Airline Crew Scheduler

Software Design Document

Team 8 - Skywalkers
Rahul Prajapati
Dipal Bhandari
Aniruddh Saxena
Shivani Tamkiya

Introduction

Purpose

The purpose of this system is to create scheduling software for Cornhusker Airlines, which will enable the airline crew administrator to schedule flights, assign airplanes, pilots, and other crew.

Design Goals

After designing and deploying the ACS, the airline will have improved reliability in data retrieval. To test reliability, the accuracy, precision, and recall will be measured and compared to values the client deems acceptable. Because of the repercussions for incorrect or incomplete results, these values need to be very close to 100%. This system will also have the ability to backup and restore all scheduling data, which reduces the risk of downtime. For the crew administrator, after login, the system will provide the ability to create new flights and allocate resources for each flight, track employees hours, track flight takeoff and landing times, edit flight information and cancel flights. Additionally, the crew will also be able to login to search for their assigned flights and track their working hours.

Design Trade-Offs

The accuracy of results and usability of the product for employees who are not technically proficient is of utmost importance. Security, to prevent intruders from changing, adding, and deleting flight information at great cost to the airline is also vital. The cost to build, due to increased quality control and system testing and revising, may be affected to ensure that these important specifications are fulfilled. The size allowance for entries in the database, and the size of the database as a whole will all for more detailed recording that users will be able to refer back to, but longer entries and a larger database mean access time could increase beyond what is acceptable for users.

Interface documentation guidelines

- Make status of system visible to the user with non-gray out buttons and fields as a sign that system is done processing and is ready for the user to use it further.
- Classes are named with singular nouns, i.e. “Person” instead of “Persons” or “People”.
- Classes are capitalized.
- Methods are lowerCamelCase and are named with verb phrases.

- Fields are lowerCamelCase and are named with noun phrases.
- Error status is returned as an exception. Exceptions created for all possible errors, the user should not see an internal error message but a user-friendly message to tell them what to do next instead. Anticipate that if the user can input anything, they will eventually try to input anything.
- All input will be constrained, sanitized, and validated for type, length, format, and range with regular expressions before being submitted to the database.
- If an input field has a limited or knowable number of acceptable inputs, utilize a drop-down box or radio button to allow data to be normalized.
- A verification window must show and require "OK" from the user before the input is submitted to prevent mistakes.
- User guide documentation will be easily accessible to the user from the main menu.

Definitions, Acronyms, and Abbreviations

CHA - Cornhusker Airlines

ACS - Airline Crew Scheduler

Captain -Qualified pilot for a particular aircraft

First Officer - Qualified pilot or co-pilot for a particular aircraft

Flight Attendant - Crew member responsible for the safety of the passengers in the main cabin for the duration of a flight.

GBR-10 - Type of aircraft, capacity 45 passengers

NU-150 - Type of aircraft, capacity 75 passengers

References

Group 8 Requirements Analysis Document

Object-Oriented Software Engineering, 3rd edition. Bruegge, Dutoit

Firebase Authentication (https://firebase.google.com/docs/auth/)

Firebase Firestore documentation (https://firebase.google.com/docs/firestore/)

Joda Time API (https://www.joda.org/joda-time/)

Overview

The scheduling system should be able to keep track of the CHA crew members. The system should be able to assign the correct position and number of crew members required for each type of airplane. Furthermore, the system should be able to keep track of both the estimated and actual time of takeoff/touchdown of the airplanes. The system should also be able to log in all the updates made to the schedules and these updates should be accessible for searching based on the flight number. The system will be comprised of a smartphone application with a database backend.

Current Software Architecture

Overview

Our primary architecture is a client-server architecture. But instead of hosting it to a local server we're using firebase tools which changes our whole architecture to a serverless architecture. A serverless architecture eliminates the need of server and hardware management by the developer to host the application, in our case the database and authentication. Instead, it hosts it on a third party service like the firebase tools. We're using the syncing based database and authentication functionality of firebase firestore.

The system is divided into the following subsystems:

1) UserInterfaceSubsystem: is assigned to provide authentication by using firebase authentication tool for all the employees and passengers of CHA.

2) DatabaseSubsystem: is assigned to provide a real-time database through firebase.

3) ModificationSubsystem: is assigned to create/edit non-existing/existing data from Database.

4) FetchSubsystem: is assigned to fetch the existing data from the Database.

Subsystem decomposition

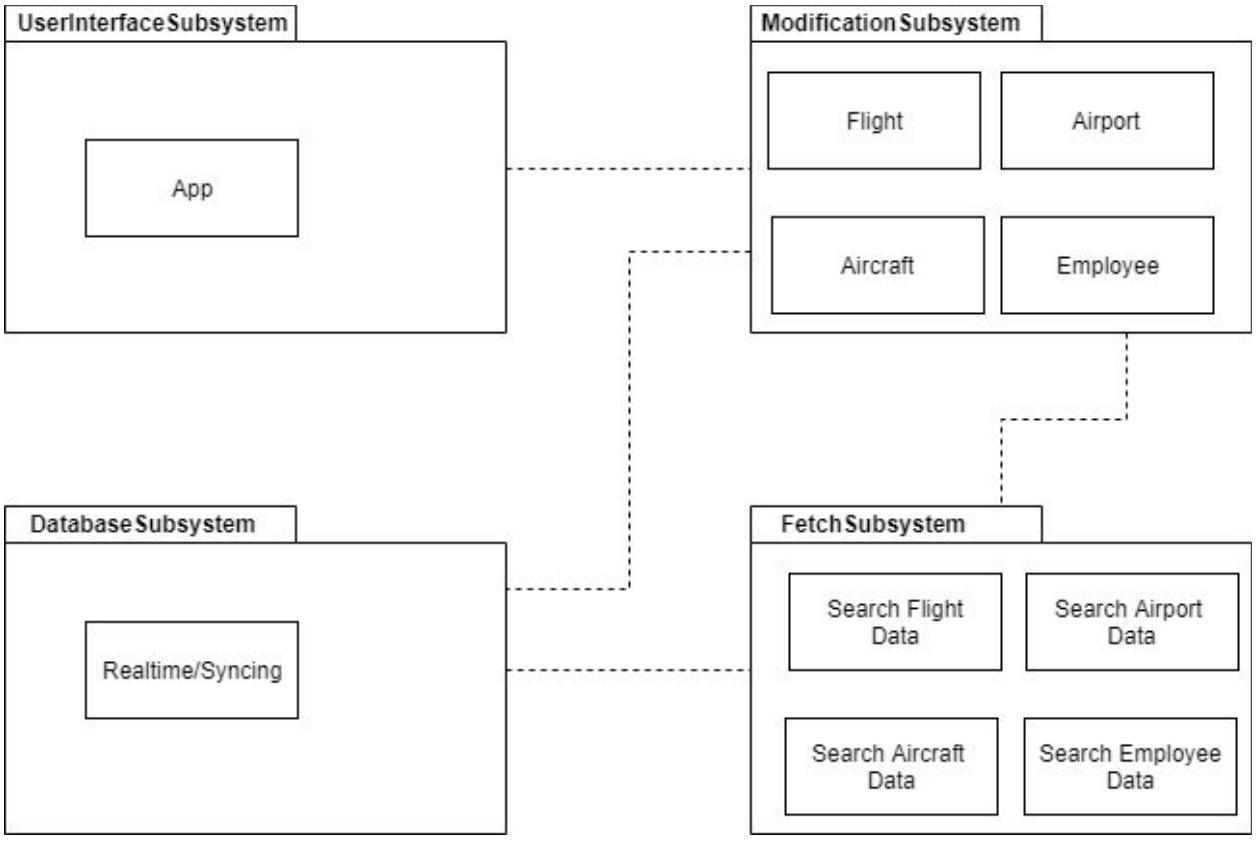
The System is divided into the following subsystems-

1) User Interface Subsystem provides an interface for the user to manage, fetch or display the desired information.

2) Database Subsystem: provides the ability to store and retrieve the data of CHA in real-time through the firestore firebase tools.

3) Modification Subsystem: provides the ability to create and modify both the employees and the flight's information.

4) Fetch Subsystem: provides the ability to search and display the existing data of both the flights and the employees.



Hardware/software mapping

Our model does not require hardware/software mapping.

Persistent data management

Our program uses Cloud Firestore, which is Firebase's new database for mobile app development. “It improves on the successes of the Real-time Database with a new, more intuitive data model.” (Firebase Real-time Database documentation, https://firebase.google.com/docs/database/rtdb-vs-firestore. Accessed Oct 31, 2018.)

Firestore stores data in documents, similar to JSON, then the documents are organized into collections. There are no tables or rows, instead, Firestore utilizes what is essentially many key-value pairs in a hierarchical model.

Access control and security

With Cloud Firestore, data validation happens automatically. Cloud Firesotre has its own security rules for mobile and web SDKs. Rules can be set to constrain queries so that if the user doesn’t have access to data that would be returned from a query, the entire query will fail.

Global software control

As there is potential for two clients to attempt retrieving information simultaneously, each server is made event-driven in order to avoid the race condition between them. Control flow is distributed within the Airline Schedule System and each service behaves on its own control flow.

Boundary conditions

Boundary conditions give the overview of how the system starts and what services need to be initialized first. The Firebase server handles all the data retrieval, so this server has to be started at the beginning as the client provides services to this server and retrieves the information from it. In our design, the Manager has control over the server. He/She is the one who manages all the databases. Once the server starts, the manager has to detect whether there is an error during the start of the server. With the help of Firebase analytics manager, we can detect the error and report the problem to the maintainer.

Proposed Software Architecture

no change

Subsystem Services

UserInterfaceSubsystem

UserInterfaceSubsystem provides users a platform to perform many tasks. It displays the existing information as well as acts as a medium to modify/create new entities. This subsystem calls methods, methods call query (which is a request from the user to modify) on ModificationSubsystem to modify/create the existing/non-existing information respectively from the database subsystem.
UserInterfaceSubsystem also calls methods that call query on fetchSubsystem to get the existing information from the firestore database subsystem.

ModificationSubsystem

ModificationSubsystem is assigned to make changes to the existing information related to a flight or an employee. It calls activities in UserInterfaceSubsystems to get desired change from the user and makes changes to the information in the database. While creating a new flight, this subsystem calls methods upon FetchSubsystem in order to use the existing information for assignments in flight. For example, ModificationSubsystem calls getEmployee() to assign the flight to the crew. While placing a new employee, this subsystem calls createEmployee() upon the DatabaseSubsystem to create a new entry in the document. ModificationSubsystem call getAirport () upon the databaseSubsystem to create new aircraft. ModificationSubsystem is called by methods which call query to create/change in DatabaseSubsystem.

FetchSubsystem

FetchSubsystem is assigned to fetch the existing data from the DatabaseSubsystem as a response to the query called by the method of the UserInterfaceSubsystem. ModificationSubsystem also calls method getEmployee() to get an employee which fetches the employee from the database with constraints applied as wells as calls method getAircraft() to get an aircraft which fetches aircraft from database. For example, while creating a flight, when getEmployee() is called, the fetchSubsystem will fetch the only employee that is eligible to work (non-functional requirement).

DatabaseSubsystem

DatabaseSubsystem is assigned to store the existing data, sync the updated data if the system was offline when the changes were made. The methods from ModificationSubsystem and FetchSubsystem called upon this subsystem are createEmployee(), getEmployee(), createFlight(), getFlight(), getEmployeeHrs() , createAircraft() ,

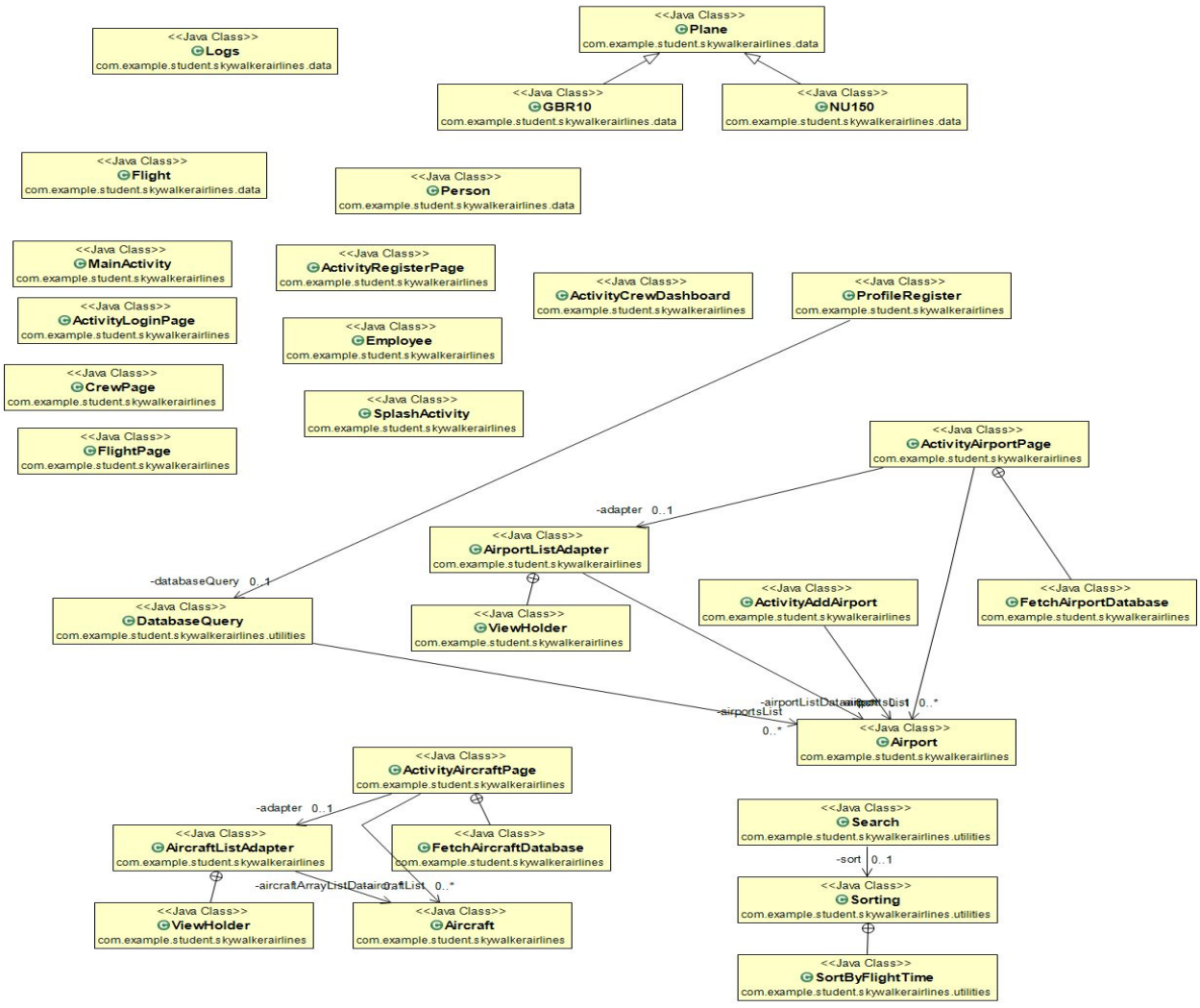
getAircraft(). The information that this subsystem provides is filtered based on the query called by the method from other ModificationSubsystem and FetchSubsystem. All the queries are invoked by the user's actions in UserInterfaceSubsystem.

Packages

User Interface Subsystem has an app as a package in this that a user can search for flight and is connected to the Modification Subsystem which has flight and Employee as packages. In the flight package, the manager can add a flight. In Employee package crew can add their hours. This modification subsystem is connected to fetch subsytem where the user can fetch the data from this subsystem like while searching for flight user can fetch the data from this subsystem. And lastly, modification and fetch subsystem is connected to the database subsystem which is real time and stores all kinds of data. The offline version database is used by the application.

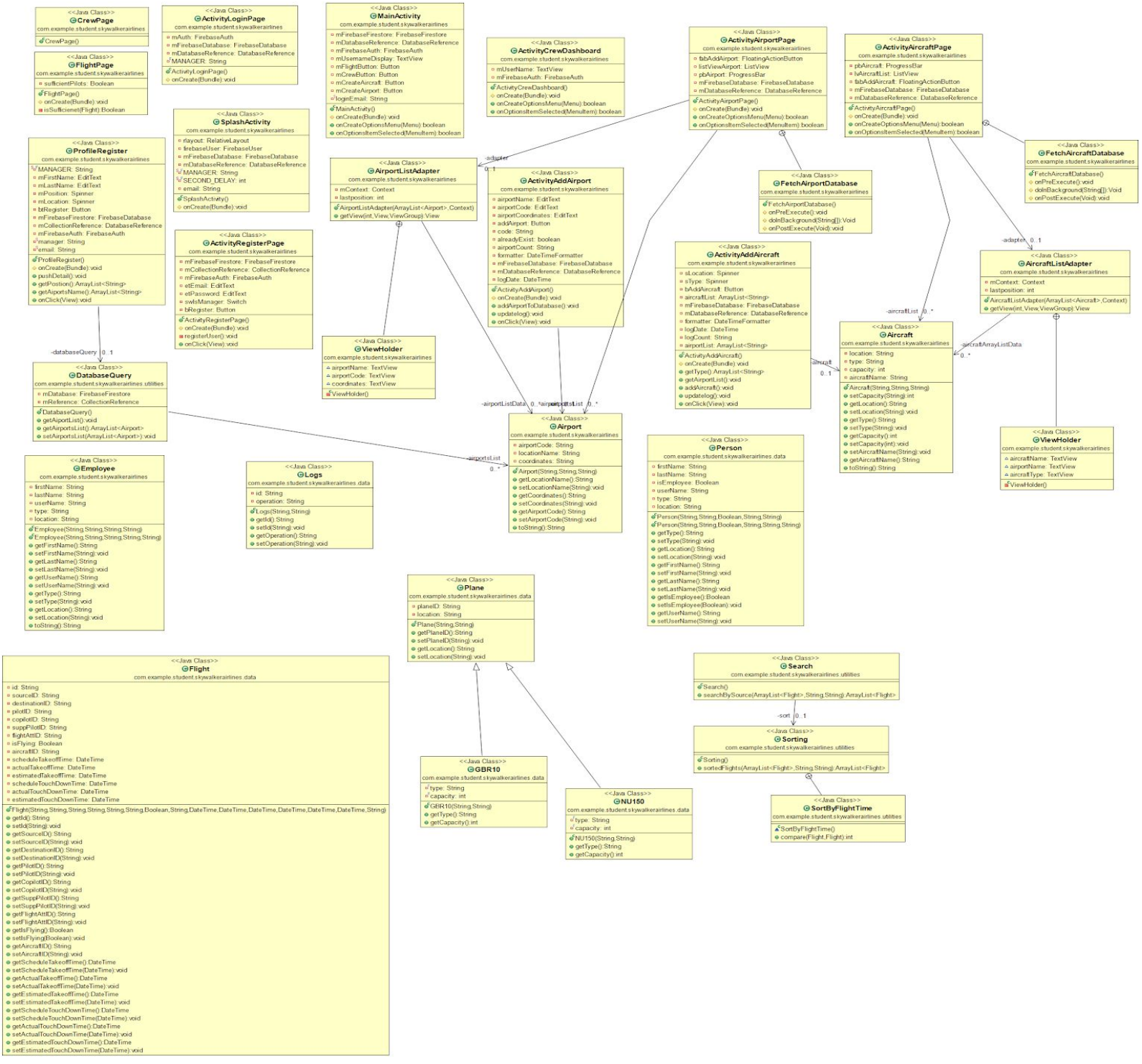
Class Interfaces

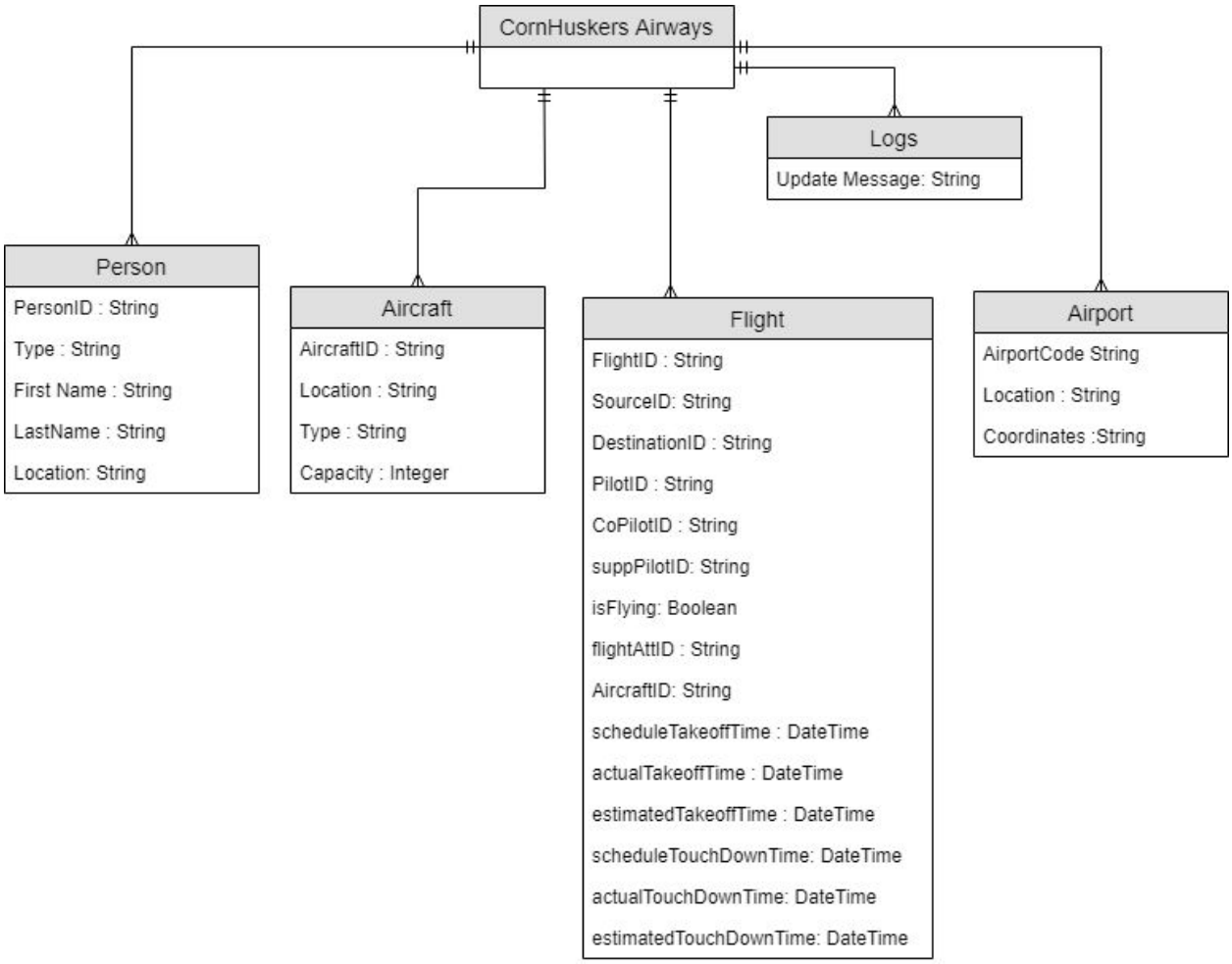
This diagram below illustrates the updated relationships between the class in our project.



Detailed Design

In the previous increment we just had a diagram depicting the tables in our database. in this increment we added a class diagram depicting all the classes and methods in our app.

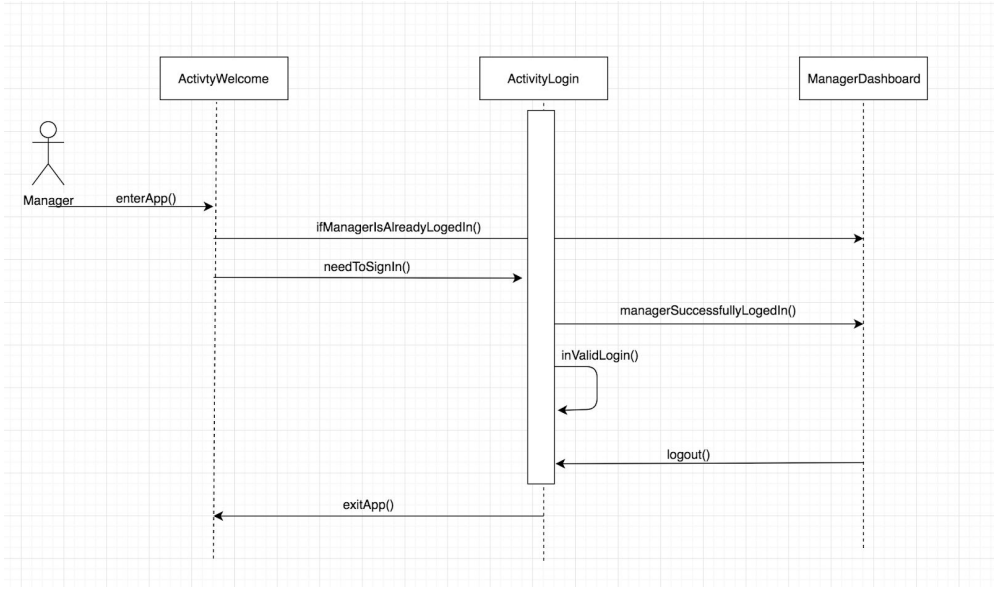




In FIrebase one collection(CornHusker Airways) is connected to documents(Person, Aircraft, Flight ,logs and Airport) which is connected to each other without any primary and foreign key. Each document can be initiated multiple times in the database.

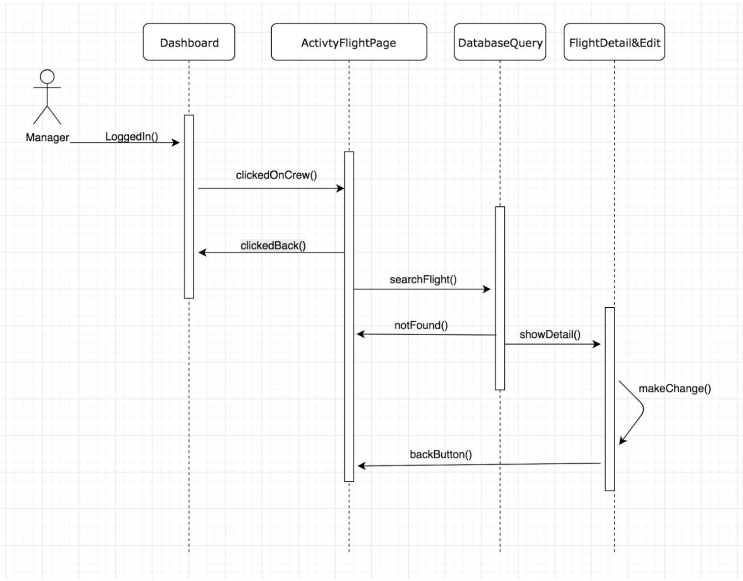
We are using firestore as the database along with the firestore authentication package for the manager. Cornhusker Airways is a collection containing documents Person, Aircraft, Airport. A person has the following attributes: First Name, Last Name, Clock in, Clock out., Aircraft has the following attributes: Aircraft, Type, Capacity, Speed. The airport has the following attributes: Airport ID, Name, Latitude, Longitude.

Authentication Sequence Diagram



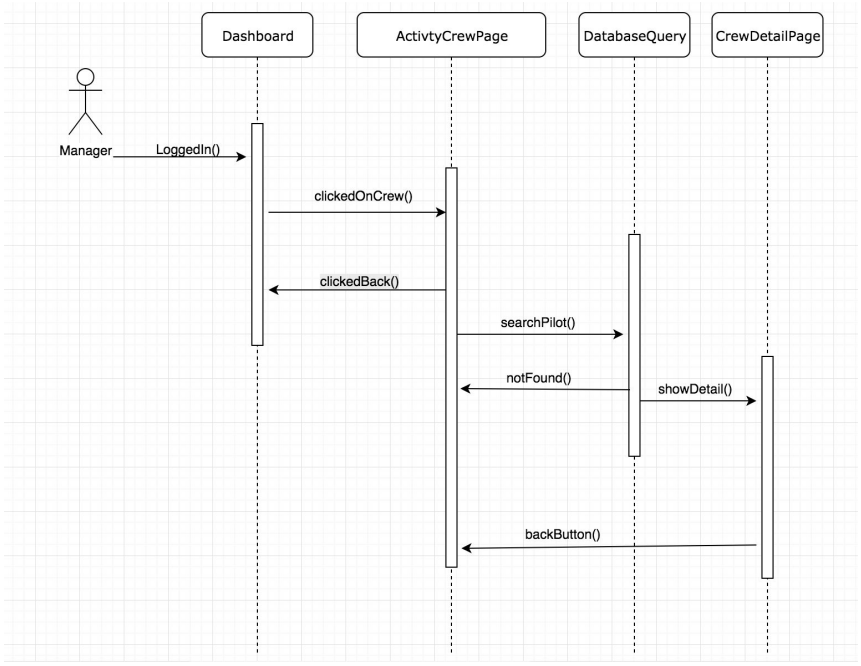
The method name we used is a high-level representation of the function. In this case, manager is trying to reach the dashboard and get authenticated. As the manager open the application the Firebase Authentication will check if the user is already logged in if not it will give an attempt to enter credential in Activity Login. If the credential doesn't match to any of case either manager or crew the system will ask to enter correct authentication. Once the manager is inside of the dashboard he/she can log out anytime.

Getting detail and edit flight sequence Diagram



The method name we used are high level representation of the function. In this case manager is trying to flight data from the database. The first manager needs to be authenticated to reach to the dashboard. Then click on flight button and ActivtyFlightPage will pop up and will show every flight in the database. If the manager wants to access any particular flight he can search in the search bar. Then the detail of that page will pop and manager can make changes to it.

Getting crew details subsequence diagram



The method name we used is a high-level representation of the function. In this case, the manager is trying to flight data from the database. The first manager needs to be authenticated to reach to the dashboard. Then click to crew button and ActivityCrewPage will pop up and will show each crew in the database. If the manager wants to see the detail of particular crew he can click in card and detail of that crew will pop up.

GLOSSARY

Actual Takeoff/Landing – the precise time at which the aircraft takeoff/landed.
Administrator – Someone who has all the access over the system.
Aircrafts – vehicles operated by the Airways.
Captain – a senior pilot who commands the crew of an airplane.
Estimated Takeoff/landing – predicted the time at which the aircraft might takeoff/land.
Functional Requirements – describes the functionality of the system.
Grounded – An aircraft not being used for a while.
Non-Functional Requirements – User level requirements including usability, reliability and implementation.
Pilot – a person who flies or is qualified to fly an aircraft.