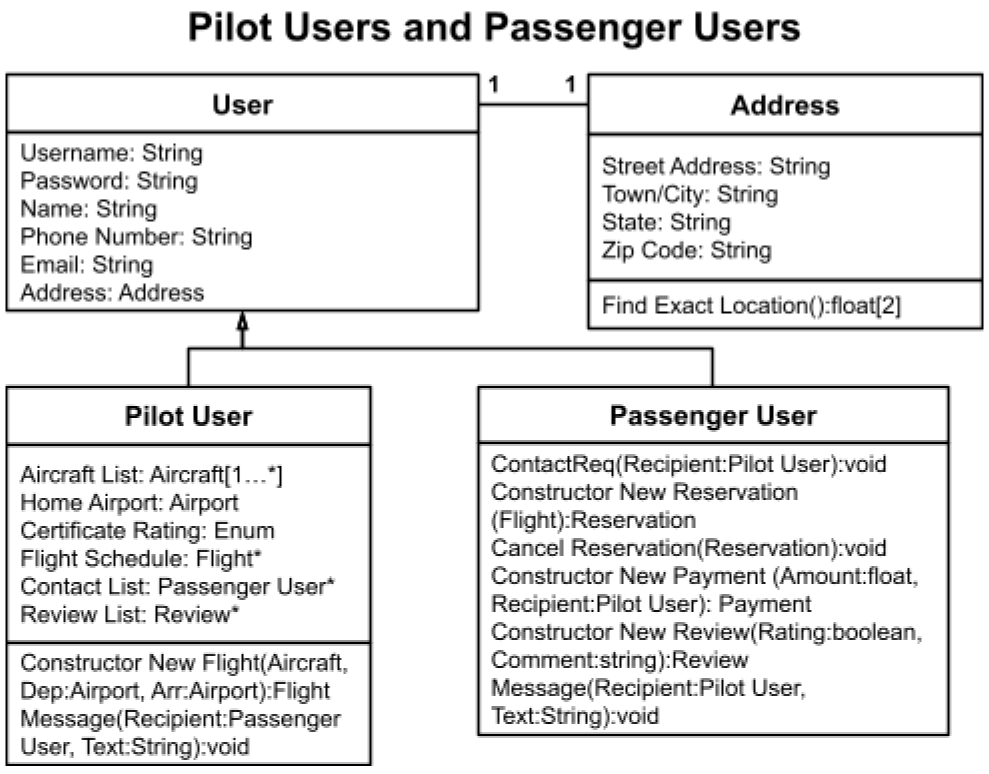
This document includes the diagrams and accompanying descriptions for an accurate representation of the domain modeling scheme used in Grey Matter's “FlyAway” product. The purpose of this document is to help familiarize external observers with the object oriented structure of the application, specifically the characteristics and attributes of each class and the relationships between classes. FlyAway utilizes React as a development framework for web-based applications, which in turn makes use of HTML, CSS and Javascript for frontend functionality and external databases such as firestore for backend functionality. In this context, classes therefore refer to the higher-level classes which define the functioning of the program as a whole and not merely to the user interface components which are the building blocks of React.

This document includes four UML diagrams describing classes and depicting relationships between them:

- The first UML diagram defines the attributes of a user of the program and the subclasses that inherit from this definition.
- The second UML diagram defines the attributes of other classes not directly related to or dependent on the user or its subclasses, but does not make an attempt to show how they are related to each other.
- The third UML diagram shows how the primary classes of the system interact with each other during the reservations/booking process, with particular emphasis on how a pilot is responsible for instantiating a flight but a passenger is responsible for instantiating a booking.
- The fourth UML diagram describes other interactions between pilot and passenger accounts, specifically payment processing and the review system.

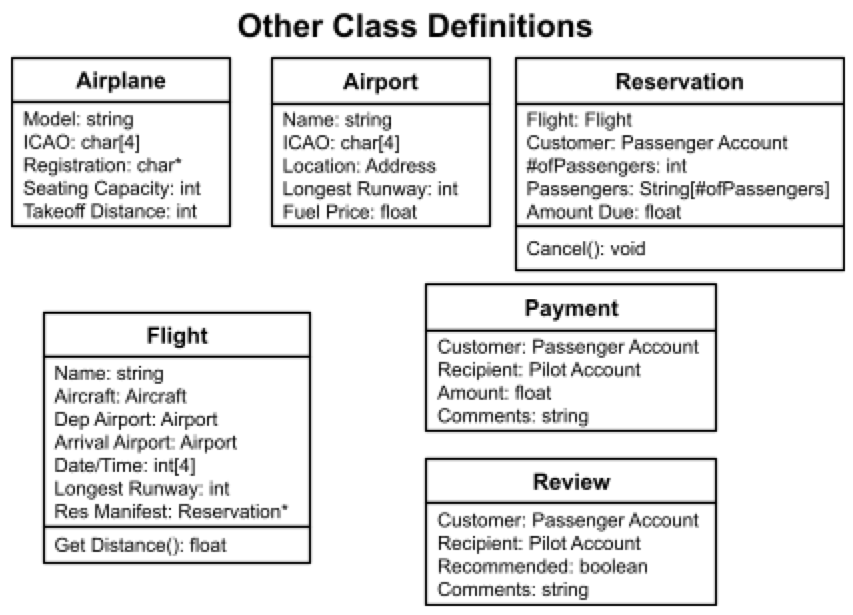
The first Domain Modeling UML diagram shows the hierarchical relationship between Pilot Users, Passenger Users, and their shared “User” superclass. A user is, at its most basic level, defined as a Username and a Password with a series of attributes specifying that user’s contact information. Notice that the various fields that specify a person’s specific address - such as Street, Town/City and Zip Code - are exported to a separate class rather than being included as a field in the base user. There are three primary reasons for this. Firstly, it increases the level of abstraction in the User class definition by exporting a variety of potentially repetitive attributes to a separate class. In addition, separating the user’s home address from their other information makes it easier for the user to choose to keep this information private, thus enhancing the security of the application. However, most importantly, the fact that FlyAway is a web application generally means that there isn’t a consistent way to track a passenger user’s location and recommend pilots that are geographically nearby. Having the address as a separate field can allow the application to calculate a person’s geographic location on a cartesian grid from that person’s provided address.

The two subclasses of “user” are the “pilot user" and the “passenger user”. The passenger user inherits no additional fields and attributes from its superclass, but does have a variety of methods that allow a passenger to instantiate various objects by calling their constructors. These include generating new reservations, payments, and reviews, as well as sending contact requests to a pilot account and canceling reservations that they no longer are able to fulfill. In addition, the Pilot user has a list of at least one aircraft, a home airport, a flight schedule (instantiated as a list of flights), a contact list (of passenger users) and a review list. In addition, the pilot has the unique ability to instantiate a new flight. Lastly, both the passenger user and the pilot user are able to send and receive text messages, but only from each other and only with accounts of the opposite type who have agreed to talk to each other.


Pilot Users and Passenger Users
User
Username: String
Password: String
Name: String
Phone Number: String
Email: String
Address: Address
1
1
Address
Street Address: String
Town/City: String
State: String
Zip Code: String
Find Exact Location():float[2]
Pilot User
Aircraft List: Aircraft[1…*]
Home Airport: Airport
Certificate Rating: Enum
Flight Schedule: Flight*
Contact List: Passenger User*
Review List: Review*
Constructor New Flight(Aircraft, Dep:Airport, Arr:Airport):Flight
Message(Recipient:Passenger User, Text:String):void
Passenger User
ContactReq(Recipient:Pilot User):void
Constructor New Reservation (Flight):Reservation
Cancel Reservation(Reservation):void
Constructor New Payment (Amount:float, Recipient:Pilot User): Payment
Constructor New Review(Rating:boolean, Comment:string):Review
Message(Recipient:Pilot User, Text:String):void

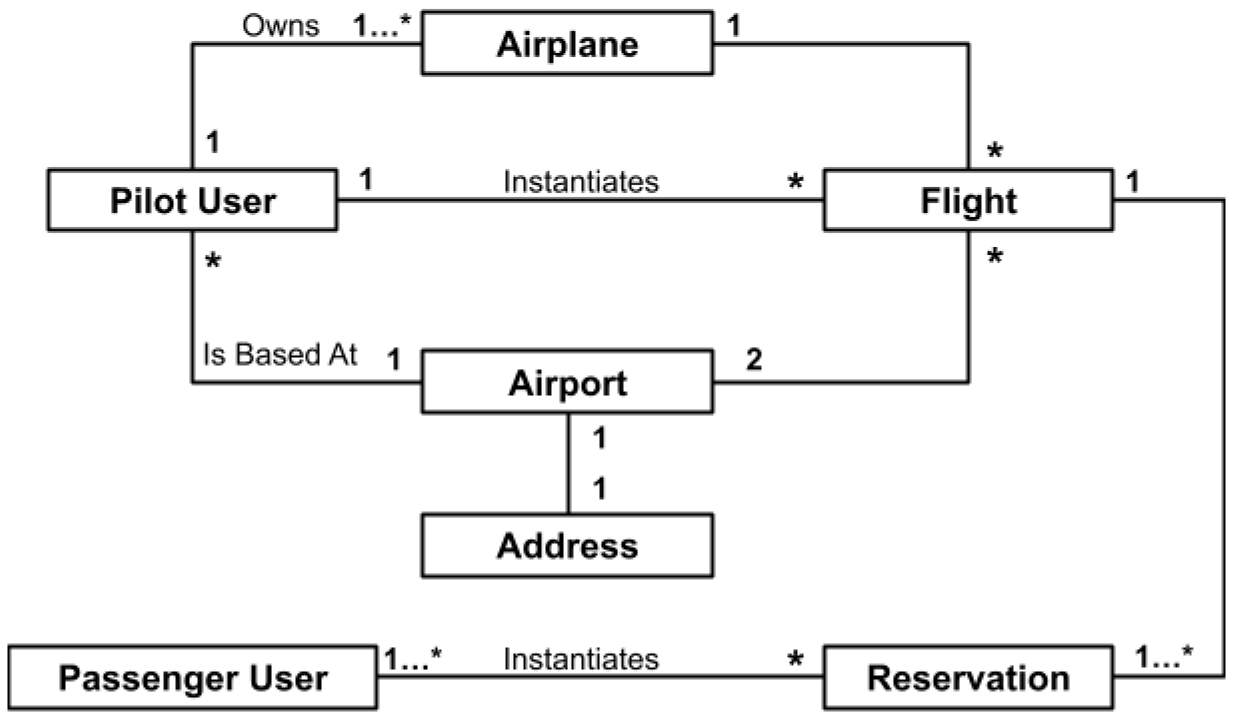
The second Domain Modeling UML diagram lays out the remaining primary classes used by FlyAway without showing how they are related to each other. Most of these classes have only a handful of attributes and few methods associated with them, and as such most of them are fairly self explanatory. Only the following items may not be immediately obvious when reading the diagram:

- “ICAO” stands for the “International Civil Aviation Organization”, a division of the United Nations responsible for regulating International Flight, which includes issuing four-letter identification codes for airports and aircraft types. Ideally FlyAway will eventually contain a reference file containing every make and model aircraft and the associated identification code, as well as the other necessary statistics of the aircraft. This code is implemented as a four-character string array. For the time being, we will simply rely on the pilot to enter this information on their own initiative. A similar rule applies with airports..
- Notice that the “airport" class references the “address” class mentioned previously. Airport location will need to be calculated in order to estimate flight time and distance between a passenger and viable potential pilots, so it makes sense to use the address class in this manner.
- The “reservation” class includes a “Cancel()” method to allow a passenger to cancel their reservation.
- The “flight” class includes a “Get Distance()” method to calculate the distance between the arrival and departure airports.
- The “review” class stores the actual recommendation rating as a true/false boolean rather than the traditional five star rating. This is in compliance with a previous design decision to use an “up or down” rating system rather than a five star rating system. We believe that this will make reviews less complex and won’t unfairly punish pilots who get mixed reviews.

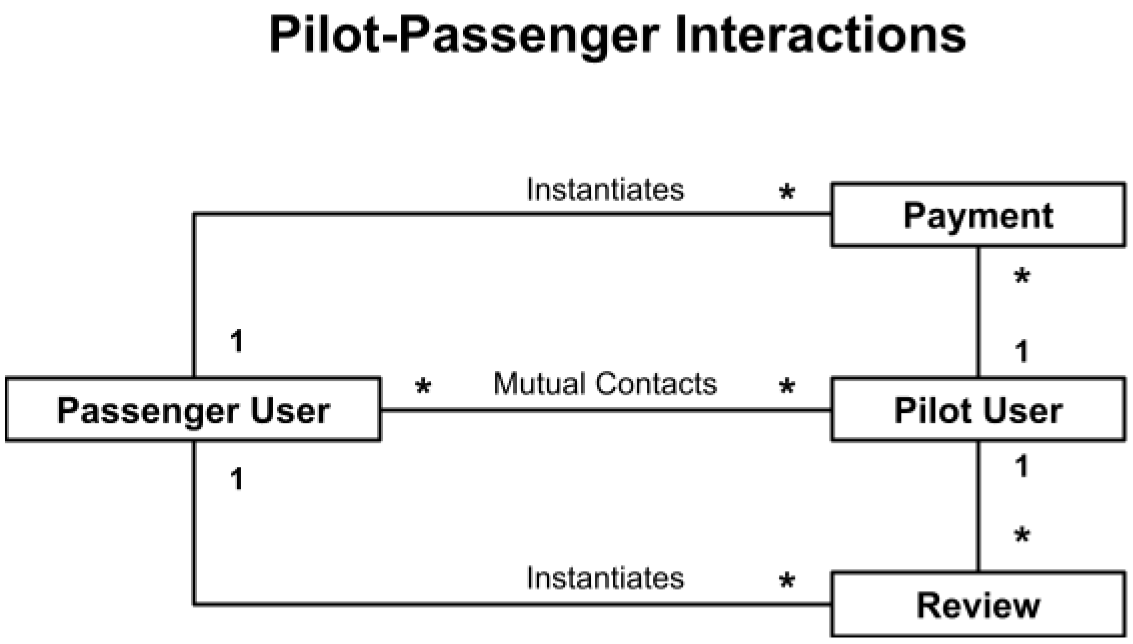


The third Domain Modeling UML diagram describes the interactions between classes that are required to take place in order to facilitate the reservation of a flight. Two processes are required to take place in order for this to happen - the pilot must take his or her own initiative to generate a new flight, which will include an airplane, two airports and a pilot flying, as well as an array of passenger reservations called “Res Manifest,” whose size is equal to the number of passengers that the aircraft type can carry on a particular flight. When a flight is initially instantiated, the array of passengers starts out as containing only null values. When a passenger decides to participate in the flight, they instantiate a “reservation”, which fits into the “Res Manifest” array. This is structured to allow a pilot whose airplane can seat multiple passengers (such as twin engine or turboprop aircraft) to fit multiple passengers with different reservations on the same flight. Notice also that, in the previous diagram, the number of passengers that may be included in a reservation is not fixed at zero, and is not tied to any information stemming directly from a passenger account. This means that the people who travel on flights generated using FlyAway need not be users of the FlyAway app. A passenger user may reserve a booking for themselves, travel with other people who are non users, or even pay for a flight entirely on behalf of other people without intending to travel on the aircraft themselves. This reduces complexity and allows non-users to benefit from the application as well by pooling their resources in a flexible manner.

Flight Reservation System



The fourth and final Domain Modeling UML diagram depicts interactions between pilot and passenger accounts that are not directly included in the booking and reservation process. Three main features are included in this group - the payment system, the review system, and the messaging system. The last of these features, the messaging system, is not included in the domain modeling diagram because it is a direct interaction between two classes already defined - the pilot user and the passenger user - and therefore does not need to be described in great detail. The remaining features - the payment system and the review system - are similar enough in their domain relationships that they essentially mirror each other in the diagram, even though there are some important functional differences on the backend. In the case of a payment, the passenger user may “instantiate” a new payment by calling the constructor for the payment class. This class includes all the necessary information for the payment to be processed completely, including the payment amount, banking and financial information, and additional user-specified comments. A pilot user is not given the ability to instantiate a payment class because they are not presumably going to be making payments to passenger users as part of their regular usage of the application. In a similar way, the passenger user has the ability to “instantiate” a new review, which is then anonymously visible to the pilot and any other passenger users who view the pilot’s account page.


Pilot-Passenger Interactions
Instantiates
*
Payment
*
1
1
Passenger User
*
Mutual Contacts
*
Pilot User
1
1
*
Instantiates
*
Review