Final Report
Project Part 6

Team #06: Flight Scheduler
Raghav Sharma
Byron Becker
Johnathan Kruse
Brian Chung
Cory Morales

Question 1

List the features that were implemented (table with ID and title).

User Requirements				
ID	Requirement	Topic Area	Actor	Priority
PAS - 002	Lookup flight status via airline name	Search	Passenger	2
PAS - 003	Lookup flight status via flight number	Search	Passenger	2
AIR - 003	Can request to add flight of their own	Request	Airline	2
PORT - 001	Login as airport admin	Authentication	Airport	1
PORT - 003	Can cancel any flight	Adjustment/ Cancellation	Airport	2
PORT - 005	Can cancel flights in a time window	Cancellation	Airport	2

Functional Requirements				
ID	Requirement	Topic Area	Actor	Priority
FUN - 001	Schedules flights due to their specific inputs	Scheduler	Airport/Airline	1
FUN - 002	All flight information is stored in database	Database/ Storage	Airport/Airline	1
FUN - 003	All actions by the airport or	Database/ Adjustments/	Airport/Airline	2

	airline (additions, cancellations, etc.) update the database and return a response (i.e. confirmed, rejected)	Response		
FUN - 005	Upon flight change, notify appropriate users	Notification/ Users	Airport/Airline/ Passenger	4

Non-Functional Requirements				
ID	Requirement	Topic Area	Actor	Priority
NFUN - 002	One user can interact with the system at a time	Usability/Users	Airport/Airline/ Passenger	1
NFUN - 003	Works on small datasets (<500MB)	Reliability/ Database	Airport/Airline/ Passenger	2
NFUN - 004	Will return the result of any action within 15 seconds	Performance	Airport/Airline/ Passenger	2

Question 2

List the features were not implemented from Part 1 (table with ID and title).

User Requirements				
ID	Requirement	Topic Area	Actor	Priority
PAS - 001	Login via Guest user	Authentication	Passenger	1
AIR - 001	Login as particular airline	Authentication	Airline	1
AIR - 002	View status of all flights	Search	Airline	3
AIR - 004	Can view the accepted status of their own flights	Request/Search	Airline	2
AIR - 005	Can cancel their own flights	Cancellation	Airline	3
PORT - 002	Can view the status of all flights	Search	Airport	3
PORT - 004	Priority can be added to a specific airline	Priority	Airport	4
PORT - 005	Can cancel time window groundings	Cancellation	Airport	3

Functional Requirements				
ID	Requirement	Topic Area	Actor	Priority
FUN - 004	Flight history deleted from database after new day	Database	Airport/Airline/ Passenger	3

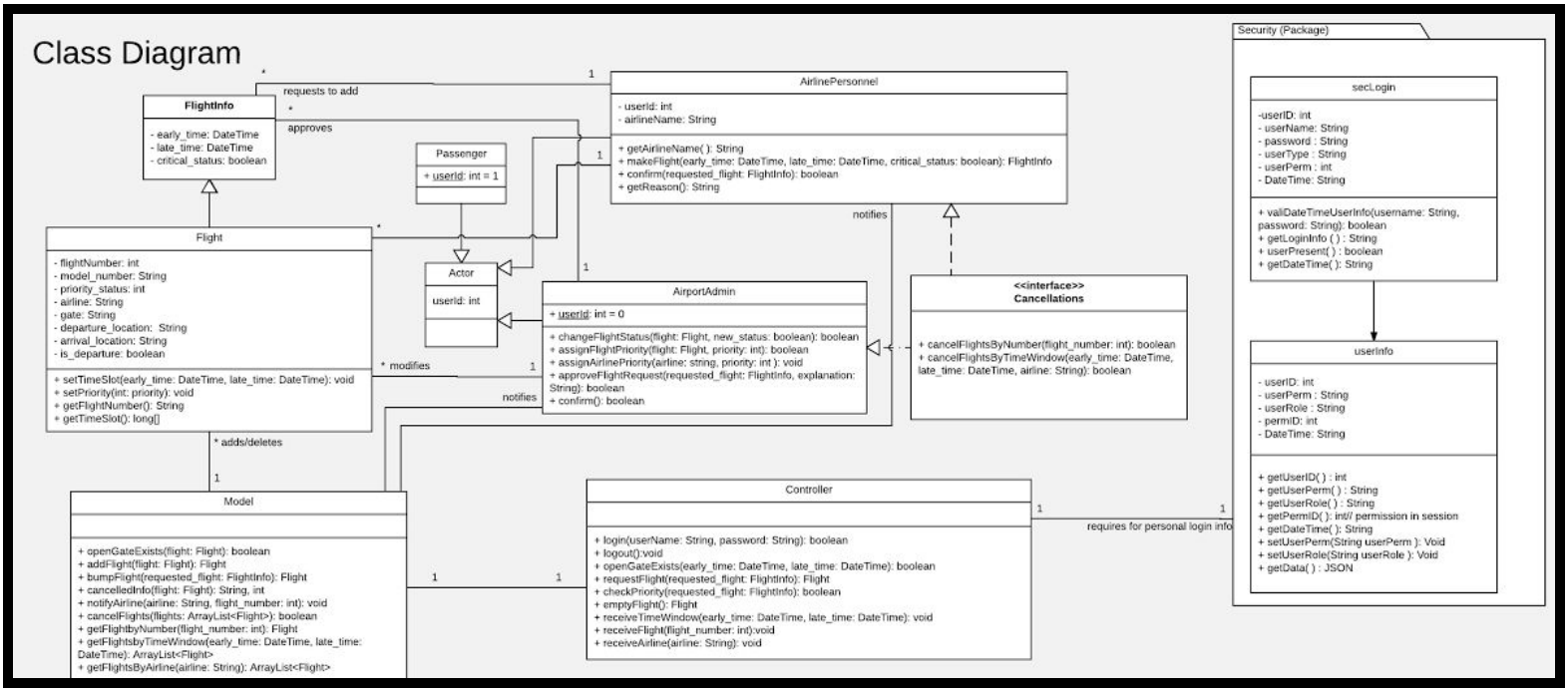
Non-Functional Requirements				
ID	Requirement	Topic Area	Actor	Priority
NFUN - 001	Actions warranting a response use a new page	Usability/Views	Airport/Airline/ Passenger	2
NFUN - 005	Developed with the intent of being used on non-mobile platforms (OSX, Windows, Linux)	Supportability	Airport/Airline/ Passenger	1

Extra Requirements				
ID	Requirement	Topic Area	Actor	Priority
EX - 001	Multiple users can interact with the system at once	Usability/Users	Airport/Airline/ Passenger	2
EX - 002	Emergency landing procedures	Usability/ Performance	Airport/Airline/ Passenger	3
EX - 003	Passengers have their own login information	Authentication	Passenger	1
EX - 004	Passengers are notified about changes to their flights	Notification/ Usability/Users	Passenger/ Airline	4

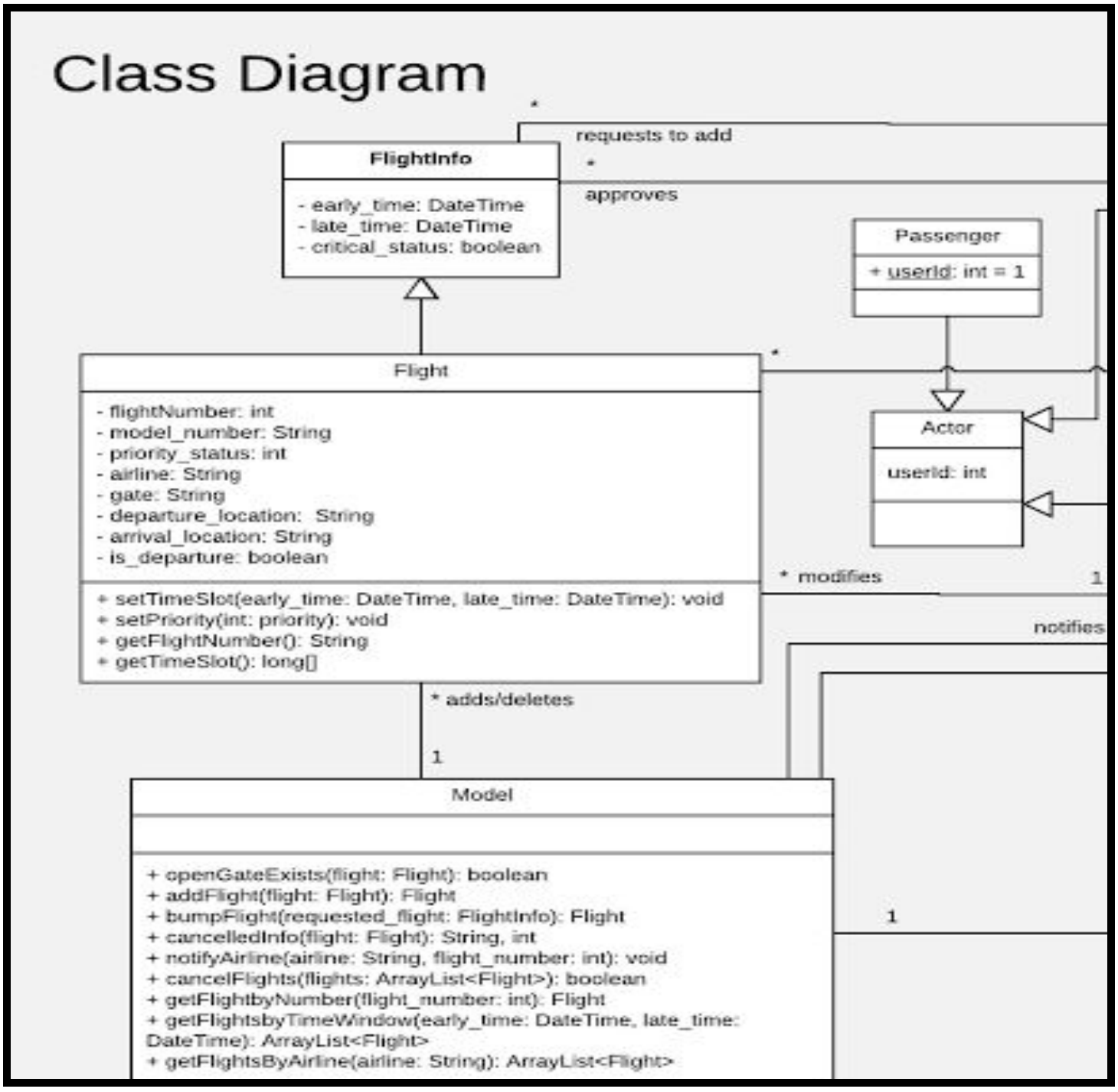
Question 3

Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

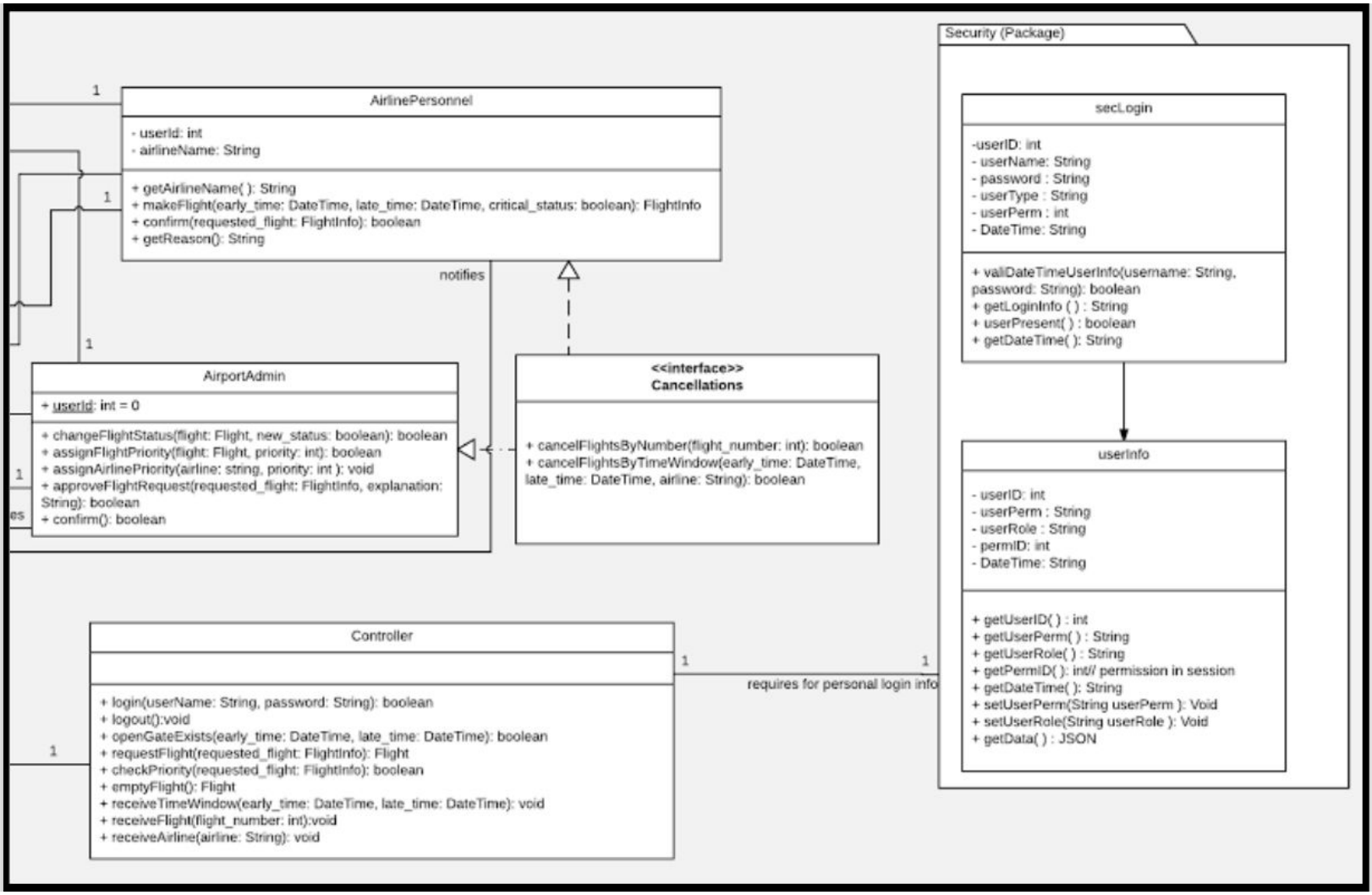
Part 2 Class Diagram:



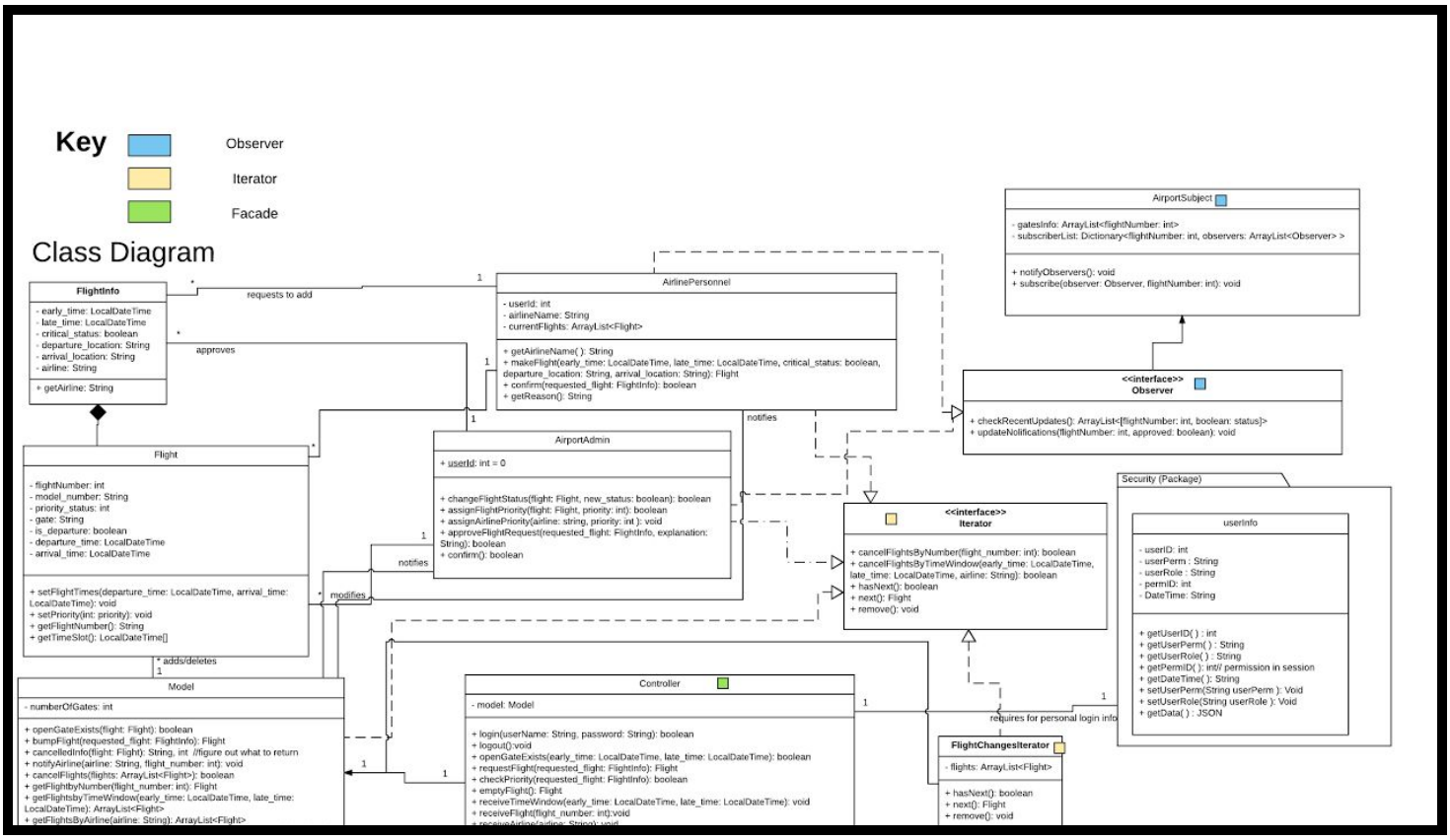
Part 2: Zoomed In
Image 1 of 2



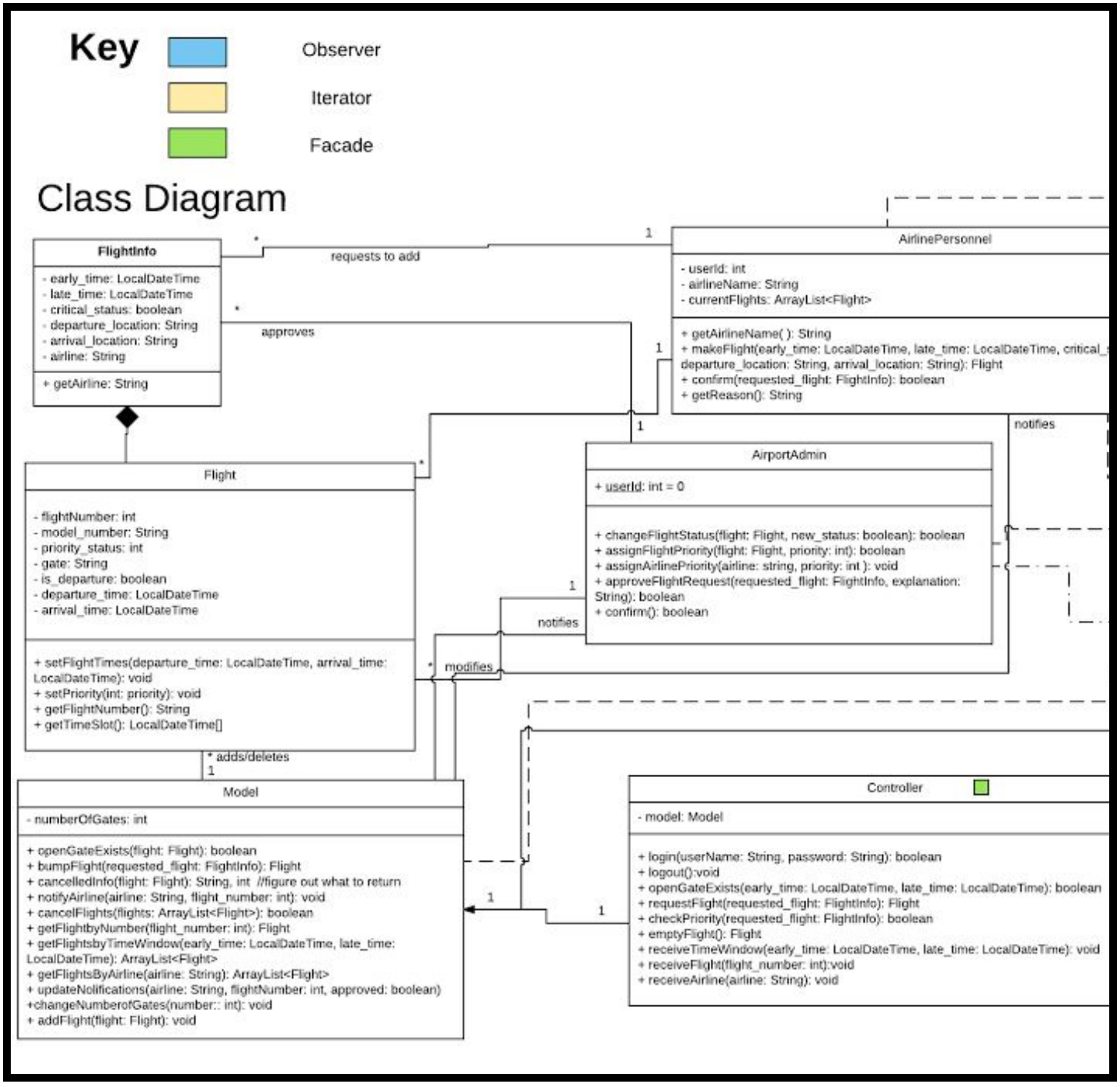
Part 2: Zoomed In
Image 2 of 2



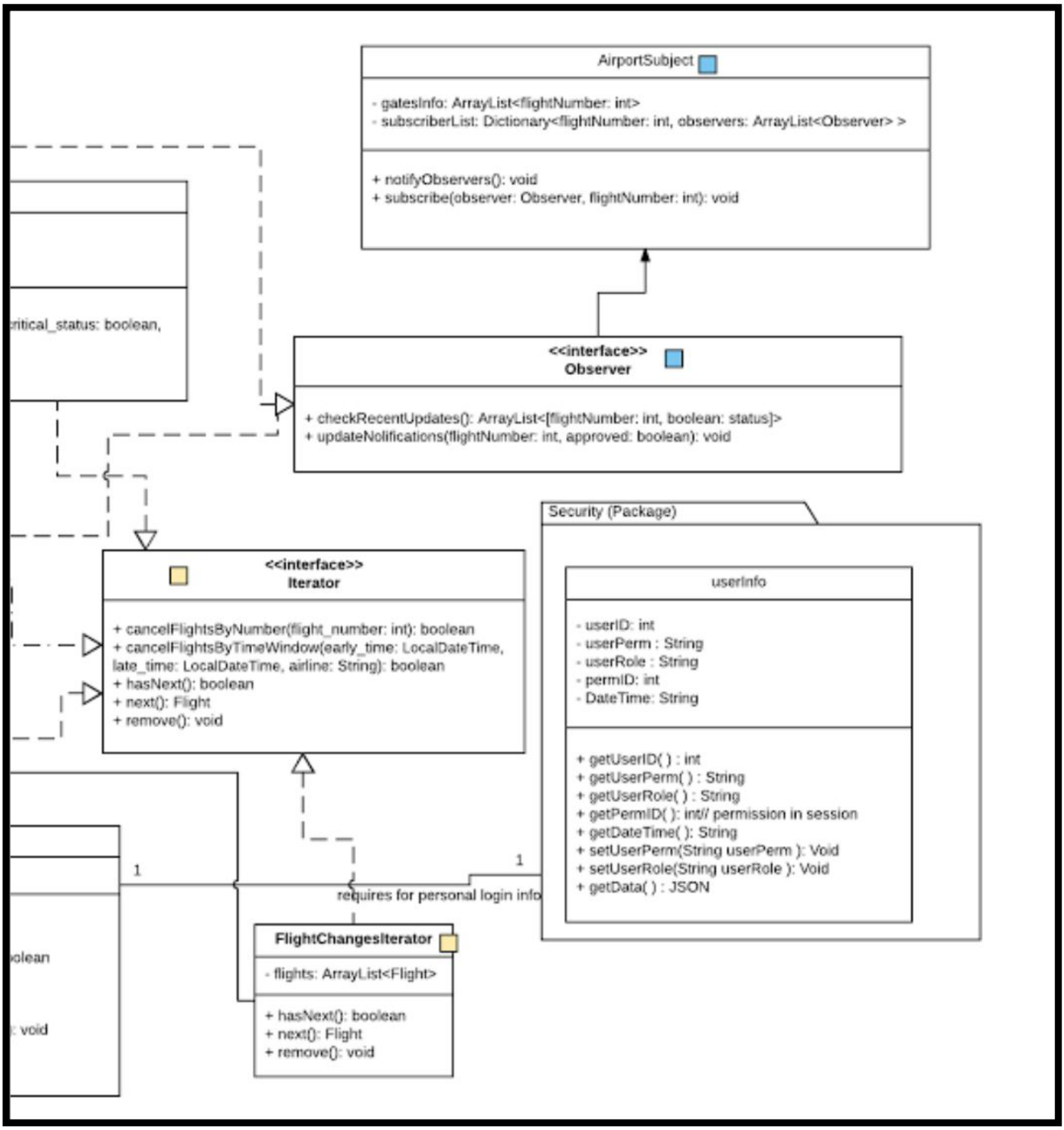
Final Class Diagram



Final Diagram: Zoomed In
Image 1 of 2



Final Diagram: Zoomed In
Image 2 of 2



What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

What changed and why:

- After refactoring, we added three main design patterns, being the observer, facade and lastly iterator (explained below).

- We added the observer design pattern because we wanted both multiple airlines (AirlinePersonnel) and the airport (AirportAdmin) to be notified of changes associated with a flight change at a gate (e.g. one flight moved to a new gate, if another flight is rescheduled).

- We added the facade pattern such that we can make calls to get certain aggregations of flights (Flight class) from the database (Model) without having to expose both the logic and retrieval from the database.

- Lastly, we added the iterator pattern to be able to loop over flights, primarily for when we cancel a group of flights. This way, we do not expose the data structure containing the flights to the caller. Additionally, if the data structure holding flights changes, the external use of iterator does not have change.

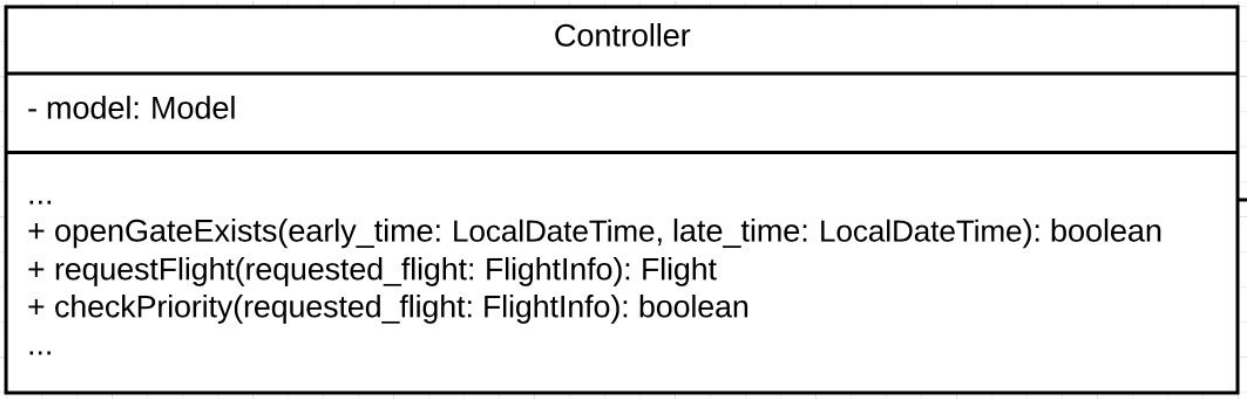
- We also changed the relationship of FlightInfo and Flight to composition over inheritance because Flight has FlightInfo. We wanted to wrap the FlightInfo class inside of Flight so when a request to make a flight is made, the original FlightInfo is preserved (containing information such as Airline and departure/arrival locations.

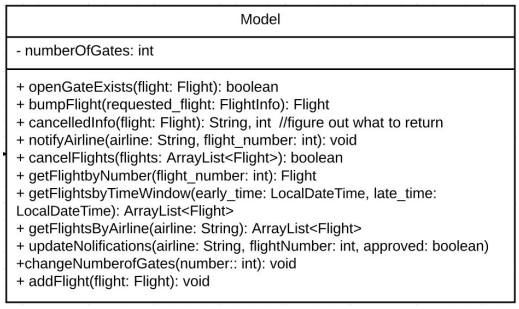
Question 4

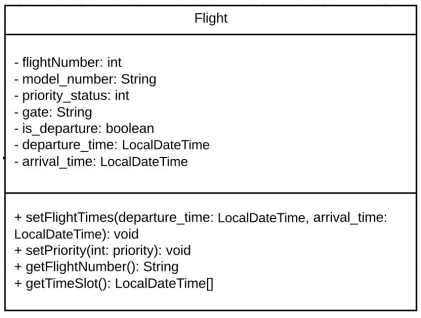
Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

We used observer, iterator, and facade in our final prototype.

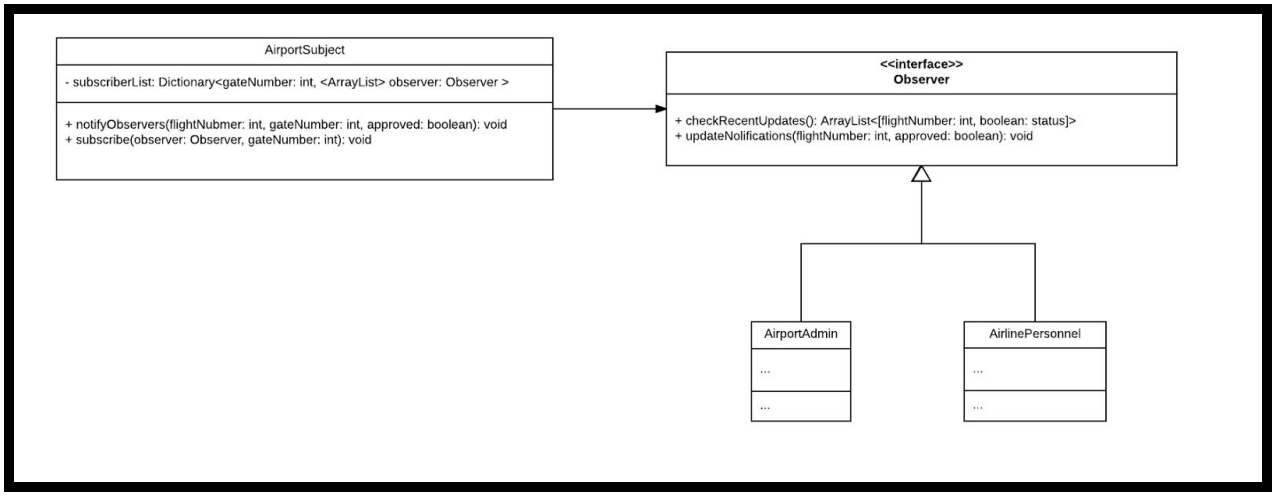
The facade design pattern was used to who different levels of access to our system. The airport admin would have the most control and is able to see all functionality of the program from canceling any flight to adding flights based on the request list. The Controller class made use of Model and Flight to retrieve and manipulate data from the database without exposing the logic to whoever is making the call (see class diagram for associations).



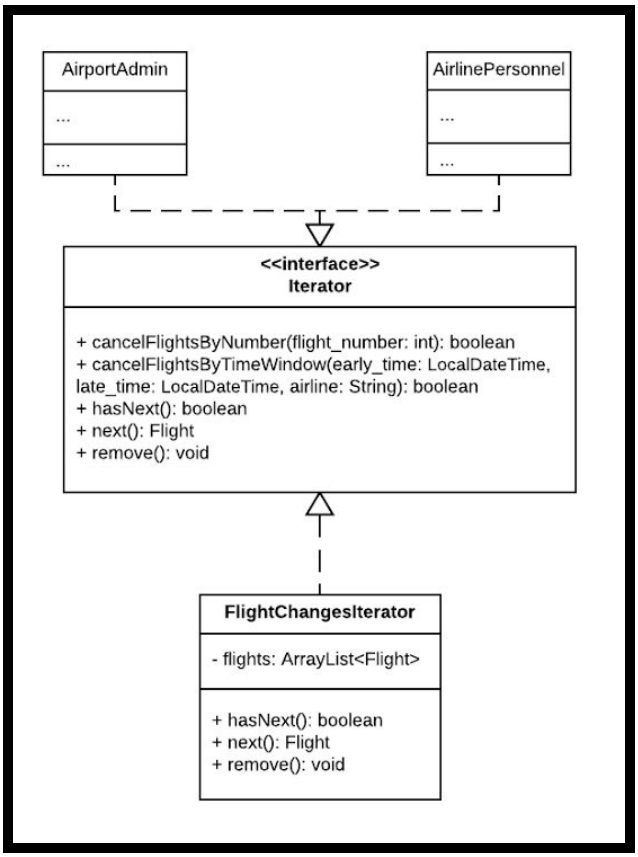




Observer was used in our notification system to show each affected party of any changes in flights that affected them, so if a flight was canceled by the airport admin then the airline personnel and the passenger would get notifications on the cancellation (See class diagram for the variables and methods of AirportAdmin and AirlinePersonne in the figure below).



Finally, the iterator design pattern was used to implement changes in our database, so when a flight was canceled, the database would iterate through and remove each entry that was affected. The same would be true with adding a flight as the database would be scanned of open slots and iterate through the available flight requests sent by the airline personnel.



Question 5

What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

What we learned:

1) Process of designing and implementing a design pattern. By using and planning an appropriate design pattern beforehand, implementing the physical implementation is not as difficult. We also learned the benefits and drawbacks of different design patterns, and therefore how to choose a design pattern and adapt it to solve a particular problem. While many of us used design patterns without our formal knowledge of them in previous projects, knowing the standard nomenclature and vernacular for these patterns proved to be a powerful tool during meetings and team communication.

2) Refactoring and making the code more reusable and therefore having to use less of it. We found that by using design patterns, our code became more reusable (e.g. easy to add new users to our observers design pattern). Additionally, our code became more compact because of this reusability. By having to refactor our initial class diagram with design patterns, we were able to recognize specific areas in the diagram where a design pattern would make sense, and modify it easily without having to go in and scavenge a whole bunch of code

3) Unified Modeling Language and the process of diagramming out the project beforehand. UML allowed our entire team to discuss the specifications of the project, specifically all the use cases and to what direction/extent they would be implemented. In contrast with a software projects class like Software Development Methods and Tools, this allowed all members to be on the same page with the requirements of the projects, and saved time during the coding process, as it drastically reduced the number of miscommunications producing deliverables and during development.