt of all the invalid records that were skipped when uploading the file.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user clicks confirm the valid records in the file should be uploaded to the database and added to the table and the user should be prompted of all the invalid records that were skipped.	High	UC-2
AT-10	*Given* the user has opened the application. *When* the user navigates to airline/airport/route/flight path tab. *Then* the records of the data type of that tab in the database will be shown in the table.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user navigates to one of airline/airport/rout e/flight path tabs the table should show all the records that are in the database.	High	UC-3
AT-11	*Given* the user is viewing the airports or routes tab. *When* a record is selected with the tick-box. *Then* the record is added to the selected data type's table in the database if valid for selecting.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user selects selectable records, they should be added to the respective data type's select table in the database.	High	UC-4
AT-12	*Given the user is viewing the airports or routes tab. When a record is selected with the tick-box and there is no associated airport for the route's source or destination.Then* an error is shown to the user and the checkbox is unchecked.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	An error is shown to the user saying that there are no associated airports for that route	Low	UC-4
AT-13	*Given* the user is viewing the airline/airport/route/flight path tab.	Flight Enthusiast (SH-5),	When the user changes the filter, the application	Medium	UC-5

	When the user changes a filter. *Then* the table will show the records in the database that match the current filters.	Environmentalist (SH-4)	should update to table to show only the records in the database that match the filter.		
AT-14	*Given* the user is viewing the map tab. *When* the user clicks “view routes”. *Then* the selected routes are shown on the map.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user clicks “view routes” the selected routes should be shown on the map.	High	UC-6
AT-15	*Given* the user is viewing the map tab. *When* the user clicks “view airports”. *Then* the selected airports are shown on the map.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user clicks “view airports” the selected airports should be shown on the map.	High	UC-6
AT-16	*Given* the user is viewing the emissions tab. *When* the user clicks “load selected routes”. *Then* the selected routes are shown and their emissions and distances as well as the environmental donation equivalent and tree planting equivalent to all the routes.	Environmentalist (SH-4)	When the user clicks “load selected routes” the selected routes should be shown and their carbon emissions and distances as well as the environmental donation equivalent and tree planting equivalent to all the routes.	High	UC-6/ UC-8/ UC-10/ UC-11
AT-17	*Given* the user is viewing the map tab. *When* a filter is changed and “apply filter” is clicked. *Then* all routes/airports that matched the filter should be shown on the map.	Flight Enthusiast (SH-5), Environmentalist (SH-4), Flight Passenger (SH-1)	When the user clicks “apply filter”, all the routes/airports that match that filter are shown on the map.	High	UC-7
AT-18	*Given* the user is on the emissions tab and “load selected routes” has been clicked. *When* the user clicks “Contributions Graph”. *Then* a new window is shown with a graph of the carbon emissions of the selected routes.	Environmentalist (SH-4),	When the user clicks “Contributions Graph” a new window should appear showing a graph of the selected routes’ carbon emissions.	Low	UC-9

4. GUI Prototypes

The lo-fi GUI prototypes of the flight tracking interface, GreenFlights, are outlined in this section. The GUI interfaces showcase the design language of the flight tracker, such as the desire to create an easy-to-use application. Prototypes showing all tabs of the flight tracking application are shown alongside the use cases of the application. For example, as climate change and the environment are the main, distinct aspects of the flight tracking application, information on carbon emissions of certain routes based on the distance is shown.

The home tab shown in Figure 3 is a simple, minimalistic interface showing everything available in the application at once. This includes all of the tabs of GreenFlights at the top. Additionally, the home tab features buttons for viewing the user manual, finding out how the carbon emission calculations are performed and a button to exit.

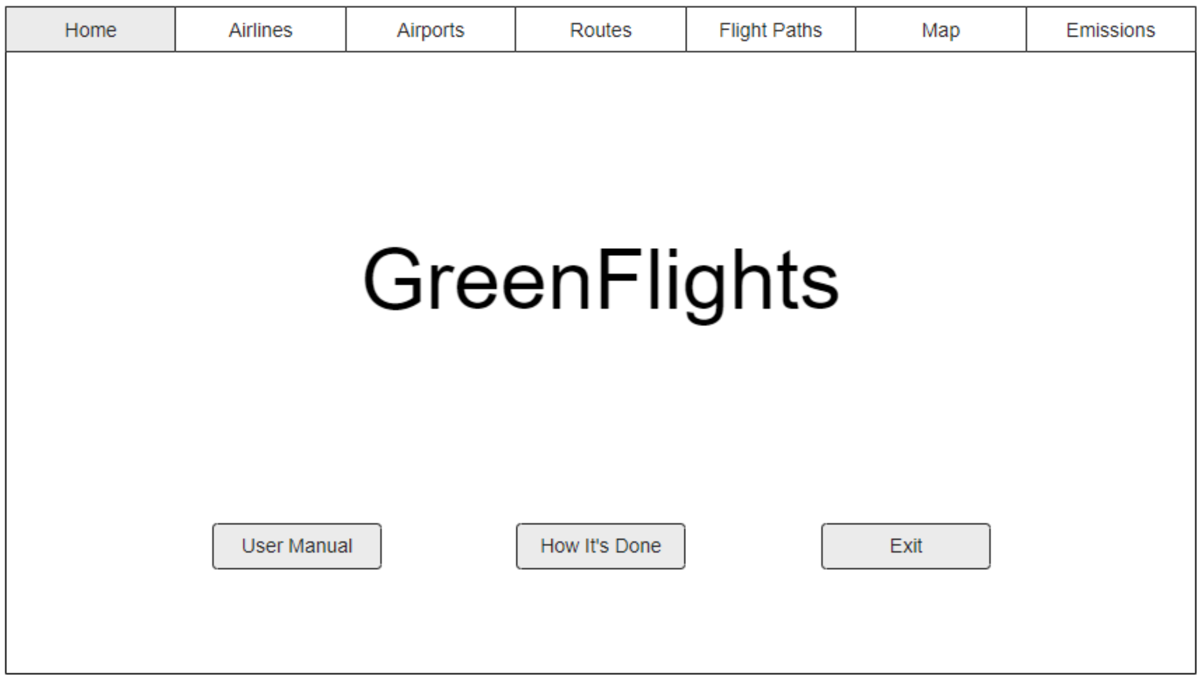

Figure 3 – GUI prototype for the home screen of the flight tracking application.

The airlines tab in Figure 4 shows all the various airlines of the application. This can be filtered by country of operation and sorted by either airline or country. This tab features a search box for finding specific airlines and airline data can also be manually uploaded, as shown in Figure 4. In this tab, and the future data table tabs, rows of data can be individually modified by adding, deleting and editing rows with information.

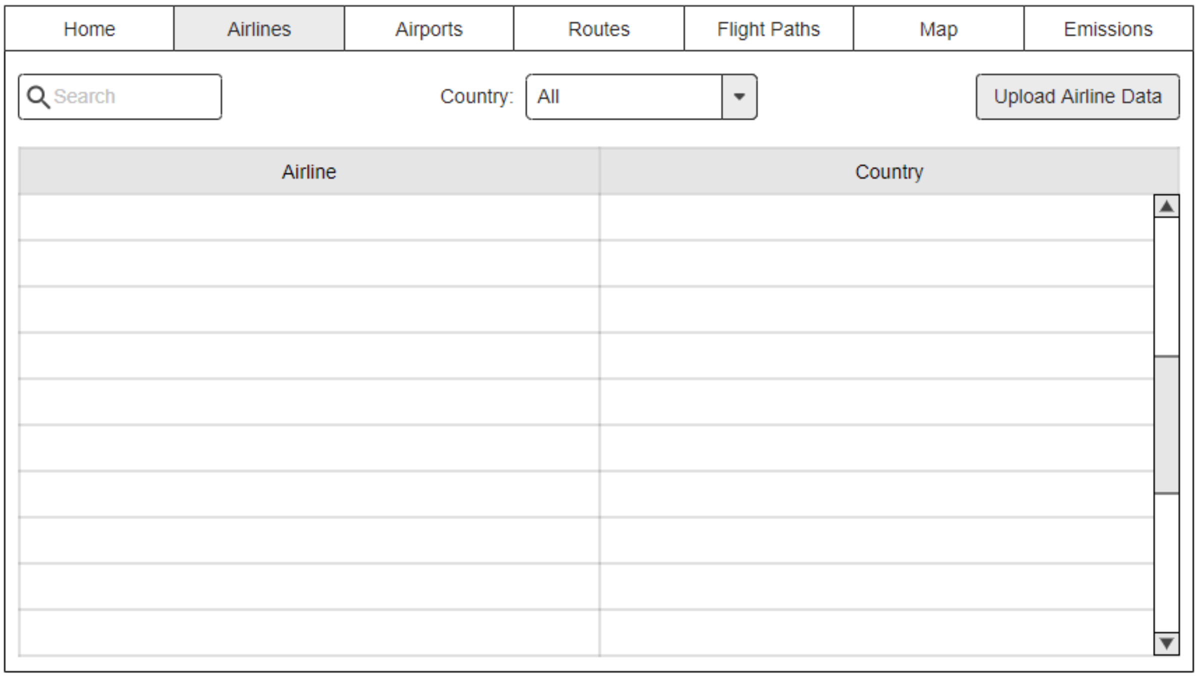

Figure 4 – GUI prototype for the airlines screen of the flight tracking application.

In the airport tab shown in Figure 5, airlines are shown alongside their accompanying city, country, and coordinates. Once again, filters can be put in place for city and country, a search box is supplied and a button for uploading airport data.

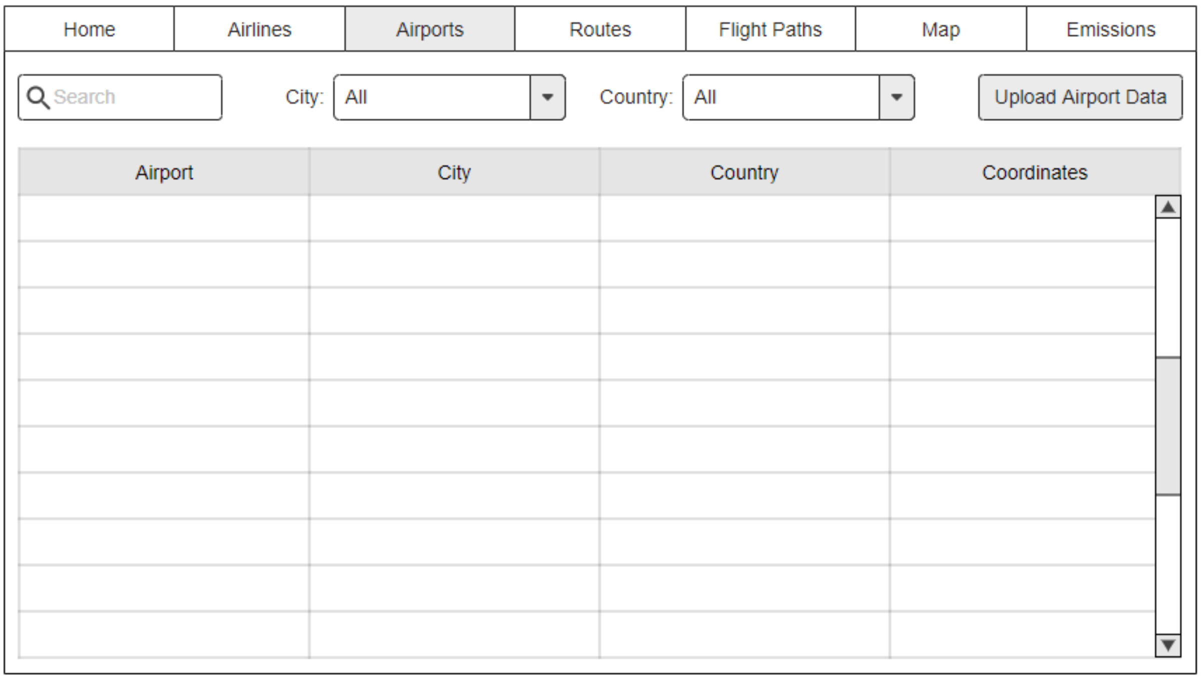

Figure 5 – GUI prototype for the airports screen of the flight tracking application.

The routes tab in Figure 6 shows the user various information such as a route's airline, airports, stops and plane type. The filters section on this tab is expanded from previous sections, in which five different filters can be applied at the same time.

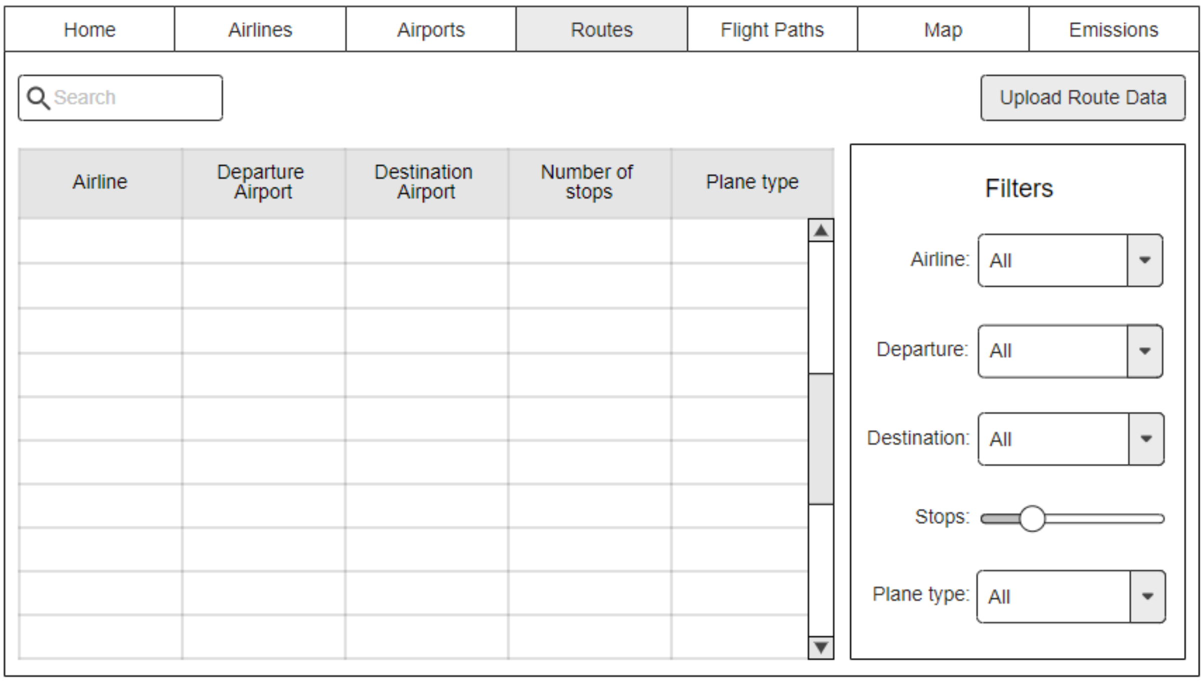

Figure 6 – GUI prototype for the routes screen of the flight tracking application.

The flight paths tab in Figure 7 shows the user various information such as a flight path's type, ID, altitude, latitude and longitude. As in common with the previous tabs with rows of data, there is a search bar and a button to upload a file of flight path data.

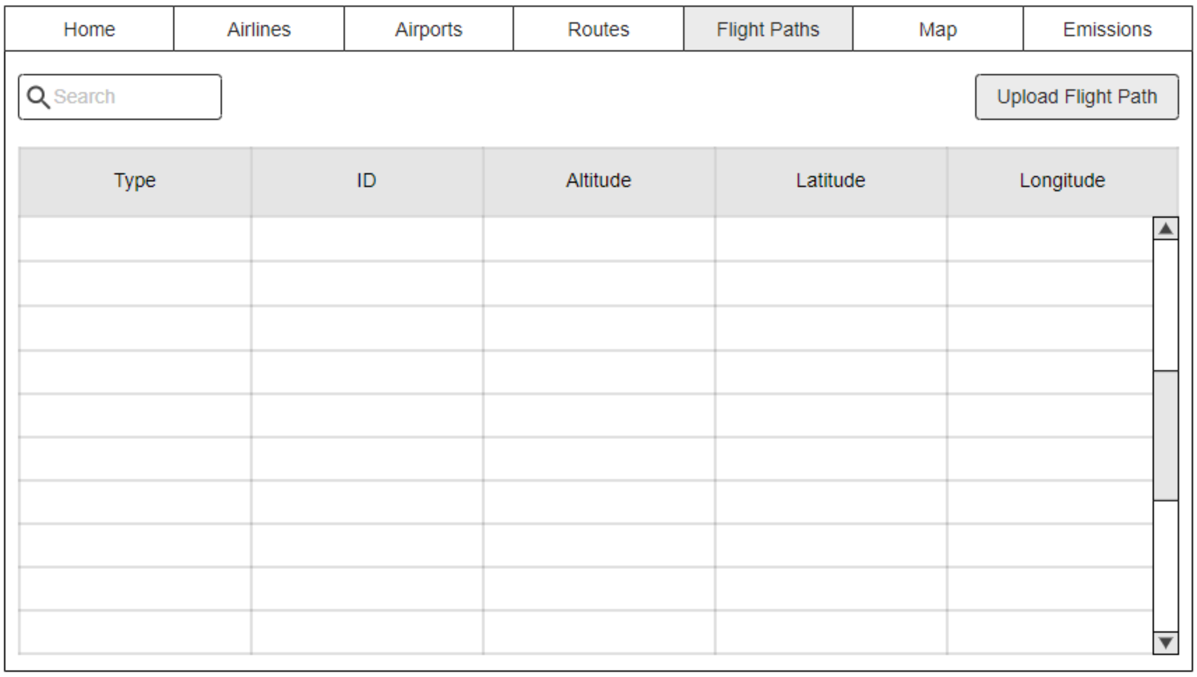

Figure 7 – GUI prototype for the flight paths screen of the flight tracking application.

In the map tab shown in Figure 8, routes can be visualised based on the currently loaded airport and route data and makes use of the Google Maps API.

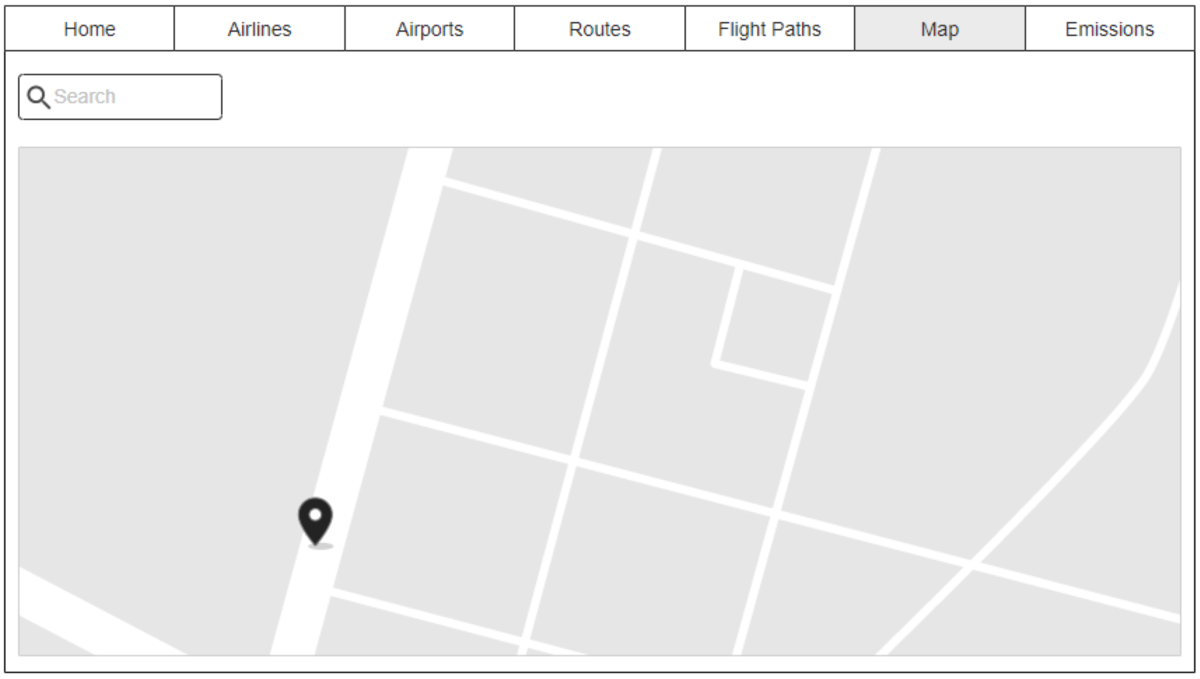

Figure 8 – GUI prototype for the map screen of the flight tracking application.

The emissions tab shown in Figure 9 showcases the unique use cases for the flight tracking application, in terms of the environmental impact of flights. Specific routes and journeys can be added from the current route data to calculate emissions. Using this data, calculations can be made to determine ways to offset emissions, such as through donations or tree planting equivalents.

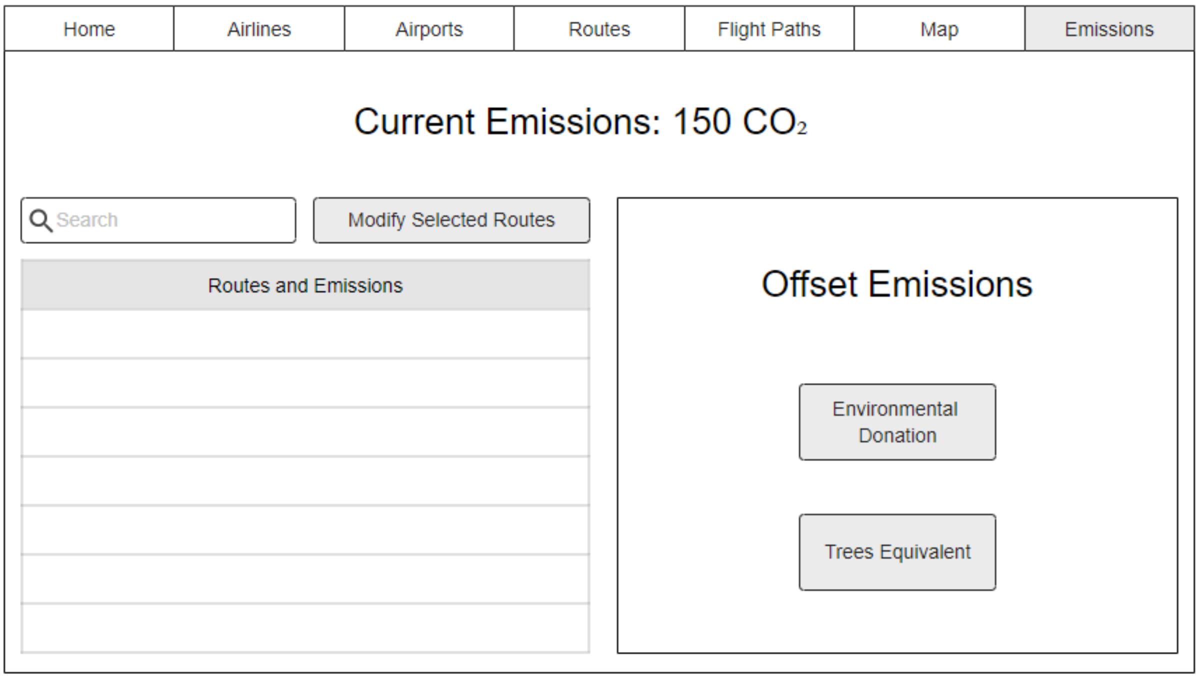

Figure 9 – GUI prototype for the emissions screen of the flight tracking application.

5. Deployment Model

The flight tracking application, GreenFlights, will be run and primarily hosted on a local PC. The PC will run a Linux operating system and have the Java Runtime Environment 11 installed (JRE 11). This PC deploys an executable Java program packaged in a JAR file. The data files that will be used in the application will also be stored on this local machine. For example, the database is stored in the same directory as the JAR file, but in a databases folder. This is done so that the database is stored persistently, and modifications made in the program are saved without the need to re-enter information on subsequent launches of the application. SQLite is utilised to generate and edit this database from the application, as a relatively easy way to manage data, with the allowing of multiple datasets in the application. The application makes use of the Google Maps API to show visuals to the user on a map of chosen and filtered routes and airports. This API is shown as an external node with an HTTPS connection to the local PC, in Figure 10.

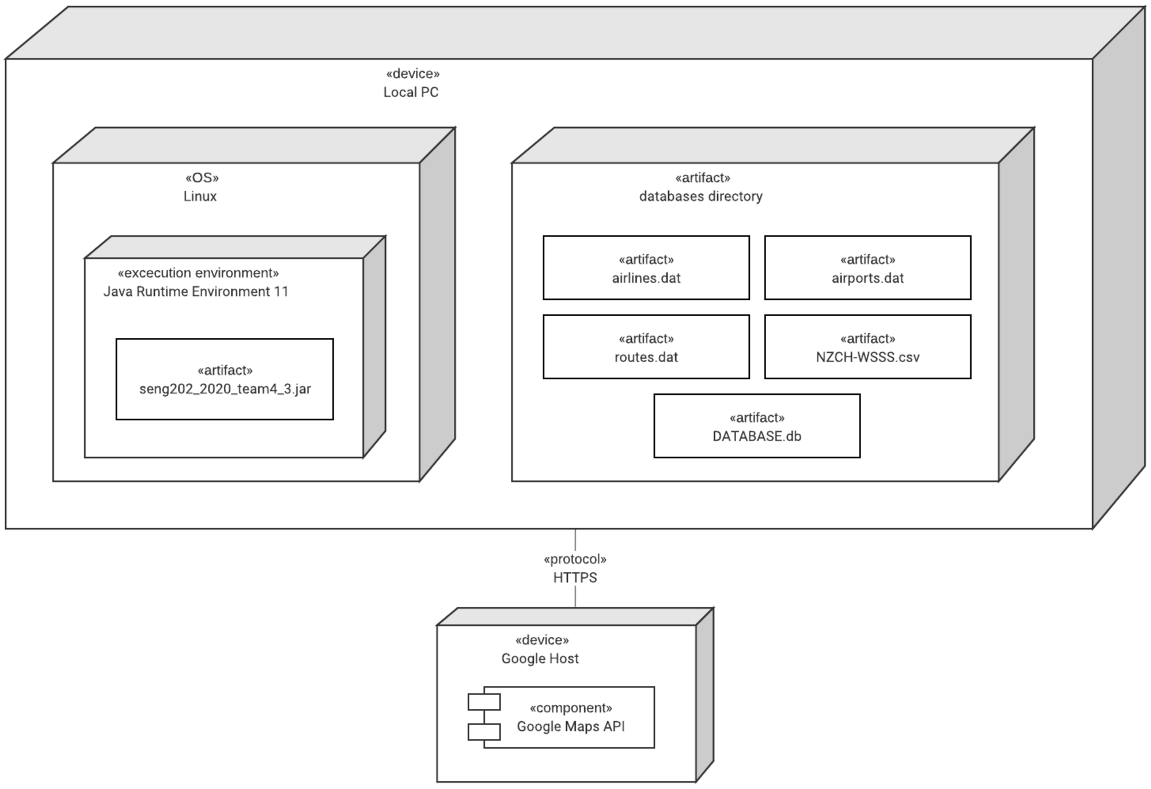

Figure 10 – UML deployment diagram for the flight tracking application.

6. Detailed UML Class Diagram

Package Diagram

Figure 11 shows a UML package diagram that outlines the overall architecture of the flight tracking application. The application utilizes an MVC architectural design pattern, in which each layer consists of a package, contributing one of three divisions of the system. JavaFX will be used to design, create, manually test, debug, and deploy the application, this library is best suited to the MVC approach.

MVC is a triangular architecture. The Model layer updates the View Layer by telling it which JavaFX scene and associated FXML file to display. The users see the View Layer. The Controller Layer deals with the user's interaction with the UI. The Controller Layer then manipulate the Model Layer, which in turn updates the View Layer to display the user interface after the interaction.

For example, when a user on the 'Home' tab clicks on the 'Airlines' tab. The airlineTab.fxml file associated with the Airline Tab will be opened. The controller associated with this fxml file, shown by the fx:controller property, will be initialized, AirlineTabController in this case. The initialization process will involve calling several methods in the controller that manipulate classes in the model layer. The setTable() method, called in the controller, queries the database, and receives tuples of Airline records. Several Airline, a model class, objects are created and then added to JavaFX table view and displayed.

The advantage of using an MVC architecture are the enhanced development speed of the application, as team members are able to work on different packages of the system in parallel with little risk of overlap. Additionally, the application will be easier to maintain and upgrade due to improved modularity and low coupling that is inherent in the MVC approach. The only disadvantage is maintaining the MVC structure when programming, as all methods and classes must follow strict rules or else compromise the architecture. The advantages of MVC heavily outweigh any disadvantages as so is used in this project.

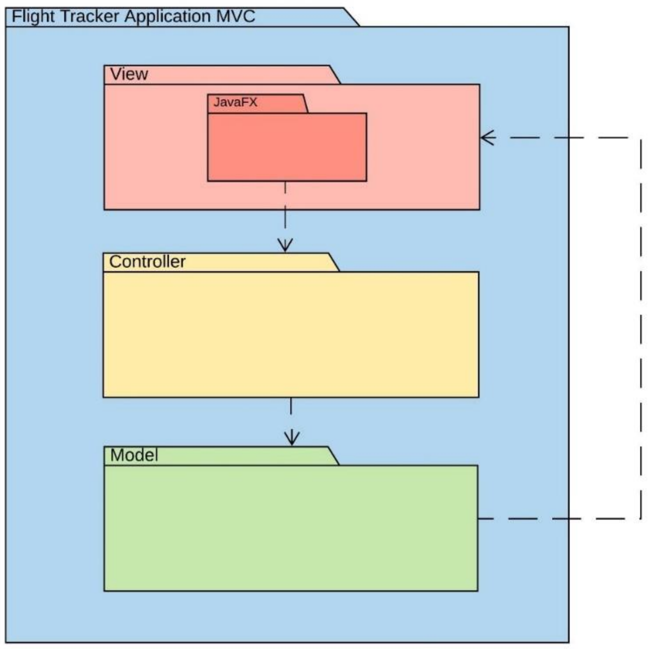

Figure 11 – UML package diagram utilizing the MVC architecture.

Class Diagram - View Package

Note - The getters and setters have been omitted from all classes.

Figure 12 shows the UML class diagram for the View package. The view package is responsible for creating the initial JavaFX stage when the application is launched. It also manages the scenes (FXML files) being displayed and is tasked with altering the graphical user interface when either, the user changes tabs, or the input it receives from the Model package changes. The user interacts with the view package and the controller package classes handle this interaction. Hence, the MainApplication.java class shown below is dependent on the Main.java class in the Model package (see Figure 14).

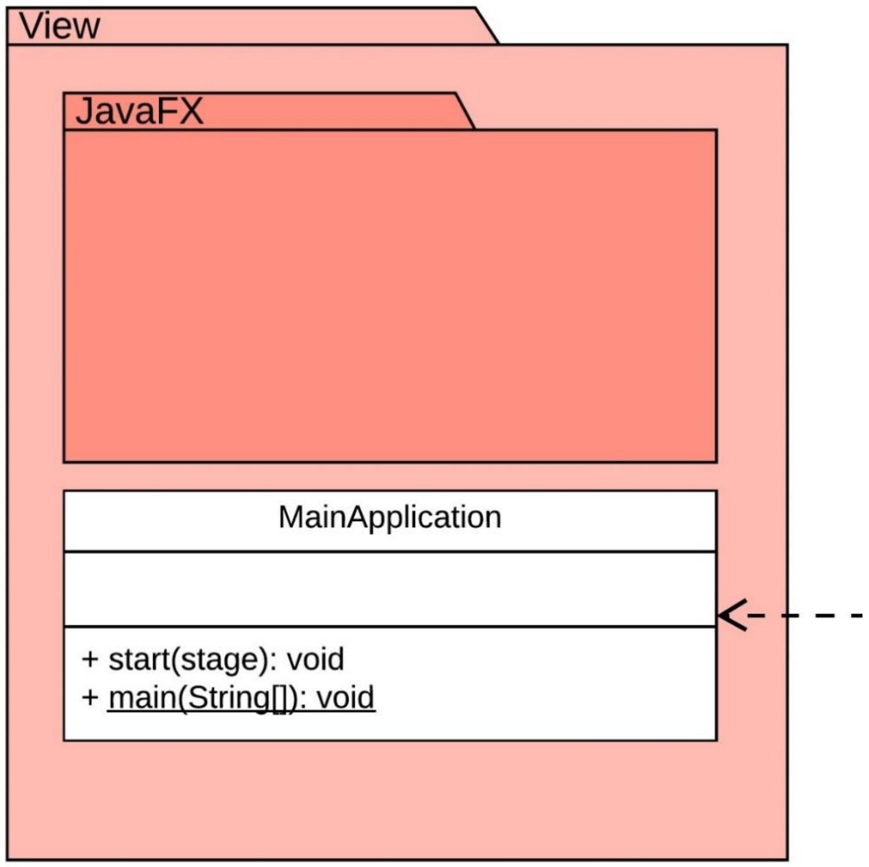

Figure 12 – UML class diagram showing the View Layer classes.

Class Diagram - Controller Package

Note – the FXML controls and components were omitted from the class diagram.

Figure 13 shows the detailed UML class diagram of the Controller package. The controller classes initialize and update all the JavaFX nodes and controls. This package is responsible for all the main functionality of the application.

The DataController abstract class describes all the required functionality for the controllers that display tables of data in their respective scenes. Hence, each tab with a data table inherits attributes and methods from the DataController. The FileUploadController allows raw data files to be uploaded to the data tables. NewRecord lets the user manually input data. The black diamond from both of these classes to the DataController represents a composite relationship. This means these two classes cannot exist without the existence of the DataController. Without the DataController these classes would not know what to do with the inputted data and would be redundant.

The filtering and instantaneous searching functionality of data in the tables is also implemented in the controller package. The filterData, addFilter and addSearchBarFilter methods perform this. Selected rows of the data table can also be deleted with the deleteRows method.

The visualization of data with Google Maps is implemented using the MapTabController. The EmissionsTabController manages the calculation and graphical display of carbon emissions for the routes selected by the user.

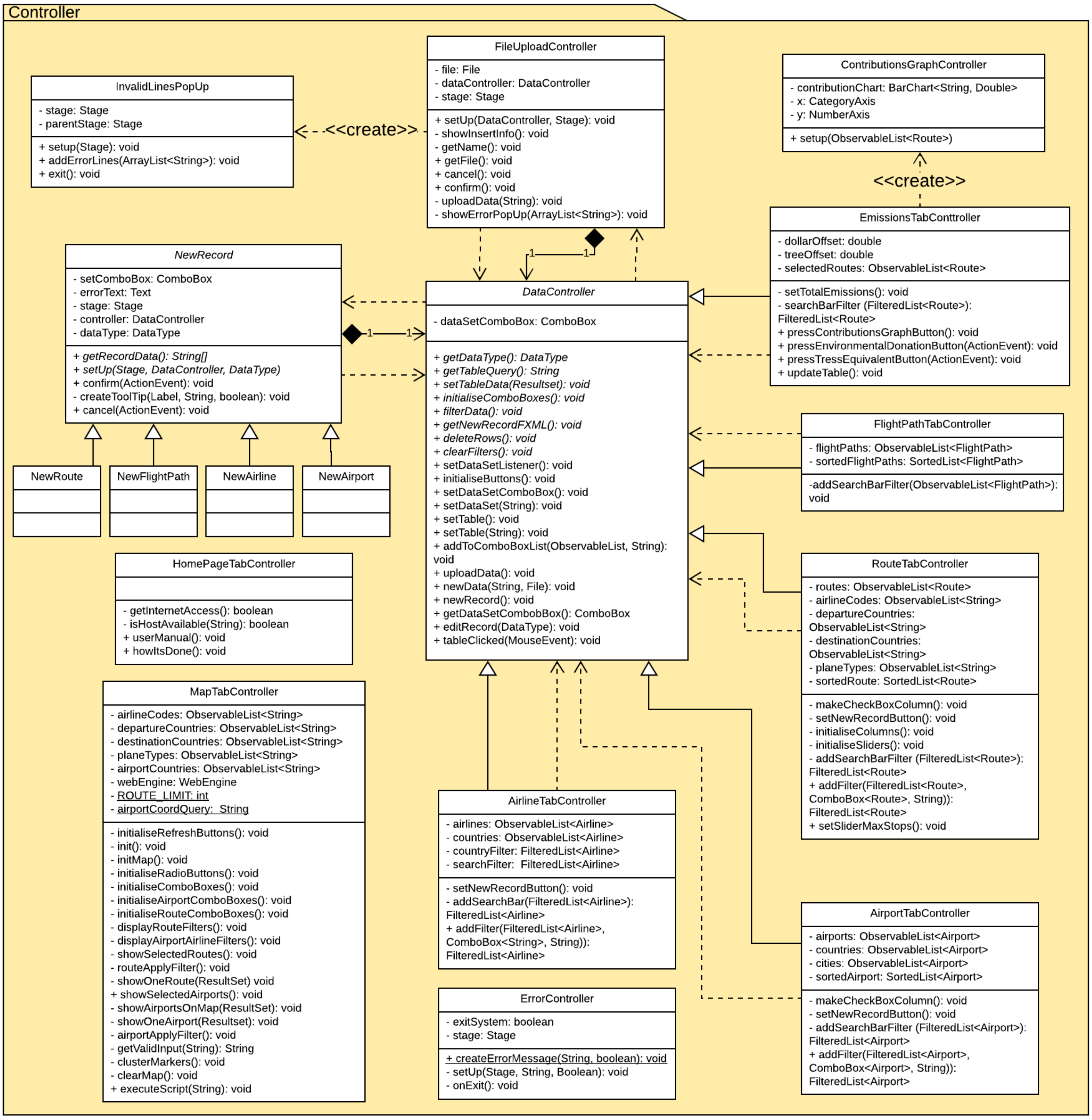
Controller
InvalidLinesPopUp
- stage: Stage
- parentStage: Stage
+ setup(Stage): void
+ addErrorLines(ArrayList<String>): void
+ exit(): void
<<create>>
FileUploadController
- file: File
- dataController: DataController
- stage: Stage
+ setUp(DataController, Stage): void
- showInsertInfo(): void
- getName(): void
+ getFile(): void
+ cancel(): void
+ confirm(): void
- uploadData(String): void
- showErrorPopUp(ArrayList<String>): void
ContributionsGraphController
- contributionChart: BarChart<String, Double>
- x: CategoryAxis
- y: NumberAxis
+ setup(ObservableList<Route>)
<<create>>
EmissionsTabConttroller
- dollarOffset: double
- treeOffset: double
- selectedRoutes: ObservableList<Route>
- setTotalEmissions(): void
- searchBarFilter (FilteredList<Route>): FilteredList<Route>
+ pressContributionsGraphButton(): void
+ pressEnvironmentalDonationButton(ActionEvent): void
+ pressTressEquivalentButton(ActionEvent): void
+ updateTable(): void
NewRecord
- setComboBox: ComboBox
- errorText: Text
- stage: Stage
- controller: DataController
- dataType: DataType
+ getRecordData(): String[]
+ setUp(Stage, DataController, DataType)
+ confirm(ActionEvent): void
- createToolTip(Label, String, boolean): void
+ cancel(ActionEvent): void
1
1
1
1
DataController
- dataSetComboBox: ComboBox
+ getDataType(): DataType
+ getTableQuery(): String
+ setTableData(Resultset): void
+ initialiseComboBoxes(): void
+ filterData(): void
+ getNewRecordFXML(): void
+ deleteRows(): void
+ clearFilters(): void
+ setDataSetListener(): void
+ initialiseButtons(): void
+ setDataSetComboBox(): void
+ setDataSet(String): void
+ setTable(): void
+ setTable(String): void
+ addToComboBoxList(ObservableList, String): void
+ uploadData(): void
+ newData(String, File): void
+ newRecord(): void
+ getDataSetCombobBox(): ComboBox
+ editRecord(DataType): void
+ tableClicked(MouseEvent): void
NewRoute
NewFlightPath
NewAirline
NewAirport
FlightPathTabController
- flightPaths: ObservableList<FlightPath>
- sortedFlightPaths: SortedList<FlightPath>
-addSearchBarFilter(ObservableList<FlightPath>): void
HomePageTabController
- getInternetAccess(): boolean
- isHostAvailable(String): boolean
+ userManual(): void
+ howItsDone(): void
RouteTabController
- routes: ObservableList<Route>
- airlineCodes: ObservableList<String>
- departureCountries: ObservableList<String>
- destinationCountries: ObservableList<String>
- planeTypes: ObservableList<String>
- sortedRoute: SortedList<Route>
- makeCheckBoxColumn(): void
- setNewRecordButton(): void
- initialiseColumns(): void
- initialiseSliders(): void
- addSearchBarFilter (FilteredList<Route>): FilteredList<Route>
+ addFilter(FilteredList<Route>, ComboBox<Route>, String)): FilteredList<Route>
+ setSliderMaxStops(): void
MapTabController
- airlineCodes: ObservableList<String>
- departureCountries: ObservableList<String>
- destinationCountries: ObservableList<String>
- planeTypes: ObservableList<String>
- airportCountries: ObservableList<String>
- webEngine: WebEngine
- ROUTE_LIMIT: int
- airportCoordQuery: String
- initialiseRefreshButtons(): void
- init(): void
- initMap(): void
- initialiseRadioButtons(): void
- initialiseComboBoxes(): void
- initialiseAirportComboBoxes(): void
- initialiseRouteComboBoxes(): void
- displayRouteFilters(): void
- displayAirportAirlineFilters(): void
- showSelectedRoutes(): void
- routeApplyFilter(): void
- showOneRoute(ResultSet) void
+ showSelectedAirports(): void
- showAirportsOnMap(ResultSet): void
- showOneAirport(Resultset): void
- airportApplyFilter(): void
- getValidInput(String): String
- clusterMarkers(): void
- clearMap(): void
+ executeScript(String): void
AirlineTabController
- airlines: ObservableList<Airline>
- countries: ObservableList<Airline>
- countryFilter: FilteredList<Airline>
- searchFilter: FilteredList<Airline>
- setNewRecordButton(): void
- addSearchBar(FilteredList<Airline>): FilteredList<Airline>
+ addFilter(FilteredList<Airline>, ComboBox<String>, String)): FilteredList<Airline>
AirportTabController
- airports: ObservableList<Airport>
- countries: ObservableList<Airport>
- cities: ObservableList<Airport>
- sortedAirport: SortedList<Airport>
- makeCheckBoxColumn(): void
- setNewRecordButton(): void
- addSearchBarFilter (FilteredList<Airport>): FilteredList<Airport>
+ addFilter(FilteredList<Airport>, ComboBox<Airport>, String)): FilteredList<Airport>
ErrorController
- exitSystem: boolean
- stage: Stage
+ createErrorMessage(String, boolean): void
- setUp(Stage, String, Boolean): void
- onExit(): void

Figure 13 – UML class diagram showing the Controller Layer classes.

Class Diagram - Model Package

Figure 14 below shows the detailed UML class diagram of the Model package. This package is responsible for loading and modifying data from the SQLite database. The DatabaseManager class is used to get a connection to the database, execute a statement, return the result, and then close the connection. The DataLoader class uploads inputted data into the SQLite database table for the corresponding type from either a stored file, manual input, or the selection of a check box.

The Model package also has classes that represent the different data types used in the application: Airline, Airport, FlightPath, and Route. While the types of data provided are dissimilar, they do share some attributes, such as name, and methods, such as getValid, getInsertStatement and equalsTest that can be generalised. Therefore, the abstract DataType class was created, each data type inherits from this class. This was done to abide by the DRY principle.

When data is inputted it must be validated before being stored as a specific data type. This is why the Validate interface was created, it has methods for validating several different common data types for flights and in general.

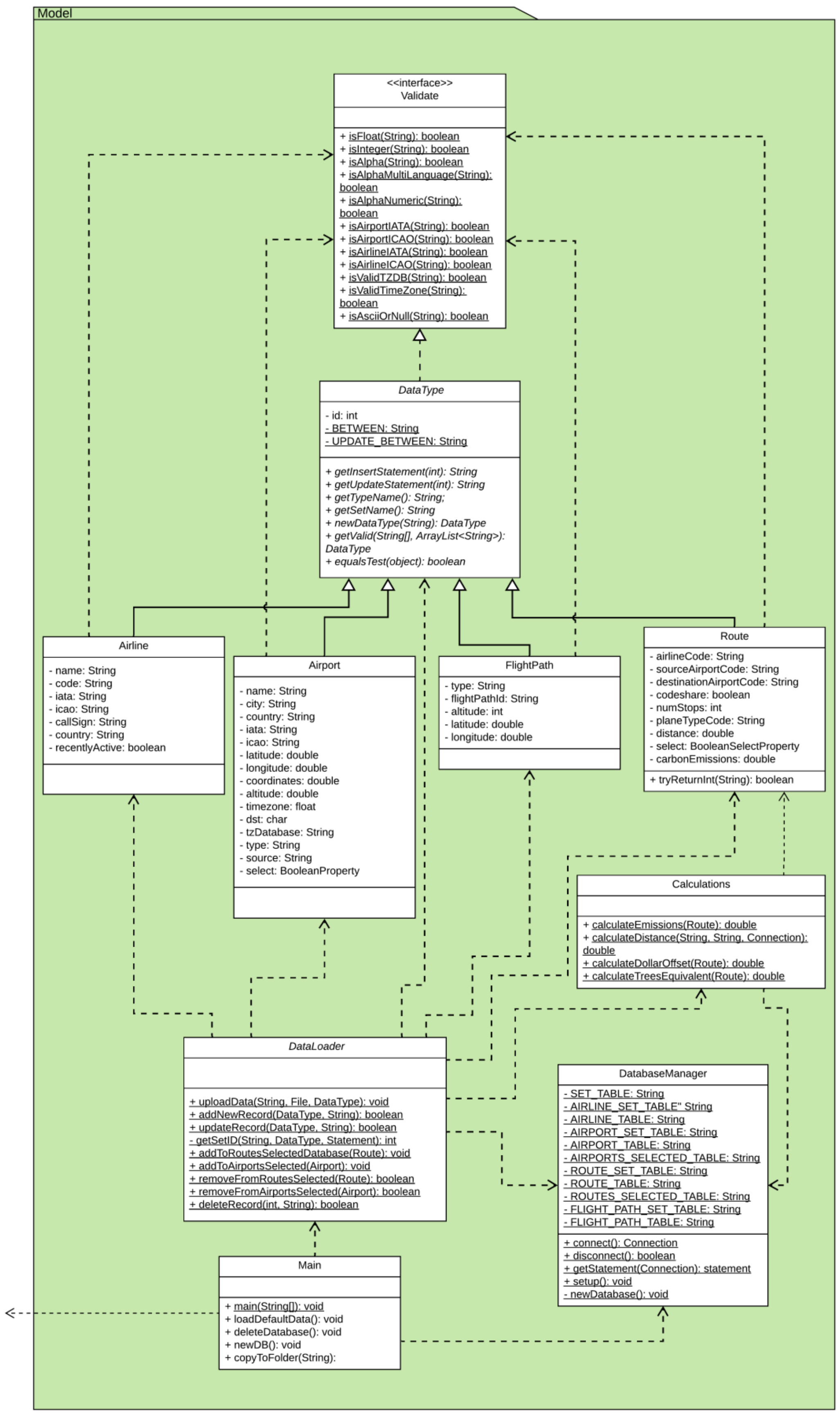
Model
<<interface>>
Validate
+ isFloat(String): boolean
+ isInteger(String): boolean
+ isAlpha(String): boolean
+ isAlphaMultiLanguage(String): boolean
+ isAlphaNumeric(String): boolean
+ isAirportIATA(String): boolean
+ isAirportICAO(String): boolean
+ isAirlineIATA(String): boolean
+ isAirlineICAO(String): boolean
+ isValidTZDB(String): boolean
+ isValidTimeZone(String): boolean
+ isAsciiOrNull(String): boolean
DataType
- id: int
- BETWEEN: String
- UPDATE_BETWEEN: String
+ getInsertStatement(int): String
+ getUpdateStatement(int): String
+ getTypeName(): String;
+ getSetName(): String
+ newDataType(String): DataType
+ getValid(String[], ArrayList<String>): DataType
+ equalsTest(object): boolean
Airline
- name: String
- code: String
- iata: String
- icao: String
- callSign: String
- country: String
- recentlyActive: boolean
Airport
- name: String
- city: String
- country: String
- iata: String
- icao: String
- latitude: double
- longitude: double
- coordinates: double
- altitude: double
- timezone: float
- dst: char
- tzDatabase: String
- type: String
- source: String
- select: BooleanProperty
FlightPath
- type: String
- flightPathId: String
- altitude: int
- latitude: double
- longitude: double
Route
- airlineCode: String
- sourceAirportCode: String
- destinationAirportCode: String
- codeshare: boolean
- numStops: int
- planeTypeCode: String
- distance: double
- select: BooleanSelectProperty
- carbonEmissions: double
+ tryReturnInt(String): boolean
Calculations
+ calculateEmissions(Route): double
+ calculateDistance(String, String, Connection): double
+ calculateDollarOffset(Route): double
+ calculateTreesEquivalent(Route): double
DataLoader
+ uploadData(String, File, DataType): void
+ addNewRecord(DataType, String): boolean
+ updateRecord(DataType, String): boolean
- getSetID(String, DataType, Statement): int
+ addToRoutesSelectedDatabase(Route): void
+ addToAirportsSelected(Airport): void
+ removeFromRoutesSelected(Route): boolean
+ removeFromAirportsSelected(Airport): boolean
+ deleteRecord(int, String): boolean
DatabaseManager
- SET_TABLE: String
- AIRLINE_SET_TABLE" String
- AIRLINE_TABLE: String
- AIRPORT_SET_TABLE: String
- AIRPORT_TABLE: String
- AIRPORTS_SELECTED_TABLE: String
- ROUTE_SET_TABLE: String
- ROUTE_TABLE: String
- ROUTES_SELECTED_TABLE: String
- FLIGHT_PATH_SET_TABLE: String
- FLIGHT_PATH_TABLE: String
+ connect(): Connection
+ disconnect(): boolean
+ getStatement(Connection): statement
+ setup(): void
- newDatabase(): void
Main
+ main(String[]): void
+ loadDefaultData(): void
+ deleteDatabase(): void
+ newDB(): void
+ copyToFolder(String):

Figure 14 – UML class diagram showing the Model Layer classes.

Inter-package relationships

Note – all attributes and models have been omitted, these are included in the respective packages class diagram above.

As the UML class diagram created was too large, we created a separate diagram, Figure 15, to demonstrate relationships between classes from different packages. This shows the relationships that the Model classes have with the Controller classes, and vice-versa.

The relationship between a controller and the data type it utilizes is a composition. This is because the controller would be redundant without the existence of its specific data type. The data table would not be able to be implemented along with all of its associated functionality. The 1 near the black diamond and 0..* at the corresponding data type represents the number of data type objects the controller may create. For example, one RouteTabController instance may create 0 to many Route objects to display in the data table. If the user chooses not to load the default data, the data table will be 0, hence the 0. If the user loads the default data, there will be thousands of Route instances displayed.

The controller classes that are dependent on the DatabaseManager interact with the database and hence require a connection to it. For example, the RouteTabController connects to the database when setting the maximum number stops, for the stops slider filter.

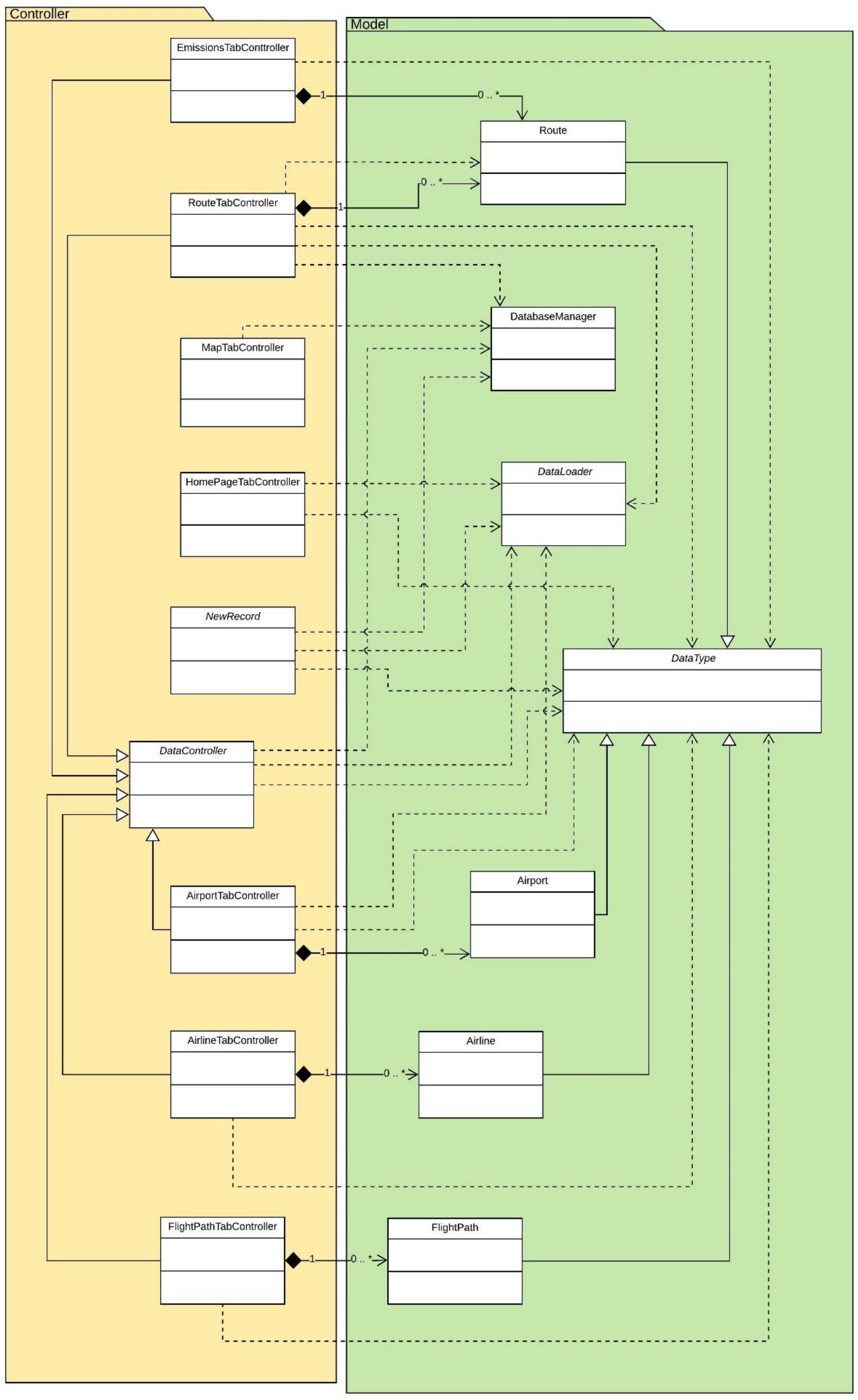

Figure 15 – UML class diagram showing the relationships between Model and Controller classes.

7. Testing Procedures

Tables 7 and 8 shows the manual tests done to test the functional and quality requirements (respectively) of the application. It describes what was done, who did it, the outcome, whether the application passed it or not and its criticality.

7.1 Functional Testing

Table 7 – Manual functional requirements and use case testing

Test ID	**Description**	**Tester**	**Result**	**Pass/ Fail**	**Criticality**	**Use Case**
Add New Record TPF-1.1	Add a new record using a valid format.	Darryl Alang	The record is added to the database in the selected dataset. Data is shown when the data table is viewed.	Pass	High	UC-1
TPF-1.2	Add a new record where latitude, longitude and altitude contain non-numeric characters.	Darryl Alang	An error message is shown to the user. The New Record window is not closed. Data is not added to the database nor shown in the data table.	Pass	High	UC-1
TPF-1.3	Add a new record where the time zone is invalid.	Darryl Alang	An error message is shown to the user. The New Record window is not closed. Data is not added to the database nor shown in the data table.	Pass	High	UC-1
TPF-1.4	Add a new record where DST is an invalid code.	Darryl Alang	An error message is shown to the user. The New Record window is not closed. Data is not added to the database nor shown in the data table.	Pass	High	UC-1
TPF-1.5	Add a new record where some fields (except IATA and ICAO) are left blank.	Darryl Alang	An error message is shown to the user. The New Record window is not closed. Data is not added to the database nor shown in the data table.	Pass	High	UC-1
TPF-1.6	Cancel adding new record.	Darryl Alang	Incomplete data that was put by the user are not inserted into the	Pass	Low	UC-1

			database nor shown in the tables.			
Sort Data **TPF-2.1**	Sort airline data by clicking the header of the table column.	Noah Irving	Clicking once show the data sorted in ascending order. Clicking the same header one more time shows the data sorted in reverse. Clicking it the third time disables the sorting method.	Pass	Low	UC-1
View Data **TPF-3.1**	View default airline, airport, and route data.	Swapnil Bhagat	Data are loaded from the database and are shown in their corresponding table in the application, separate from other datatypes.	Pass	High	UC-1
TPF-3.2	View data from another dataset (from an uploaded file)	Swapnil Bhagat	Only the data from the selected dataset are shown in the application table.	Pass	High	UC-1
TPF-3.3	View distinct data from all datasets.	Swapnil Bhagat	An 'All' dataset is automatically created, however the sets this dataset is created from may have overlapping rows which are not condensed and so show up multiple times.	Fail	Low	UC-1
Upload New Data **TPF-4.1**	Upload new distinct data from a file and save it to the database.	Griffin Baxter	New data is added to the database and has its own ID that represents its dataset.	Pass	High	UC-2
TPF-4.2	Upload data from a file with duplicate lines.	Griffin Baxter	Data values are inserted in the database and are shown in the table with their corresponding dataset. There is however no test for duplicity, thus table shows duplicate data.	Fail	Low	UC-2
TPF-4.3	Upload data from file with 1 or more invalid lines of input.	Griffin Baxter	Valid lines of data are added to the database. Invalid lines of data are ignored.	Pass	Medium	UC-2

TPF-4.4	Upload data without selecting a file.	Griffin Baxter	A warning text is shown asking the user to pick a file. File uploader window remains open.	Pass	Low	UC-2
TPF-4.5	Upload data with a name similar to current dataset names.	Griffin Baxter	A warning text is shown telling the user that the dataset name already exists. File uploader window remains open. File is not uploaded to the database.	Pass	Low	UC-2
TPF-4.6	Cancel uploading data from file.	Griffin Baxter	Data from file are not inserted into database. New dataset is not created.	Pass	Low	UC-2
View Data **TPF-5.1**	View a table of raw data containing the attributes of airports/airlines /routes	Griffin Baxter	Each of the 'Airports', 'Airlines', and 'Routes' tabs have an imbedded raw data table (JavaFX TableView) which shows all of the data in the chosen data set.	Pass	High	UC-3
Select Data **TP 6.1**	While on the airports/routes tabs with loaded data, select one or more rows of the raw data table	Griffin Baxter	Airport and route raw data tables have a checkbox column and more or more of these checkboxes can be ticked	Pass	High	UC-4
Filter Data **TPF-7.1**	Filter airline data by selecting New Zealand from the country combo box.	Swapnil Bhagat	Only airlines in New Zealand are shown in the table.	Pass	Medium	UC-5
TPF-7.2	Filter airport data by using two combo boxes: New Zealand from country combo box and Christchurch from city combo box.	Swapnil Bhagat	Airport data that matches both combo box filters are shown in the table. That is, airports in Christchurch, New Zealand are shown.	Pass	Medium	UC-5
TPF-7.3	Filter route data using slider for	Swapnil Bhagat	The number of stops is increased as the slider moves to the right.	Pass	Medium	UC-5

	the number of stops.		Only the routes with the selected number of stops are shown in the table.			
TPF-7.4	Filter route data by using multiple filters: airline code, MCO departure airport IATA, 1 stop.	Swapnil Bhagat	Only the route data that matches all filters are shown in the table. That is, routes from airline WN that departs from MCO airport and have 1 stop.	Pass	Medium	UC-5
TPF-7.5	Undo filter by selecting the blank option from the combo box or deleting the characters from the combo box text field.	Swapnil Bhagat	All data in the selected dataset are shown in the table.	Pass	Medium	UC-5
Search Data **TPF-8.1**	Search airline data by typing the keyword "new" in the search text field.	Kye Oldham	Only the airline name and country from the currently selected dataset that contains the keyword "new" are shown in the table.	Pass	Medium	UC-5
TPF-8.2	Search airport data by typing the keyword "new" in the search text field.	Kye Oldham	All airport names, cities, and countries that contains the keyword "new" are shown in the table, such as New Port Nelson (airport), New York (city), New Zealand (country).	Pass	Medium	UC-5
TPF-8.3	Search route data by typing the keyword "2b" in the search text field.	Kye Oldham	All route data whose airline codes, departure, and destination IATAs, and plane types that matches the keyword "2b" is shown in the table.	Pass	Medium	UC-5
Visualise Data (Map) **TPF-9.1**	View a clear world map when Map tab is clicked, with internet connection.	Darryl Alang	Clear map (no airport or route marker) is shown to the user. Map can be zoomed in and out.	Pass	High	UC-7

TPF-9.2	Using route filters, put route paths and airport (departure and destination) markers on selected airport.	Darryl Alang	Airport marker is shown on CHC airport. Airport markers are also shown on all airports connected to CHC airport and route paths are drawn to show this connection.	Pass	Medium	UC-7
TPF-9.3	Using airport filters, put airport markers in the country, New Zealand, selected from the country combo box.	Darryl Alang	On the map, markers are shown on all New Zealand places where there are airports.	Pass	Medium	UC-7
TPF-9.5	Put route paths on routes selected from Route tab.	Darryl Alang	Route paths are shown on the map that marks the routes selected from the Route tab.	Pass	High	UC-7/ UC-6
TPF-9.6	Put airport markers on airports selected from Airport tab.	Darryl Alang	Airport markers are shown on the map that marks the airports selected from the Airport tab.	Pass	High	UC-7/ UC-6
TPF-9.7	View a clear map by clicking the refresh button.	Darryl Alang	All airport markers and route paths are removed from the map.	Pass	Medium	UC-7
TPF-9.8	View map without internet connection.	Darryl Alang	Warning message is shown in the home tab.	Pass	Medium	UC-7
Calculate Carbon Emissions **TPF-10.1**	Calculate carbon emission of 1 route.	Noah Irving	Correct calculation of carbon emission of the selected route is shown.	Pass	High	UC-9
TPF-10.2	Calculate carbon emission of a route different from previous selection.	Noah Irving	Previous selection does not affect the current one. Accurate calculation of carbon emission is shown for the currently selected route.	Pass	High	UC-9
TPF-10.3	Calculate carbon emissions of 2 or more routes.	Noah Irving	Total carbon emitted from all routes is shown.	Pass	High	UC-9
TPF-10.4	Show carbon emissions	Noah Irving	Emissions tab show 0.00 carbon emissions	Pass	High	UC-9

	without any selected routes.		when no routes are selected.			
Graph Carbon Emissions **TPF-11.1**	Show a graph comparing the carbon emissions of selected routes.	Noah Irving	Graph comparing the carbon emissions of selected routes are shown via clicking a button on the emissions tab.	Pass	Medium	UC-10
View Donation Amount **TPF-12.1**	View donations to offset carbon emissions of 1 route.	Noah Irving	Donations based on the carbon emissions of the selected route is shown.	Pass	High	UC-11
TPF-12.2	View donations to offset carbon emissions of 2 or more routes.	Noah Irving	Donations are shown based from the calculated carbon emissions of the selected routes.	Pass	High	UC-11
TPF-12.3	View donations without any selected routes.	Noah Irving	Donation offset show $0.00.	Pass	High	UC-11
View tree equivalent amount **TPF-13.1**	View trees equivalent to offset carbon emissions of 1 route.	Noah Irving	The number of trees to offset the carbon emission of the selected route is shown.	Pass	High	UC-12
TPF-13.2	View trees equivalent to offset carbon emissions of 2 or more routes.	Noah Irving	The number of trees to offset the carbon emission of the selected routes is shown.	Pass	High	UC-12
TPF-13.3	View trees equivalent without any selected routes.	Noah Irving	Equivalent trees show 0.	Pass	High	UC-12
Read Data **TPF-14.1**	Read additional information about a certain row of airline, airport, route, flight path, emissions data.	Griffin Baxter	Double clicking a row from airline, airport, route, flight paths, and emissions tables show a window with the selected data and some additional information about it.	Pass	Medium	UC-1
TPF-14.2	Read additional information from an empty table row.	Griffin Baxter	Details window is not opened.	Pass	Medium	UC-1
Update data **TPF-15.1**	Edit the selected airline, airport, route,	Noah Irving	Updated data is shown in the table view. Reading additional	Pass	Medium	UC-1

	flight path, or emissions data.		information also shows the updated data.			
TPF-15.2	Cancel editing data.	Darryl Alang	None of the data values are modified.	Pass	Medium	UC-1
Delete data **TPF-16.1**	Delete a record without selecting any data.	Swapnil Bhagat	No data is deleted. All data are shown in the table.	Pass	Medium	UC-1
TPF-16.2	Delete an airline record by selecting a row from the table view and clicking the X button.	Kye Oldham	Selected data is removed from both the table and the database. Changing the displayed dataset does not show the data, even when searching for it.	Pass	Medium	UC-1
TPF-16.3	Delete multiple records by selecting multiple rows and clicking the X button.	Kye Oldham	All selected data are removed from the table and the database. Data does not show when dataset is changed or when data is searched.	Pass	Medium	UC-1

7.2 Quality Testing

Table 8 - Quality requirement testing of application

Test ID	**Description**	**Tester**	**Result**	**Pass/ Fail**	**Criticality**	**Quality Require-ment**
Perfor-mance **TPQ-1.1**	Loading times for files with size <= 100000 lines should be less than 10 seconds.	Swapnil Bhagat	Uploading airport, airline, and route files of around 140000 lines took about 6 seconds.	Pass	High	QR-1
Respon-siveness **TPQ-1.2**	Navigation throughout the application should be perceivably instantaneous.	Swapnil Bhagat	Switching between tabs, loading (not uploading) medium sized (around 30000 lines) data from the data set chooser was instantaneous provided sufficient RAM was available.	Pass	High	QR-1

Readability **TPQ-2.1**	Code should be readable with sufficient comments and JavaDoc such that a new developer can follow with minimal guidance	Griffin Baxter	Every class has an explanatory comment. Important methods and attributes have an associated JavaDoc. In-line comments are used to understand the more complex parts of the code.	Pass	Medium	QR-2
Maintaina-bility **TPQ-2.2**	Code should be modular and follow principles of low coupling high cohesions so that a new programmer can add to the code with ease	Griffin Baxter	An MVC architecture was used to achieve separation of tasks over the three different packages. Inheritance and interfaces were used to lower repetition and increase cohesion.	Pass	Medium	QR-2
Usability **TPQ-3.1**	The app should be usable for a non-technical person who will rely on intuition or instruction to navigate and carry out their tasks	Noah Irving	The GUI was made to look and feel simple using JavaFX so that it does not feel cluttered and the user has no idea what to do. The app was given to friends of development team to see any pitfalls of this GUI design. The current version of the GUI is simple to use, especially with the use of the user manual, however, this is not necessary or a reliance.	Pass	High	QR-3

Compatibility **TPQ-4.1**	The application should be compatible on primarily the Mint Linux OS that the UC lab machines use, and secondarily on consumer grade Windows 10	Noah Irving	The application was tested on the lab machines as well as the developers' own personal (Windows) machines. Mac OS was not tested.	Pass	High	QR-4
Reliability **TPQ-5.1**	Calculations should produce reliable and reproducible results for carbon emissions of routes and its donation equivalent	All Development Team	Each developer did their own research on carbon emissions for flights and corroborated their findings to ensure reliability. The same, although to a lesser extent, was done for the donation/trees equivalent calculations	Pass	High	QR-5

7.3 Discussion

The table above shows all the tests that have been performed to assess both the functional and the quality requirements of the application.

The manual tests cover all use cases however some functionalities planned for the application are not yet implemented in this version. For the tested use cases, the different test cases were used such that they covered all the basic, alternative, and exceptional flows that the team expected the target users to do while using the application. Automated testing (using Cucumber) was not implemented this Phase due to time constraints, prioritisation, and lack of knowledge. This will be done in Phase 3 to ensure all required acceptance tests are passed.

The application passed 49 out of 52 of the functional test cases and all 7 of the quality test cases, exceptions being the ones that upload and view duplicate data. When uploading new data from a file or manually adding data records, the application should check whether the given set of values is already in the database or not. These tests have low criticality; thus, it is not one of the priorities at the moment, but this will be implemented on the next version of the application.

For the unit tests, they only included the model classes. Although the controller classes were not assessed by the unit tests, their functionalities were mostly covered in the manual tests mentioned above. In the model package, the unit tests have 100% class coverage, 69% method coverage and 77% line coverage. These tests did not cover the basic setters and getters of each class. Currently, the application passed all the unit tests written by the team. Additionally, we also have automated acceptance tests with Cucumber. All of these tests pass as well, and also test model package classes.

Quality testing of the application was done largely throughout the creation process with iterative improvements over the course of Phase 2 and Phase 3. The functional testing above, the testing description of the quality test and procedures used are outlined in Table 8 above.

8. Current Project Version

Requirements and features implemented and not implemented in this release

Table 9 below shows which use cases have or have not been implemented in Phase 2 and Phase 3, and any reasons associated with this outcome.

Table 9 - Implementation table of use cases and associated functional requirements

Use Case	Functional Requirements	Description/Future implementation
UC-1	*Implemented:* FR-1, FR-2, FR-3, FR-4, FR-5, FR-6	User is currently able to create and delete rows of data from the raw data viewer. They are also able to see further information (as most information is on the table anyway) and modify the data attributes.
UC-2	*Implemented:* FR-7, FR-8, FR-9 *Not implemented:* FR-10	The user is able to upload data through a file browser – which only allows .dat, .txt, and .csv files to be uploaded (not possible to upload any other type). This data is parsed and validified before being put into the raw data table. However, duplicates are not condensed and will show up in the table if files contain them.
UC-3	*Implemented:* FR-11	The user is able to view data in the raw data viewer under 'Airlines', 'Airports', and 'Routes' tabs.
UC-4	*Implemented:* FR-12	The user is able to navigate to the 'Airports' and 'Routes' tabs and, provided data is loaded, tick a checkbox to select it to view later in the emissions or map tab. The user must refresh the emissions table for data to be loaded into it.
UC-5	*Implemented:* FR-13, FR-14, FR-15, FR-16, FR-17	Filtering of airlines, airports, routes, and maps data was completed during Phase 2 and happens instantaneously through listeners. Searchable combo boxes were implemented to make filtering quicker and user friendly while a holistic search bar for each data type was implemented for ease-of-use. Sorting was handled automatically by the JavaFX TableView object via clicking on the column header.
UC-6	*Implemented:* FR-18, FR-19, FR-20, FR-21, FR-22	The user is able to select data using the checkbox column in the airports and routes tabs. These checked routes/airports are able to be viewed on the map through the 'View Selected {Airports|Routes}' button. Similarly, selected routes are able to be viewed on the emissions tab alongside their distance and carbon emissions. A sum of all routes selected carbon emissions is also shown.
UC-7	*Implemented:* FR-23, FR-24, FR-25	Routes and airports are now visualised using the Google Maps API. Airports are shown as points, while routes are shown as lines between two points. The routes/airports shown are able to be filtered and shown through searchable combo boxes. Limits for how many routes/airports are shown on the map are in place to avoid clutter.

UC-8	*Implemented:* FR-26, FR-27 *Not implemented:* FR-28	The user, after selecting their chosen routes, is able to view the routes' distance travelled and carbon emissions. This was not done for airports as discussed below.
UC-9	*Implemented:* FR-29 *Not implemented:* FR-30	Graphing has been implemented into the application in the emissions tab via pressing a button. However, this only works for routes in the emissions tab, and not airports, this is discussed below.
UC-10	*Implemented:* FR-31	The user, after selecting then loading routes into the emissions tab, is able to see the New Zealand Dollar equivalent of the carbon emissions generated from said routes.
UC-11	*Implemented:* FR-32	The user, after selecting then loading routes into the emissions tab, is able to see the tree donation equivalent of the carbon emissions generated from said routes.
N/A	*Implemented:* FR-33, FR-34	The ability to create and switch between different datasets (i.e. different files) was implemented as well as persistent storage system when new databases are stored in the "GreenFlights_Resources" folder. If a user loads a few sets of data then closes the app, they will be able to continue from where they left off once they re-open the app.

Note:

As a group we decided to omit carbon emissions calculations for airports (FR-28, FR-30). This was because we realised that basing the carbon emissions of an airport solely based on the number of outgoing flights was inaccurate and not representative of real life.

Features we are most proud of

SQLite database implementation: This allows for fast uploading and especially fast filtering and sorting of data in the application. In terms of metrics, this accounts to being able to upload all of the sample flight data for the airlines, airports, and routes within eight seconds. This is due to the fact that database systems such as SQLite include various optimisations, many of which would be time-consuming and not as effective to implement ourselves if we were to create our application without a database.

Responsive GUI design: This means that the program is resizable and scales well to various screen sizes and shapes. This makes the application suitable for various devices such as desktops, laptops and even smartphones which is an avenue for the future. Additionally, adding JFoenix support to the JavaFX implementation has significantly improved the visual appeal of the application, which uses design concepts from Google's material design.

Data set management: The application's ability to swap between different data sets in the four tables of data for airlines, airports, routes, and flight paths. This allows the user to quickly switch between various data sets and perform different analysis, sorting and filtering for each one.

Another feature that we are most proud of is the ability to view the carbon emissions of certain chosen routes. This feature works quickly by selecting routes in the route table, and adding them to the emissions tab. The emissions tab allows the user to view the total CO2 emissions from the selected routes and their donations and tree equivalents, in order to offset emissions. The application also features the ability to view a graph of all of the selected routes' CO2 emissions.

And finally, another feature that we are most proud of is the Google Maps integration, which allows for flight routes and airports to be selected in their respective tables with a check box and load them into the map to be visualised.

9. Risk Assessment

The team identified several risks that would be detrimental to the project should they not be managed responsibly. The likelihood of a risk occurring, and impact of a risk should it occur were measured on a Low-Medium-High scale, see Table 10.

Table 10 – Potential Risks analysed; prevention techniques identified

ID	Description	Impact	Likeli-hood	Responsi-bility	Consequences	Prevention
RA-1	Team conflict	Medium	Low	Develop-ment Team (SH-2)	Lack of communica-tion.	Make sure all team members are on the same page and are happy with the direction the project is heading. This will be done during meetings, where every team member will have a chance to voice any concerns so that a resolution may be reached. If not, the SENG teaching staff will be consulted to reach a decisive answer.
RA-2	Unfamiliar development tools/APIs	Medium	High	Develop-ment Team (SH-2)	More time required to complete the technical aspects of the project.	Read the documentation and watch relevant and informative videos. Allocate tasks to those who are most experienced in that tool/API. Get a team member who understands the tool/API to bring the other team members up to speed. Do not use extremely complex tools and APIs. As this project uses an iterative approach, it is very likely that features will be worked on more than once, so different team members will work on the same technology (e.g. Google Maps API) as to learn and understand the code better.

RA-3	Various program-ming skill levels	Low	High	Develop-ment Team (SH-2)	Only some team members have the knowledge to complete a task. May lead to some team members completing more work than others.	If someone is stuck with a particular programming related challenge, at a team meeting or in text communication, get other team members to help and explain concepts or parts of the code, additionally, further help can be accessed by contacting the SENG 202 teaching team. Help is done through instant messaging in Discord, where every team member will remain active often as to not miss a message.
RA-4	Loss of data (i.e., code or database files got deleted, or hard drive broke)	High	Low	Develop-ment Team (SH-2), Database Host (SQLite), GitLab	Significant project delay. Frustrated team members.	Turn on auto-save if applicable. Save files as often as possible. Push changes to repository for every minor milestone/Trello task. And finally, everyone will have and create back-ups of repo files in case the repository becomes messed up.
RA-5	Product does not agree with stakeholder expectations	High	Medium	Develop-ment Team (SH-2)	Delayed or cancelled product release and implementa-tion, the product will need to be adjusted to meet stakeholder expectations.	Use the given feature packages as skeleton-requirements so that the stakeholders – primarily the teaching staff is given at least the minimum of what they expect. Further functionality will be created to make the app unique, with some features being discussed with the teaching staff for viability/usefulness.

RA-6	Code written by team member not readable by another	Medium	Medium	Develop-ment Team (SH-2)	The team member who cannot understand code written by a teammate may be unable to complete their task. This causes a project delay.	If a team member finds code that they do not understand, the writer(s) of the code must explain to the team what the code does in a longer form discussion than can be communicated through a Javadoc comment (e.g. in a meeting). From this, comments will most likely need to be modified such that everyone understands the contents and even refactoring and organising of the code can be done to make it more readable and easier to understand by every team member. Google's coding conventions will be followed, and team members are instructed to write informative comments.
RA-7	Inaccurate calculations	Medium	Medium	Develop-ment team (SH-2)	Application cannot be trusted by target audience.	Ensure that the formulas used in said formulas are accurate for the calculation we are trying to make, by getting multiple team members to reference and research formulas for different types of calculations so that we may corroborate our findings for increased reliability and accuracy. Additionally, for important calculations of the application such as carbon emission statistics, the calculations should be checked with Junit tests.

RA-8	Miscomm-unication with lecturers or teammates	Medium	Medium	Develop-ment team (SH-2), SENG202 Course Staff (SH-3)	Could lead to several teammates working on the same thing, which is a waste of time. If miscommuni-cation occurs with the lecturer the team may be working incorrectly, this will also cause time to be wasted.	Ask for help if needed or clarification on topics mentioned in lectures, tutorials or outside of classes. Have constant communication with team members and the SENG 202 teaching team. Always notify teammates of what you are working on through our main communication channel, Discord.
RA-9	Personal Entangle-ments	Medium	Medium	Develop-ment Team (SH-2)	A team member may not be able to complete an assigned task, leading to a delay in the implementa-tion of the product	Notify teammates via our Discord text instant-messaging communication channel or the shared timetable as soon as possible of any personal entanglements, therefore then necessary steps can be taken to keep the project on track. Other team members will have to share the task(s) assigned the unavailable team member so work progress is not compromised.
RA-10	COVID-19 causes Level 4 lockdown	High	Low	SENG 202 Course Staff (SH-3), Develop-ment Team (SH-2), University	Team unable to meet in person or receive help from tutors and lecturers in-person. Stress and lack of motivation.	Plan for how the team will stay on track if another lockdown occurs, dependant on the alert level and restrictions at the time. Our team will conduct daily video calls and stand-up meetings, either conducted on Discord or Zoom if with teaching staff or tutors.

RA-11	Develop-ment tool becomes unavailable	High	Low	Develop-ment Team (SH-2)	Team unable to implement and maintain the product. Flight tracker application down. Problem unable to be fixed by the development team.	Do not rely too heavily on a single development tool, research viable alternatives before attempting the implementation of a feature that relies on an external tool. Put measures in place so that if one tool fails, another can be used instead as a back-up, which should minimise or prevent unnecessary or unplanned downtime.
RA-12	No Internet connection for one or more team members	High	Low	Develop-ment Team (SH-2)	Developer unable to: work on product, communicate with the team or complete assigned tasks.	Team members should be willing to travel to and from UC campus if their internet stops working. If needed, UC computers may need to be used to solve the issue or personal devices may need to be used instead – in this case, all members should have a device with IntelliJ installed with all the required libraries/technologies.
RA-13	Product too ambitious and complex to be completed in the given timeframe	High	Low	Develop-ment Team (SH-2)	Planned features become unfeasible to implement on-time. The product will not be what stakeholders expect it to be.	Features should come out in separate releases and the important/integral components of the application should be implemented before anything else. This is so that a minimum viable product is able to be achieved within the time constraints of the project, before the addition of extra features that could prove unnecessary, especially without the basic product implementation.

RA-14	Assignments and/or tests for other courses are prioritised over project	Medium	High	Develop-ment Team (SH-2)	Team members will have less time to spend on the project. Assigned tasks may not be completed.	Check workload for all other courses in advance, with information from course outlines and calendars. Therefore, the team can plan how each team member can best spend their time during this busy period, especially with most of the team being enrolled in the same courses as each other.

10. Project Plan

The team aims to develop an application that is sustainable, on-time, on-budget, meets all the requirements for the project and is of adequate quality. The application will be sustainable by having minimal reliance on external sources, therefore keeping most of the application to be self-sufficient and easily maintainable. This means any issues encountered in the application can be fixed within the application, which dramatically reduces the complexity of the application. To produce the project on time, team deadlines are set before the hard deadlines set by the course, allowing plenty of buffer time in the case of unexpected difficulties that may be encountered. These buffers of which, are visualised in the graphs below. Furthermore, tasks are delegated to different team members to work on concurrently to make optimal use of the team and will keep track of these tasks on Trello. To keep the project on budget, only free-to-use software are used throughout the development of the project, and all libraries used have a free-use license. The project will have passed its requirements if all the tasks that are planned on the Trello board, which include all the requirements, have been completed. The product will be adequate quality once the application is tested by real-world users to see if they are easily able to complete the series of acceptance tests. Should there be time left over, it will be used to increase the feature set of the application; this could include cloud-based databases or additional API integration.

Given the risk of New Zealand re-entering a lockdown, the development team made a contingency plan which was put into effect once Christchurch was put into level 2. The team has had multiple detailed voice call meetings to discuss what they have done, what they need help with and the plans on what to do in the future. We also have had daily communication over the preferred messaging app discussing problems and helping each other out. Constant communication has made it such that being put into level 2 has not had much of an effect on working on the project.

The milestones for the project, including future milestones, are shown below, where Table 11 shows the major project milestones, Table 12 shows the milestones of phase 1, Table 13 shows the milestones of phase 2 and Table 14 shows the milestones of phase 3. Additionally, project tasks, with an estimated number of days for completion is shown in Table 15.

Table 11 – Major milestones for the project.

Milestone	Date	Description
Deliverable 1	03/08/20	Design document and checklist completed and reflections for weeks 1-3.
Deliverable 1 Presentation	05/08/20	
Deliverable 2	21/09/20	Update design document and add new sections. Finish first release build of application.
Deliverable 2 Presentation	24/09/20	
Deliverable 3	5/10/20	Update design document and add new sections. Finish the application and testing of it.
Deliverable 3 Presentation/Demo	Course Week 11	Present about the application and demo the application.

Table 12 – Milestones for the phase/deliverable 1 of the project.

Milestone	Date	Description	Related tasks
Business and system context completed	28/07/20	Clarify the vision and rationale behind the product	TA-8 to TA-9
Stakeholders and requirements completed	28/07/20	Stakeholders and requirements for the product stated	TA-10 to TA-14
Acceptance tests completed	30/07/20	Determines if the described key features are functioning correctly, with tests performed by the end-user.	TA-15 to TA-18
GUI prototypes Completed	29/07/20	Lo-fi GUI prototypes created to the give end-user an idea of the final product	TA-19
UML class diagram completed	30/07/20	UML class diagram that describes design patterns and all major classes with their respective relationships	TA-21
Deployment model completed	29/07/20	Provides insights about the system	TA-20
Risk assessment completed	26/07/20	Risks identified so prevention techniques can be put into place	TA-22
Project plan outlined	01/08/20	Project plan created; this will be adapted as the project progresses	TA-23 to TA-24
Design document finished	02/08/20	All sections of the design document completed	TA-7 to TA-15, TA-19 to TA-23
Checklist completed	30/07/20	All components of the checklist signed off by teaching staff	-
Presentation given to class	05/08/20	Presentation for phase one completed	TA-26
Phase 1 completed	05/08/20	Phase 1 of project completed. and a presentation	TA-1 to TA-15, TA-19 to TA-23, TA-25 to TA-26

Table 13 – Milestones for the phase/deliverable 2 of the project.

Milestone	Date	Description	Related tasks
Added model classes	12/08/20	Basic model classes for different data types are added to the application	TA-25
GUI and loading data implemented	16/08/20	Application has a user interface with a menu bar to navigate from different data types. It can also load data from files.	TA-40 to TA-42
Viewing and filtering implemented	21/08/20	Application allows user to view data in tables and filter based on names, countries, etc.	TA-27 to TA-34
Data analysis implemented	02/09/20	Users can search for airlines, airports, or routes and view route distance and carbon emissions. Users can view distance and carbon emissions calculations.	TA-35, TA-36
Extended loading and data input	15/09/20	Users can upload data from file or manually add new records	TA-37 to TA-39
Implemented map visualization	31/08/20	Users can view world map and add airport, airline, and route markers	TA-32 to TA-33
Application testing done	20/09/20	Application has been tested as outlined by the testing procedures section.	TA-15, TA-43, TA-44
Design document finished	21/09/20	Updated design document sections from phase 1. Added new sections for phase 2.	TA-45 to TA-52

Table 14 – Milestones for the phase/deliverable 3 of the project.

Milestone	Date	Description	Related tasks
Notify users of errors	4/10/20	Notifies user of errors encountered in uploading data and adding new records.	TA-60 to TA-61
Finish application features	4/10/20	Finish all the feature coding of the application.	TA-38 to TA-42, TA-57 to TA-63
Finish unit testing	30/10/20	Finish writing unit testing for application and pass all test.	TA-57 to TA-63
Finish acceptance testing	30/10/20	Finish manual acceptance testing of the application.	TA-38 to TA-42, TA-57 to TA-63
Finish design document	03/10/20	Finish writing and updating the new sections of the design document	TA-68 to TA-69
Write demo presentation	05/10/20	Write and practice the demo presentation.	TA-68 to TA-70

Table 15 – Project tasks, with an estimated time for completion.

Task ID	Task name	Days	Dependencies (other tasks)
TA-1	Have initial team meeting	1	-
TA-2	Set expectations for members of team	1	TA-1
TA-3	Set up communication policies, document sharing and Trello	1	TA-1
TA-4	Create yellow card and coding conventions policy	1	TA-1
TA-5	Set up Maven for all team members	2	-
TA-6	Create GitLab repository	1	TA-1
TA-7	Finalise ideas for flight tracking application	7	TA-1
TA-8	State relevant business information	2	TA-7
TA-9	Identify system context	1	TA-7
TA-10	Identify stakeholders of product	1	TA-8 to TA-9
TA-11	Create use case diagram	2	TA-10
TA-12	Identify functional requirements	1	TA-10
TA-13	Identify quality requirements	1	TA-10
TA-14	Identify key drivers	1	TA-13
TA-15	Write black box tests	1	TA-12
TA-16	Create GUI wireframes	2	TA-12
TA-17	Create deployment diagram	2	TA-7
TA-18	Create UML class diagram	3	TA-12
TA-19	Complete risk assessment table	2	-
TA-20	Identify milestones	1	TA-1
TA-21	Record tasks in Trello	8	TA-1
TA-22	Complete team reflections	3	-
TA-23	Create presentation	4	TA-8 to TA-22
TA-24	Get OpenFlights data	1	-
TA-25	Create classes for each datatype	2	TA-24

TA-26	Create GUI FXMLs	2	TA-15
TA-27	Upload data into database	1	TA-24
TA-28	Load airport, airline, and route data into the application	2	TA-24
TA-29	View data in tables	2	TA-24, TA-26
TA-30	Add data into existing datasets from file	2	TA-26 to TA-29
TA-31	Add new records within the application	1	TA-26 to TA-29
TA-32	View map	3	TA-26
TA-33	Put airport and route markers on map	2	TA-32
TA-34	Add data filters for all data types	2	TA-29
TA-35	Add data search for all data types	2	TA-29, TA-34
TA-36	Sort data	1	TA-29
TA-37	Browse between different data sets of the same type	2	TA-29 to TA-31
TA-38	Calculate distance between airports	2	TA-28
TA-39	Calculate carbon emissions of selected routes	1	TA-38
TA-40	Calculate offset donations	1	TA-39
TA-41	Calculate trees equivalent	1	TA-39
TA-42	Add data validation	2	TA-30 to TA-31
TA-43	Add Junit tests	3	TA-38 to TA-42
TA-44	Do white and black box tests	5	TA-29 to TA-43
TA-46	Create application jar file	1	TA-29 to TA-42
TA-45	Update business and system context	1	TA-8 to TA-9
TA-47	Update stakeholders and requirements	2	TA-10
TA-48	Update detailed UML class diagram	2	TA-18
TA-49	Update risk assessment	1	TA-19
TA-50	Update project plan	2	TA-20 to TA-21
TA-51	Write testing procedures	3	TA-43 to TA-44
TA-51	Write current product version	1	-
TA-52	Write design document change log	3	TA-45 to TA-50
TA-53	Write application README	1	TA-46
TA-54	Prepare presentation for phase 2	3	TA-46 to TA-51
TA-55	Deliver presentation	1	TA-54
TA-56	Generate data from FlightPlanner API	2	-
TA-57	Disallow duplicate data sets	2	TA-30 to TA-31
TA-58	Update records within the application	2	TA-28 to TA-29
TA-59	Delete records within application	2	TA-28
TA-60	Display error message for invalid new record	1	TA-31
TA-61	Display error message for invalid records in uploaded data	2	TA-27
TA-62	View additional data	1	TA-29
TA-63	Add cucumber tests	3	TA-38 to TA-42, TA-57 to TA-62
TA-64	Update Junit tests	4	TA-57 to TA-62
TA-65	Write user manual	1	TA-63 to TA-64
TA-66	Generate application jar file	1	TA-63 to TA-64
TA-67	Update other sections from phase 1 & 2	2	TA-45 to TA-52
TA-68	Write lessons learned	1	-
TA-69	Prepare phase 3 presentation	2	TA-67 to TA-68
TA-70	Deliver presentation	1	TA-69

Additionally, the following figures outline the time taken for these tasks in phase 1 in Figure 16, phase 2 in Figure 17 and phase 3 in Figure 18.

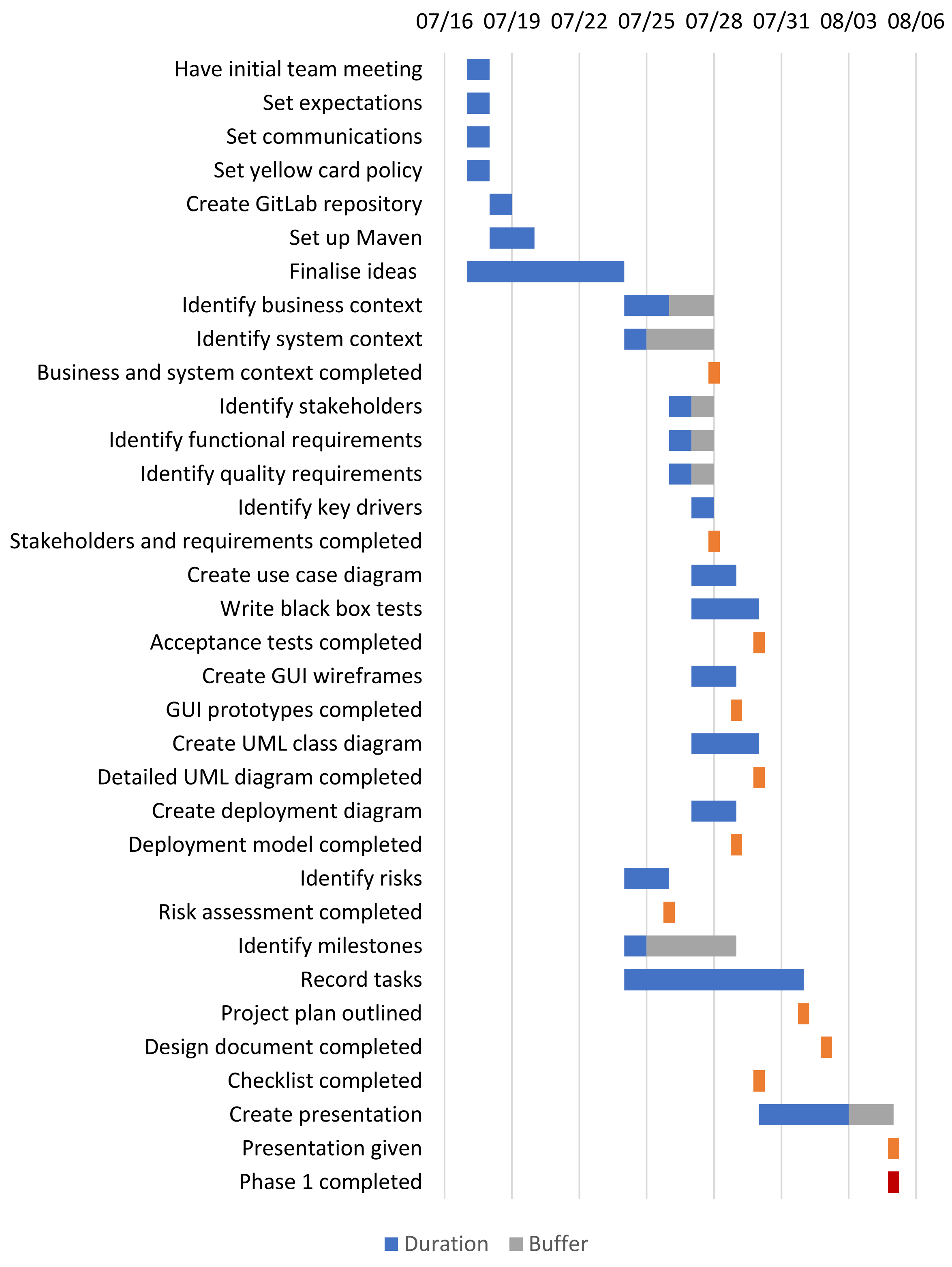

Figure 16 – Gantt chart for phase 1 tasks. Orange marks represent minor milestones and red marks represent major milestones.

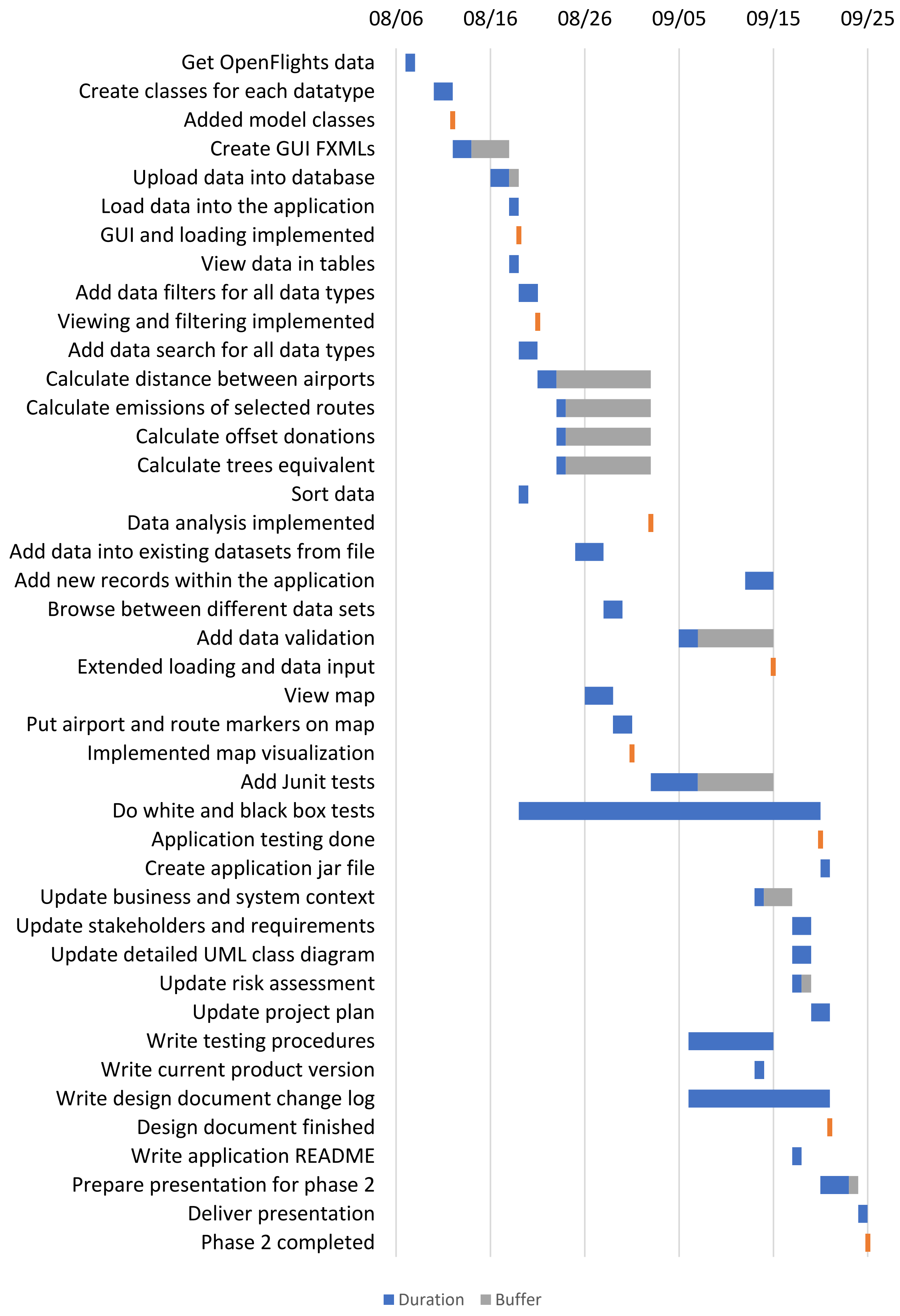

Figure 17 – Gantt chart for phase 2 tasks. Orange marks represent minor milestones.

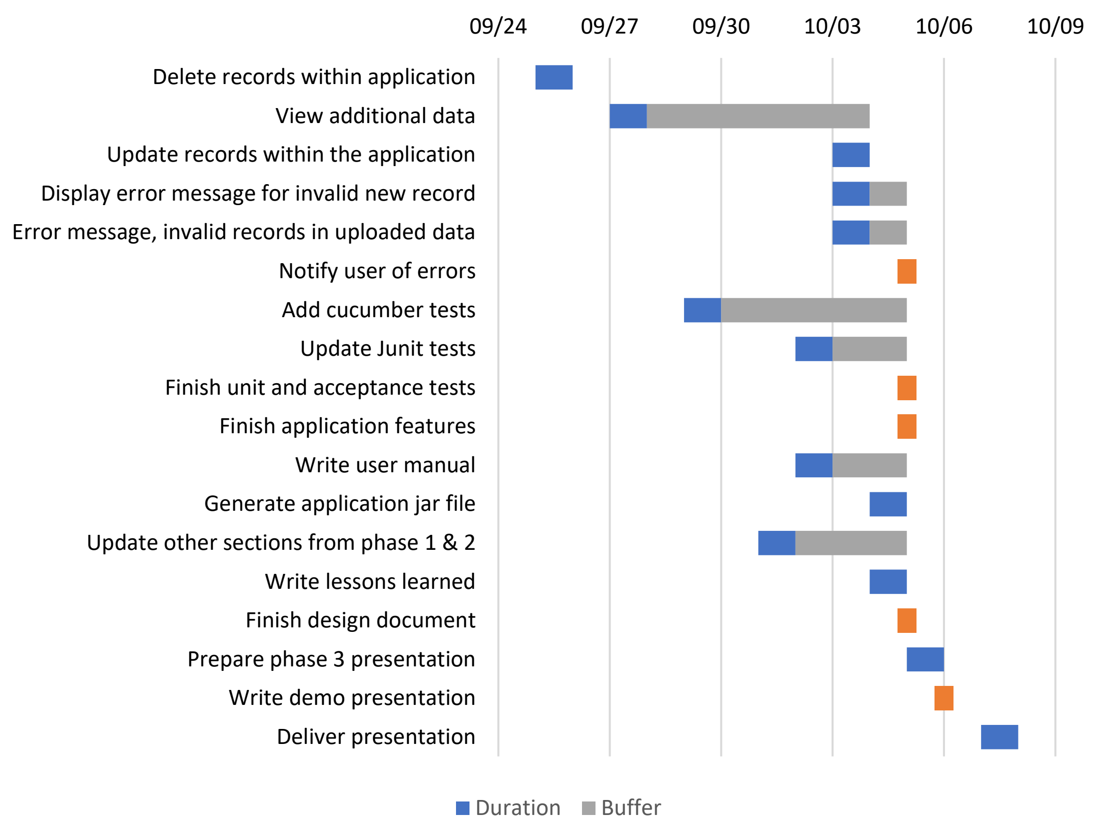

Figure 18 – Gantt chart for phase 3 tasks. Orange marks represent minor milestones.

11. Lessons Learned

Darryl

Communication is key. Always talk to your team members when you have problems about the project no matter how small. Your teammates might have the answers or could help you find answers for your problem. This helps solve problems quicker and encourages cooperation within the team. Also, it is important to actively listen when other team members are speaking. Be attentive during meetings and try not to cut others off when they are speaking. For online meetings, members should be considerate of other members and mute their mics especially when they have loud background noise. This was especially important during the COVID lockdown when everything moved online.

Provide honest feedback and learn to accept constructive criticism. Instead of keeping opinions to yourself, whether good or bad, it is better to speak up and share it to the team. For example, in the GUI designs, the team shared their opinions whether they liked the style or not and how to improve it if it does not look as good. It is also important to take in consideration other people's feedback openly. Do not be afraid to ask for and receive feedback as it helps you see where you can improve on and how to improve on it on another perspective.

Finish tasks on time, or earlier if possible. In this way, the team would have enough time to get feedback from the lecturers or the tutors and improve where needed. This course emphasized the importance of planning and time management.

Kye

I learnt that in a software engineering project, most things take longer than expected to complete. I came to this realization when designing the checkbox/'select' system and while updating the UML class diagram for phase two. For the checkboxes I thought adding and removing a route would only require an ActionEvent FXML method on the table column. However, a complex event listener was required, which required a longer period of time to understand. Similarly, the UML class diagram took a large period of time. If I were to do the project again, I would have larger time buffers in the project plan for these tasks.

Always plan for the worse. When initially writing the risk assessment matrix for this project it seemed unlikely that New Zealand would go back into a lockdown. Despite the unlikelihood, I still decided to include a move in alert levels, due to another COVID outbreak, as a risk for our project. I also wrote a contingency plan describing how the team would maintain communication and high productivity during the potential lockdown. When the move in alert levels came, our team was ready and our progress was not hindered at all. This taught me that anything can happen in a project and that planning for these scenarios will save a team countless amounts of resources in the future.

Ask for help if stuck. Often members of the team would become stuck on a task and waste precious time trying to solve it. We found that by submitting the problem to the team discord server and getting other team members to have look, solutions could be solved more efficiently.

Swapnil

The necessity of planning. I imagine most software engineers would want to 'jump straight into the code' when a new exciting project begins, this was most certainly the case for me in the beginning of this project. I realised however that although it may be boring, having an outline for the technologies to use for your project, and when to implement them saves hours of time in the long run. An example would be using the SQLite database from the beginning. Had we not discussed with the teaching staff earlier, we likely would have been tediously loading and uploading to txt files instead of simply querying a database, costing us much more time that researching and implementing SQLite.

Never underestimate time (do not get over-confident). I found that implementing complex features, for example filtering, takes a lot longer than what you would intuitively expect. As such, to compensate, it is (generally) good to overestimate the time needed to complete a task. This is especially useful when in a team project as your fellow team members may rely on you to finish a task so that they themselves can finish theirs. I also realised that doing this, I gradually understood how much time is needed for a specific level of complexity of a task. Of course, this needs to be improved, but that will only come with experience.

Do not tunnel-vision. Often when trying to fix a bug, or add a new feature, I found one or two other bugs and (ignorantly) tried to fix those lower priority bugs first before doing my main task. This, coupled with my second lesson learned, lead to poor time management as I was left with much less time for the original task than intended, sometimes making the outcome code messy and unmaintainable (which would have to be fixed later, wasting even more time). To fix this, the simplest approach was to have my own priority queue of (independent) tasks in the form of a to-do list. This way, I could keep track of my tasks, while doing those that matter the most first, preventing tunnel-vision.

Griffin

One of the key lessons I have learnt is the importance of having at least one person in the team able to organise milestones and schedule meetings. I naturally took the bulk of this work and task for my team, and I'm glad it worked out this way. I definitely wasn't being a leader of any sort, as that is counterproductive for our desires and purposes, due to us all wanted to be on the same playing field in terms of decisions made with the project, however, being the one to organise everything was extremely beneficial, and I certainly intend to use these skills in future group work and projects.

Another key lesson I have learnt is to add a large amount of buffer to the estimated time for a task's completion. In SENG 201, my project partner and I dealt with this quite often, however, over time was table to mitigate this by adding buffer to my estimated task completion times. In this SENG 202 project, due to the fact of this being a significantly larger scale project due to the number of people and the length of time being spent working on it, these projections for task completed were significantly out of kilt, even with my learnings from my previous pair programming project work. I'm glad I took part in this large project with significantly more people than I have in the past, because being able to scale predictions of completing tasks has been stressful throughout this project, but has helped me to learn a lot more and estimate these tasks better.

Finally, one other key lesson I have learnt is to try and make meeting times flexible, in terms of the amount of time spent in them. Our team never struggled with having topics to talk about during a meeting, however, sometimes when we had a room booked in Erskine or the library, it was usually for a one-hour interval. This meant that sometimes we were done half an hour in, and other times we needed an extra half an hour. This wasn't usually able to be estimated accurately before the meeting time, so this was often frustration. When Christchurch went back into level 2 lockdown, we had to hold our meeting over the internet via our Discord platform. A huge advantage we found was the ability to end a meeting at whichever time we desired, due to the fact that we hadn't booked a particular room and for a certain amount of time, and the meeting could go on for as long as we required. That is why I believe it is important to have flexible meeting times, and why that is one of my key lessons I learnt throughout this project.

Noah

During this project I have learnt how incredibly important it is to work in a team on a software project. I found about working in a team is that as a team we are able to produce a better product because we are able to come to better solutions to problems together than apart, also we are able to fill in each other's gaps of knowledge which wouldn't be possible alone.

Whilst of course working in a team we will come across a few issues, but if we are all able to clearly and concisely shared our ideas and have an environment where sharing is actively encouraged, we are able to solve these issues. I'd often have to explain why I think my solution to a coding problem is better than another team member, and if I always explained my idea and they explain theirs we'd end up coming to a happy conclusion.

I've learnt that we often underestimate the time to implement a feature. In software there can be a lot of hidden functionality that is immediately obvious to understand. This is especially so in working in teams where your often work off other people's code and may not necessarily understand how it works. One of the best solutions to this though is to go through and comment code that I've written and for other's to comment the code they've written, therefore enabling us to better understand the code that's been written.

References

1. Capoccitti, S., Khare, A., & Mildenberger, U. (2010). Aviation Industry - Mitigating Climate Change Impacts through Technology and Policy. *Journal of Technology Management & Innovation, 5*(2). doi:10.4067/s0718-27242010000200006
2. Flightradar24. (n.d.). Live Flight Tracker - Real-Time Flight Tracker Map. Retrieved August 01, 2020, from https://www.flightradar24.com/
3. Airport, airline, and route data. (n.d.). Retrieved August 02, 2020, from https://openflights.org/data.html

Appendix

Griffin Baxter mainly worked on the sections current product version, risk assessment, business and system context, stakeholders and requirements, GUI prototypes, project plan, lessons learned and deployment model.

Kye Oldham mainly worked on the sections Risk assessment, Deployment Model, Project Plan, Detailed UML Class diagram and Lessons learned.

Swapnil Bhagat mainly worked mainly on the sections Business/System context, UML class/package diagrams, and acceptance testing.

Darryl Alang mainly worked on the sections stakeholders and requirements, acceptance tests, and deployment model.

Noah Irving mainly worked on the sections business/system context, stakeholders and requirements and the project plan.

Note: All sections were worked on by all team members, the above dictates the sections the members primarily worked on.