1. **Name:** Javier Ramirez

2. **Description:** A flight service that can be used by a customer and admin. The customer will be able to search destinations, pick a date and time, search a flight, and book a flight. Admin will be able to search number of vacant seats on a flight and search customer information of their picked flight. I set this up for one client only.

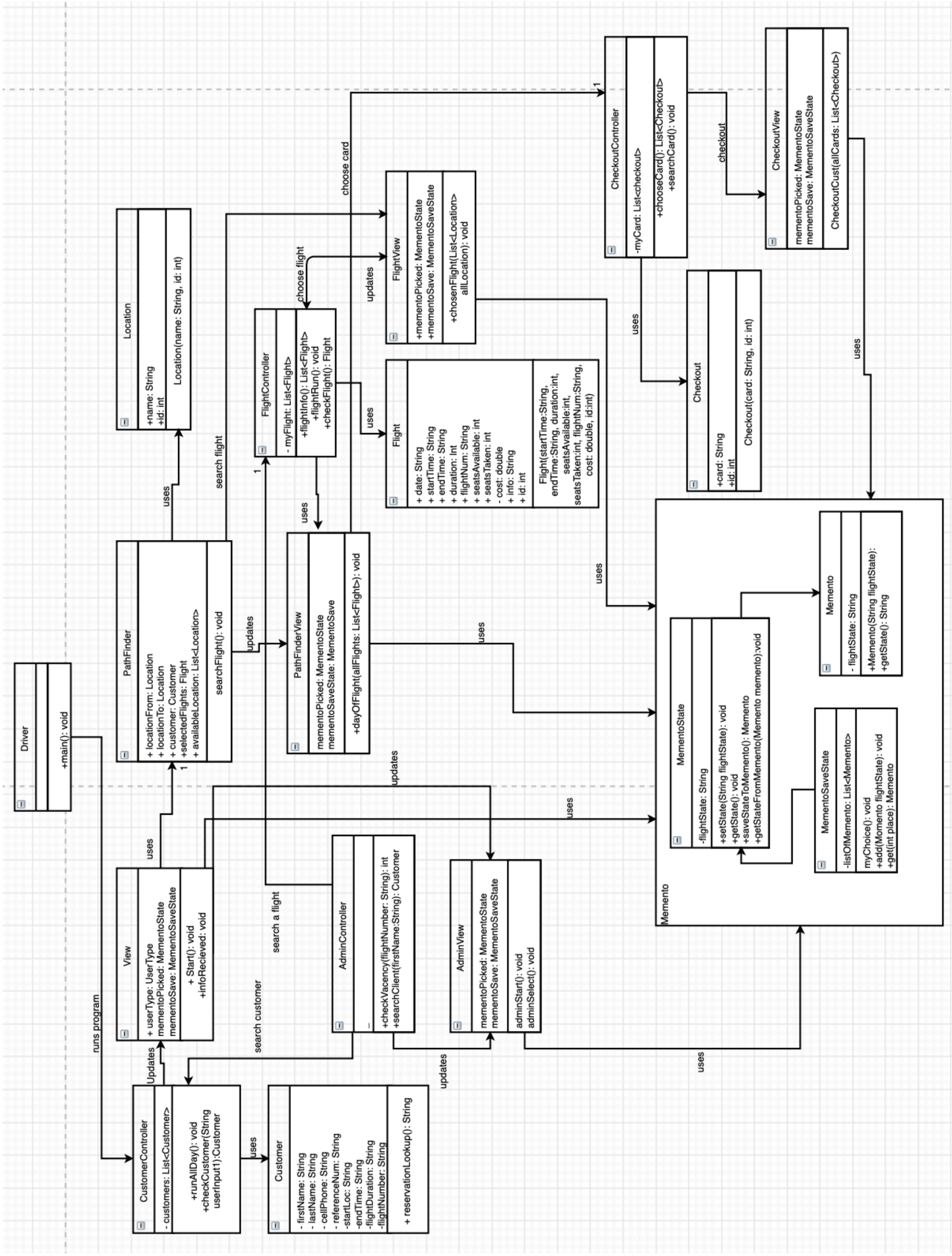
3. **List of features implemented:**

ID	**Requirement**	**Responsibility**	**Actor**	**Priority**
UR-001	As the user, I can pick between the admin or the customer.	Start	User	Critical
UR-002	As the admin, I can check vacancy of seats on a specific flight.	Admin functionality	Admin	Critical
UR-003	As the admin, I want to search a customer by their name.	Admin functionality	Admin	Critical
UR-004	As the customer, I want to input my information and receive a reference number.	Customer	Customer	Critical
UR-005	As the customer, I can select a destination which has multiple flights.	Customer	Customer	Critical
UR-006	As the customer, I can select a flight with multiple flight details for the given date.	Customer	Customer	Critical
UR-007	As a customer, after selecting a flight, I can check out.	Customer	Customer	Critical
UR-008	As a customer, I must agree to the payment amount.	Customer	Customer	Critical

4. **List of features not implemented:**

All features were implemented.

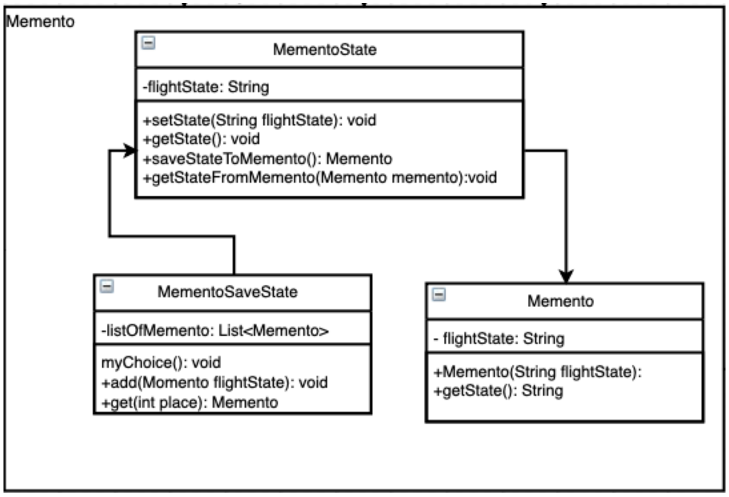
5. Final class diagram:



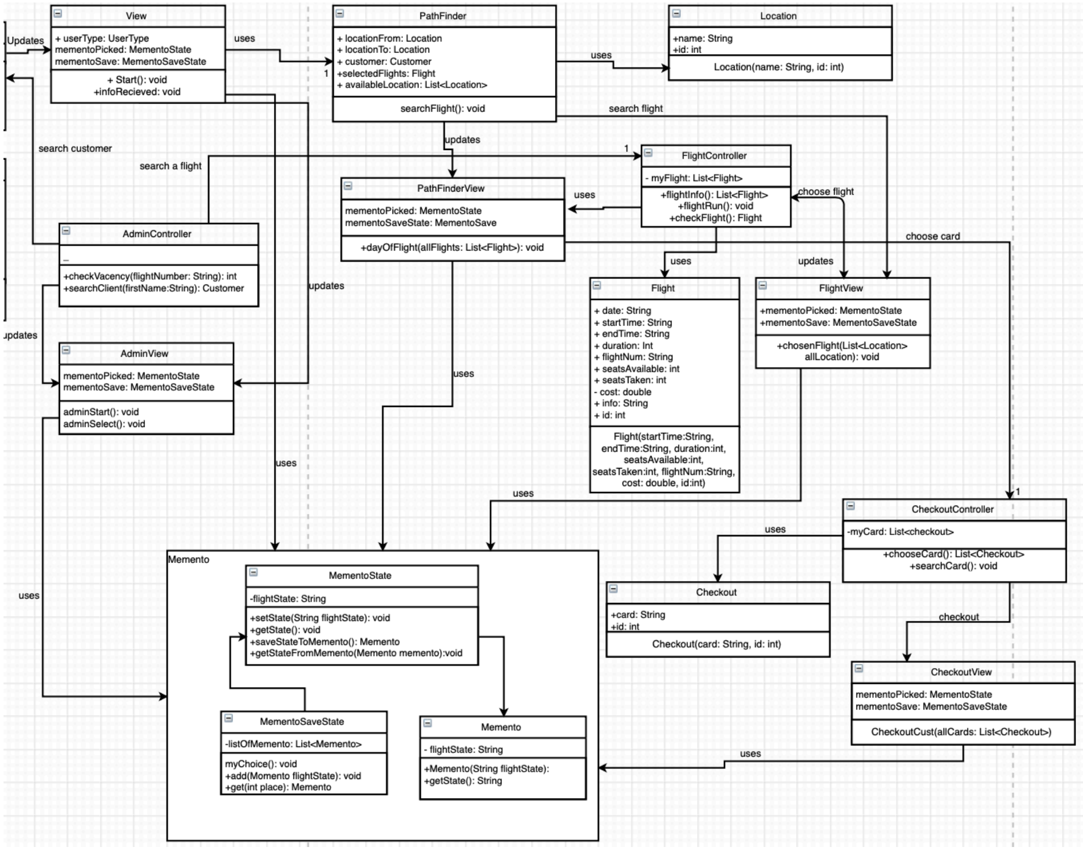
Changes in the final class diagram:
My final class diagram did change. I designed my code and diagram in MVC (model, view, and controller) framework. Which I did not implement in my first diagram. Because of this, I had to make more classes for the MVC framework to be applied. I also implemented the Memento design pattern to my code and diagram, where Memento would be able to be used.

6. Design pattern implemented:

Memento is the design pattern I implemented on to my class diagram.


Memento
MementoState
-flightState: String
+setState(String flightState): void
+getState(): void
+saveStateToMemento(): Memento
+getStateFromMemento(Memento memento):void
MementoSaveState
-listOfMemento: List<Memento>
myChoice(): void
+add(Momento flightState): void
+get(int place): Memento
Memento
- flightState: String
+Memento(String flightState):
+getState(): String

These are the classes from my class diagram that implement the design pattern.


View
+ userType: UserType
mementoPicked: MementoState
mementoSave: MementoSaveState
+ Start(): void
+infoRecieved: void
Updates
uses
1
PathFinder
+ locationFrom: Location
+ locationTo: Location
+ customer: Customer
+selectedFlights: Flight
+ availableLocation: List<Location>
searchFlight(): void
uses
Location
+name: String
+id: int
Location(name: String, id: int)
search flight
updates
search customer
search a flight
1
FlightController
- myFlight: List<Flight>
+flightInfo(): List<Flight>
+flightRun(): void
+checkFlight(): Flight
PathFinderView
mementoPicked: MementoState
mementoSaveState: MementoSave
+dayOfFlight(allFlights: List<Flight>): void
uses
choose flight
choose card
AdminController
_
+checkVacency(flightNumber: String): int
+searchClient(firstName:String): Customer
updates
uses
updates
Flight
+ date: String
+ startTime: String
+ endTime: String
+ duration: Int
+ flightNum: String
+ seatsAvailable: int
+ seatsTaken: int
- cost: double
+ info: String
+ id: int
Flight(startTime:String, endTime:String, duration:int, seatsAvailable:int, seatsTaken:int, flightNum:String, cost: double, id:int)
FlightView
+mementoPicked: MementoState
+mementoSave: MementoSaveState
+chosenFlight(List<Location> allLocation): void
updates
AdminView
mementoPicked: MementoState
mementoSave: MementoSaveState
adminStart(): void
adminSelect(): void
uses
uses
uses
1
CheckoutController
-myCard: List<checkout>
+chooseCard(): List<Checkout>
+searchCard(): void
uses
Memento
MementoState
-flightState: String
+setState(String flightState): void
+getState(): void
+saveStateToMemento(): Memento
+getStateFromMemento(Memento memento):void
MementoSaveState
-listOfMemento: List<Memento>
myChoice(): void
+add(Momento flightState): void
+get(int place): Memento
Memento
- flightState: String
+Memento(String flightState):
+getState(): String
uses
Checkout
+card: String
+id: int
Checkout(card: String, id: int)
checkout
CheckoutView
mementoPicked: MementoState
mementoSave: MementoSaveState
CheckoutCust(allCards: List<Checkout>)
uses

Reason for selecting Memento design pattern

The design pattern I implemented is Memento. I selected this design pattern to let the user go back to a previous state, so they have the option to choose another selection. As the user is moving forward in the process, they may want to choose to save what they did and go back at later time. I selected this design pattern, because it seems more appropriate of a design pattern in a flight service application. Such as a customer wanting to choose a different flight, so they want to go back to a previous state and have that option.

7. Learned about process of analysis and design

Now that I have stepped through the process to create, design, and implement an application, I learned it’s very beneficial to create a class diagram to plan out a project before beginning to code. Where you must have a solid plan, so you know how classes interact with each other. Designing the project, it helped having an MVC framework, because it structures how classes should be connected and progressing as the application runs. After analyzing which design pattern is most appropriate, it was less difficult implementing Memento, because having the class diagram and knowing where the user will be making decision makes it seamless to contrivance. Doing all the steps takes a lot of focus and brainstorming but saves time from implementation errors than when starting from code and having to keep changing the code until it completes the listed user requirement features. Overall, having done this project has helped me improve my development skills by listing the user requirements, making UI mockups, planning out the classes and methods, selecting the appropriate framework, and implementing the most suitable design pattern.