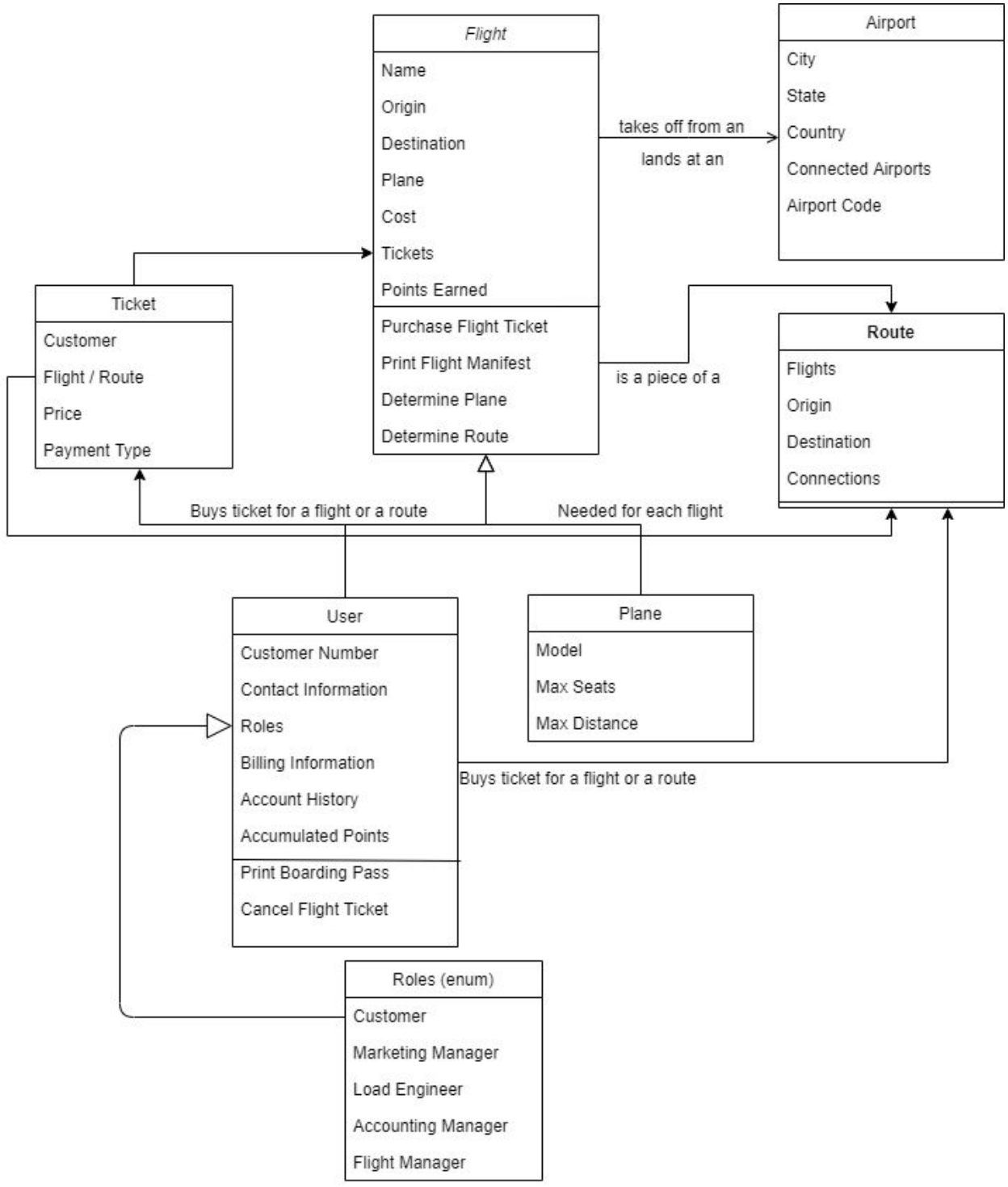
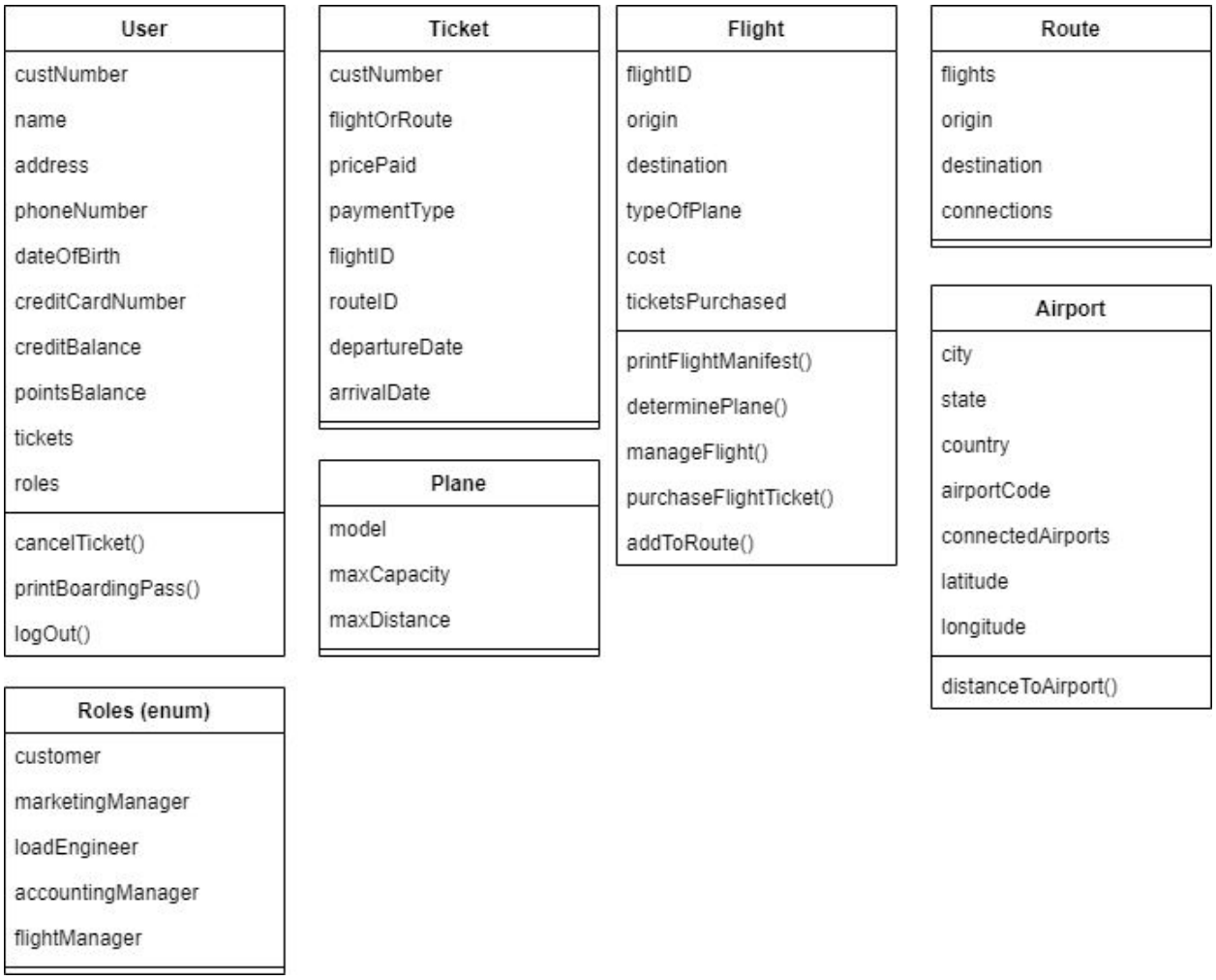
EECS 3550-001: Software Engineering
Air 3550
Group 11
Design Document
03/07/2021
James Golden, Edward Walsh, & Quinn Kleinfelter

Architecture & Component Level Design



This UML Diagram details the relationship between a Flight, Airport, User, Plane, Ticket and the roles enum in our project. For example, a User with the roles Customer and Marketing Manager, can purchase a Ticket, print their boarding pass, or determine which plane will be used for a

flight. This flight has both an origin and destination, which are airports, that are connected to each other.



Pictured here is the individual component level design, including more specific field and method names for each of our main functionality classes, not related to user interface. Not pictured in this diagram are methods to get or set any of the associated fields, though these will be available to appropriate users as needed. The “tickets” field in a user object will be used to store all flight tickets a user has purchased, past and present, and will be used to handle their account history. Meanwhile the roles a user has (handled by the roles enumeration), will determine what things they can do such as adjust flights, print manifests, etc. Other connections between objects are available such as flights being a part of routes, and planes and airports being pieces of a flight.

Planes

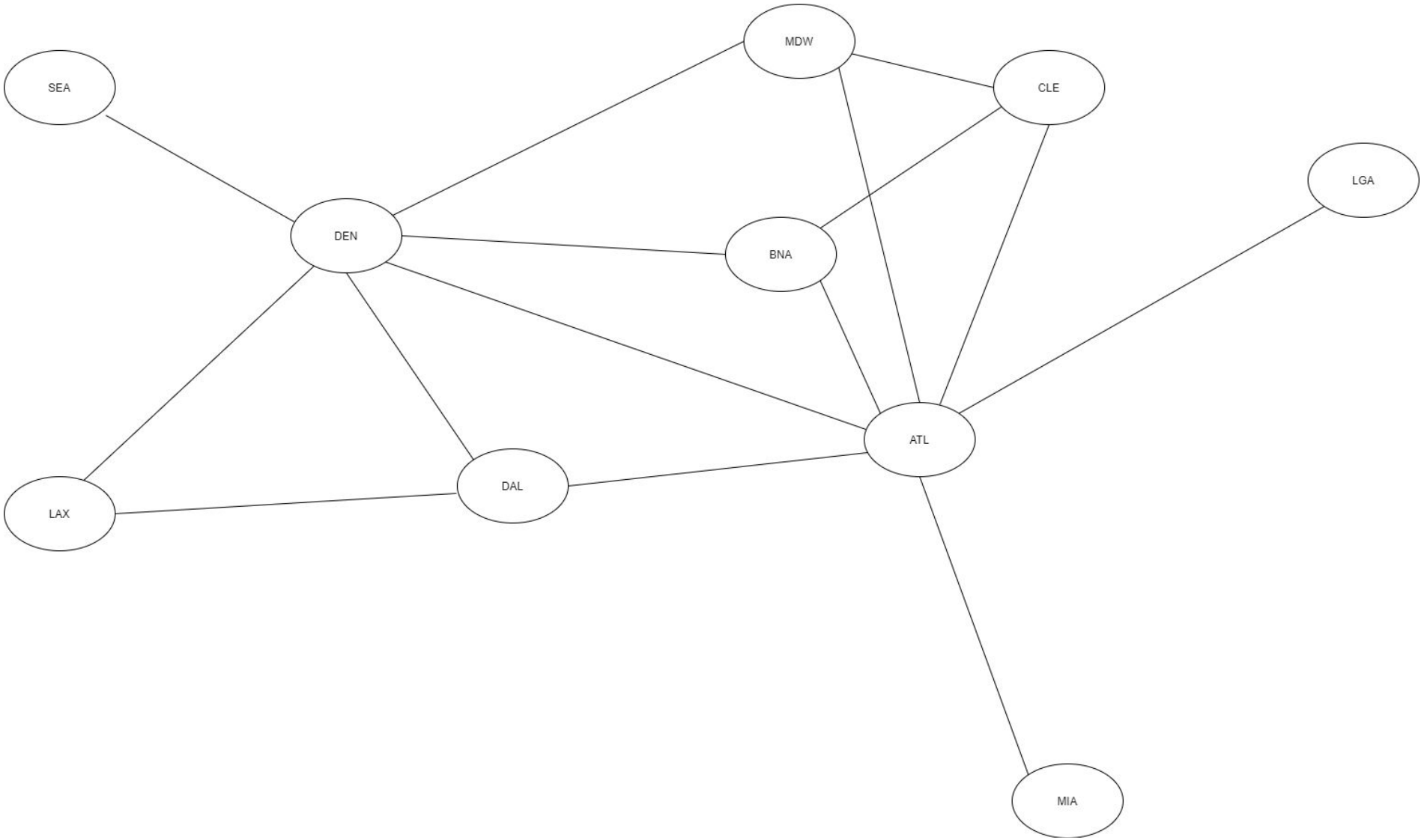
Plane
Model: 747
Max Seats: 410
Max Distance: 7,730 nautical miles

Plane
Model: 737
Max Seats: 220
Max Distance: 3,000 nautical miles

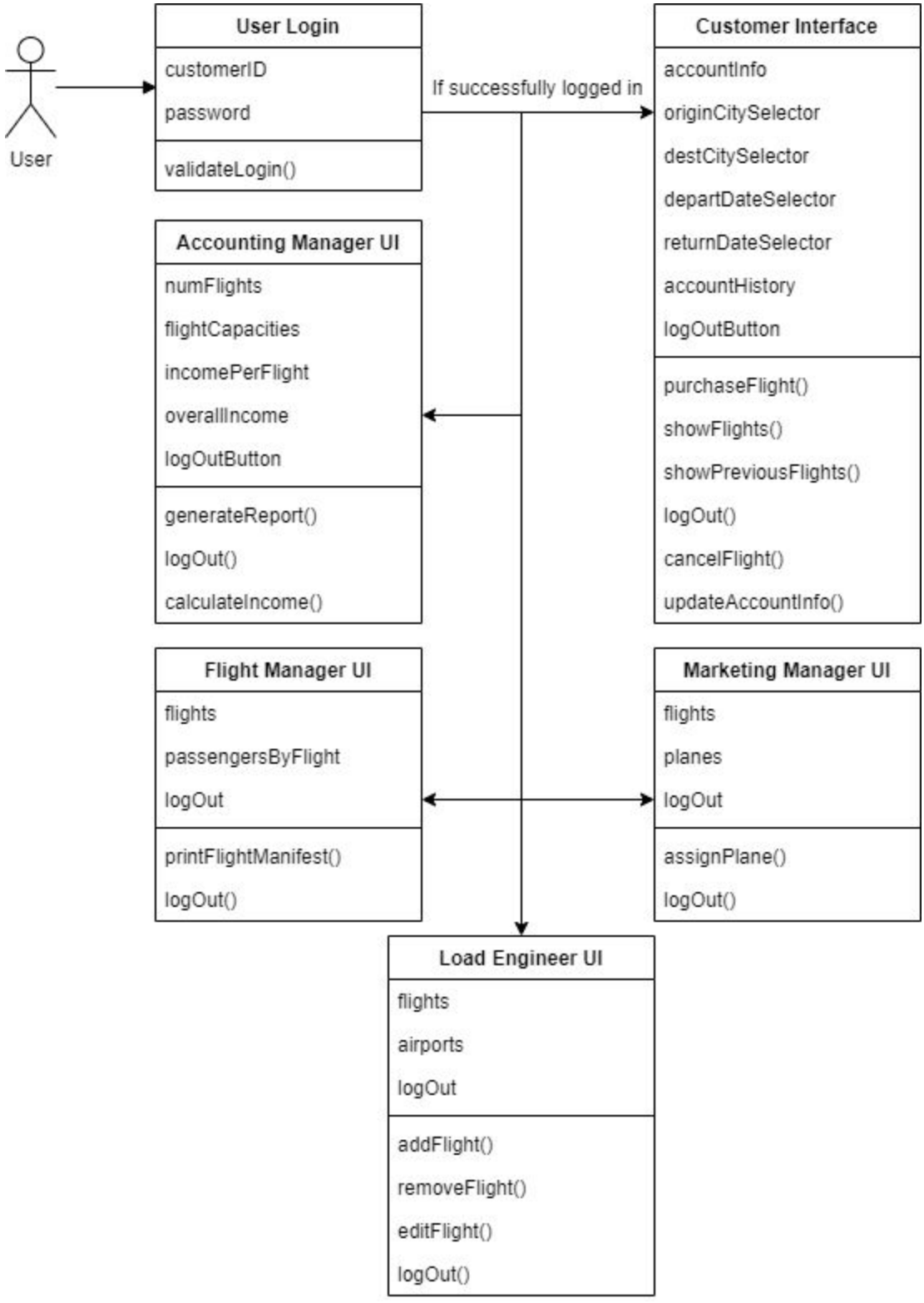
Plane
Model: 757
Max Seats: 199
Max Distance: 3,370 nautical miles

Airports

Hartsfield-Jackson International Airport	ATL	Atlanta	GA
Los Angeles International Airport	LAX	Los Angeles	CA
Chicago Midway International Airport	MDW	Chicago	IL
Dallas Love Field	DAL	Dallas	TX
Denver International Airport	DEN	Denver	CO
LaGuardia Airport	LGA	New York	NY
Miami International Airport	MIA	Miami	FL
Seattle–Tacoma International Airport	SEA	Seattle	WA
Nashville International Airport	BNA	Nashville	TN
Cleveland Hopkins International Airport	CLE	Cleveland	OH


MDW
SEA
CLE
LGA
DEN
BNA
ATL
DAL
LAX
MIA

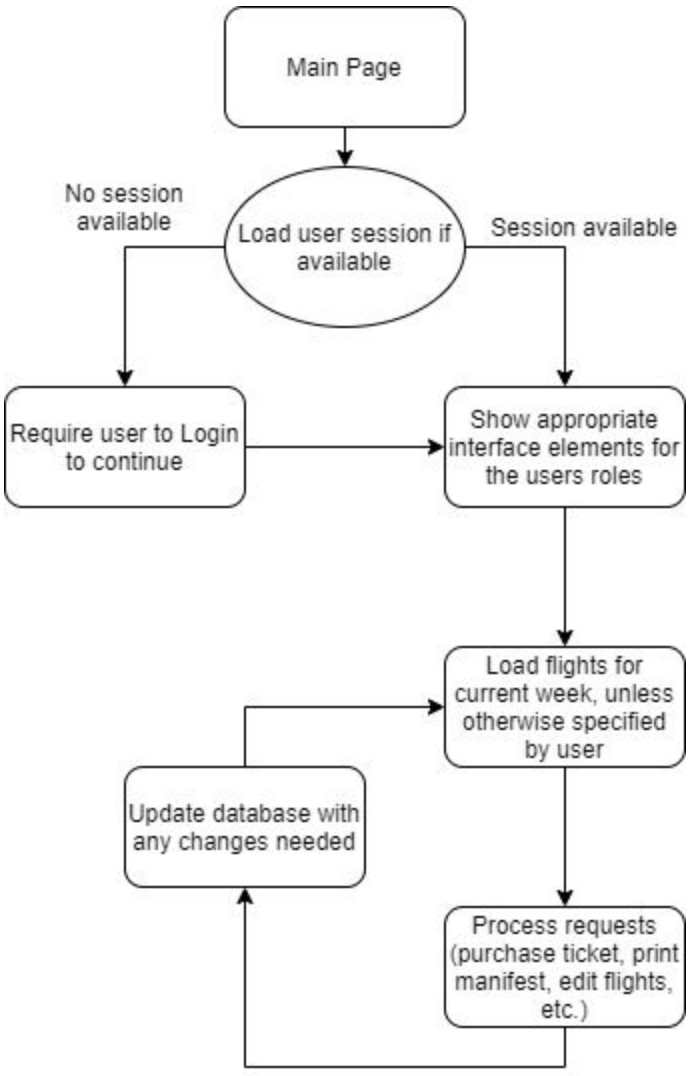
Interface Design



This diagram displays the various interface components that exist for our users. First, all users must login using their customerID and password, from there depending on the roles their user has they are shown various UI components to allow them to accomplish their goals. For example, someone who is both a customer and flight manager can access their accountInfo, select information about a desired flight, purchase a flight ticket, and print flight manifests for any flight as needed.

Data Flow Architecture

This chart shows the general data flow within our system, from a user opening the application to using it for any purpose, either purchasing tickets, editing flights or any other required tasks. Requests are processed by several smaller pieces as detailed in the requirements document as well as in the UML diagrams provided in this document.



Determining Routes Algorithm

Since the airports and flights between them can be represented graphically, graph algorithms seem to be the most appropriate solution. To determine all possible routes, a modified Breadth First Search should work well. The algorithm will use a simple first in first out queue and flights between cities will serve as adjacency lists.

Pseudo-code:

```
determineRoutes(origin city (O), destination city(D))
1       for each city
2               set to unexplored and number of connection to infinite
3       set number of connections to 0 (O.connections = 0) for origin city and mark as seen
4       enqueue origin city (O)
5       while queue is not empty
6               look at first city in queue (dequeue) call this city U
7               if U.connections >=3, break out of loop
8               for each flight from city (U)
9                       get connected city (C)
10                      if connected city is destination (C=D)
11                              C.connections = U.connections + 1
12                              backtrack through flights to create route and save to list
13                      else if connected city is unexplored (not seen or explored)
14                              mark as seen
15                              C.connections = U.connections + 1
16                              set preceding city of C to U
17                              enqueue connected city C
18              set city (U) to explored
19      return list of routes
```