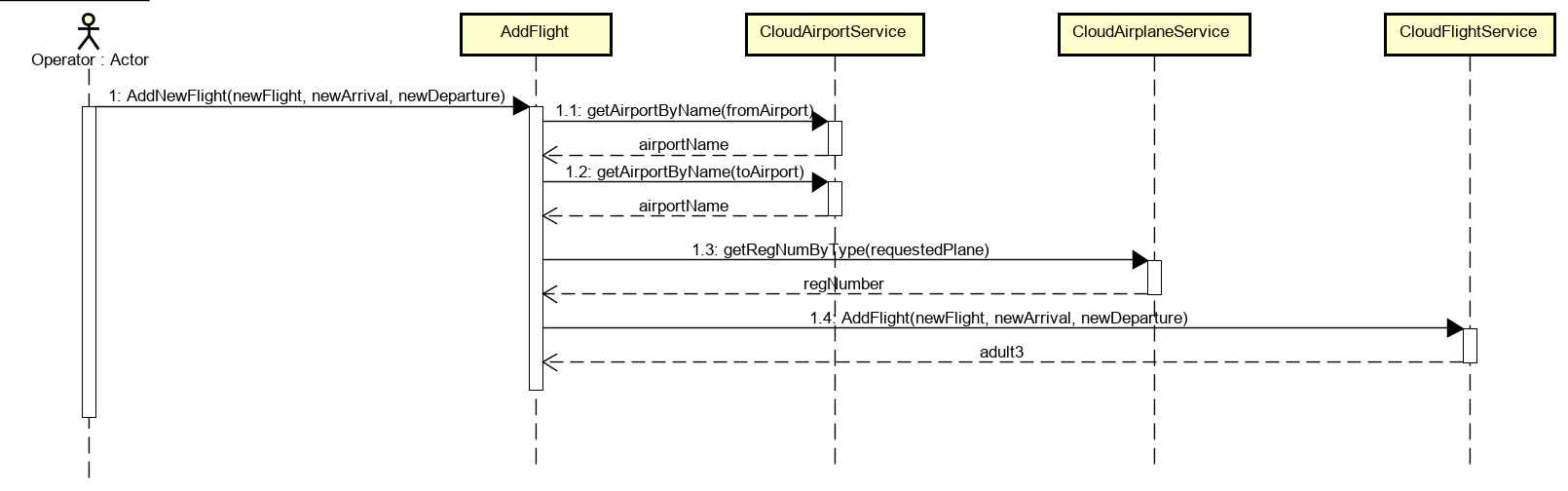
# Sequence diagram (Nicolas Popal)

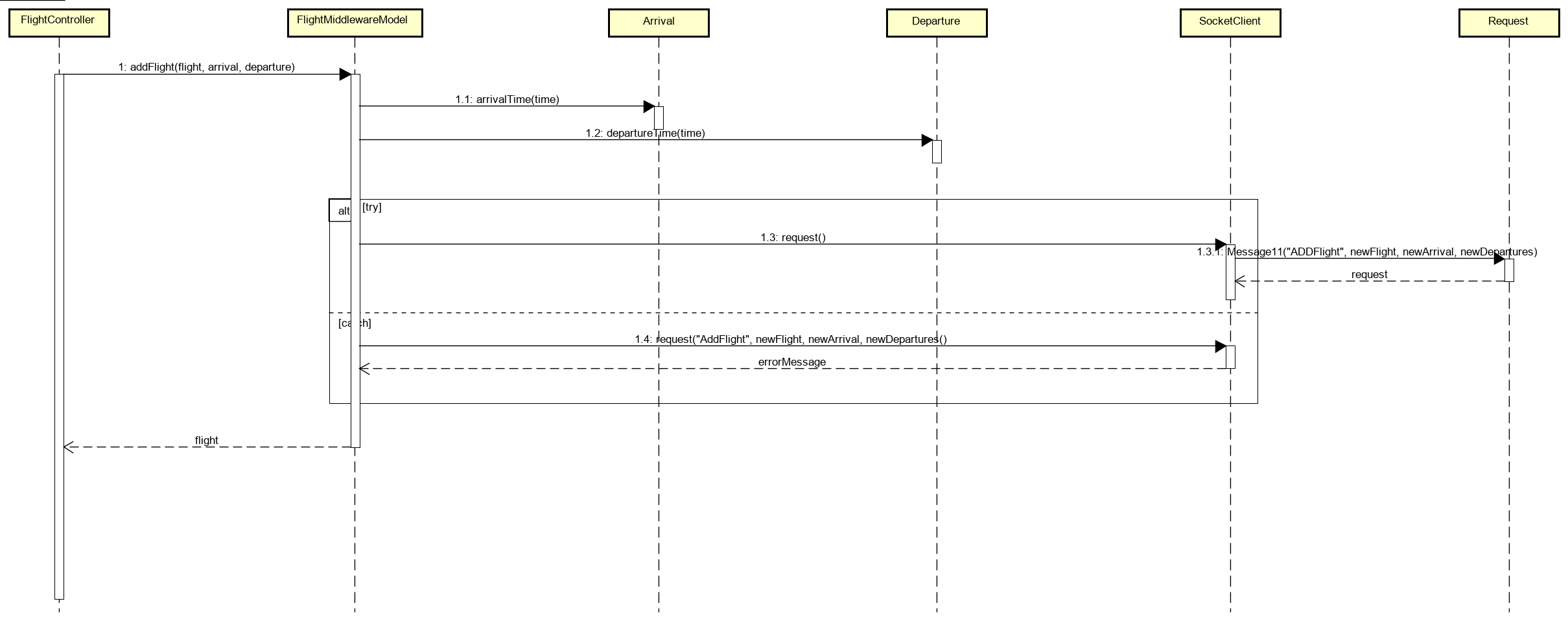
Sequence diagrams are based on the methods in the actual code and are drawn, so that it can be seen what methods and classes are used sequentially in the system. For example, the system sequence diagrams were used for adding flights and view available flights from the airline reservation system.

## Adding flights (client)



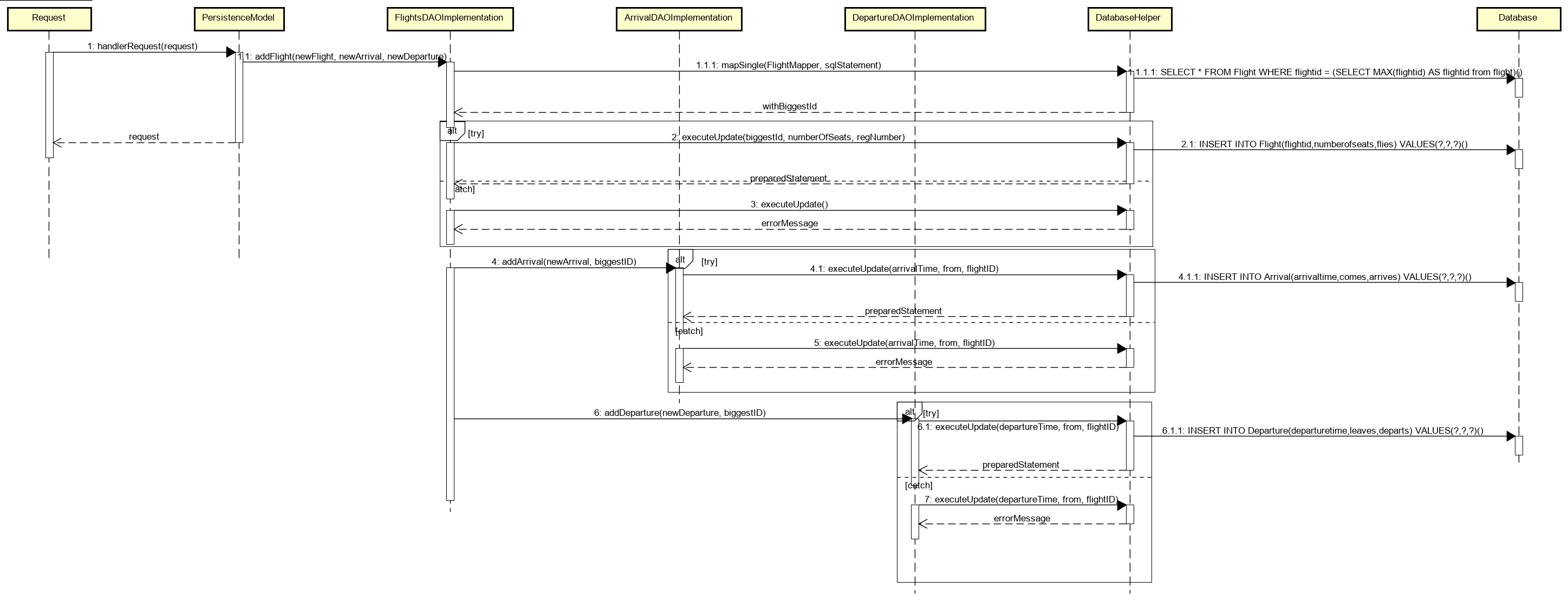
Flight is added, when the confirmation button by operator is clicked. System needs arguments flight, arrival and departure to add the actual flight. Argument arrival and departure is read by the system in CloudAirportServices, which contains information about all the saved airports in the system. Argument flight will be looked up from the system based on the argument regNumber, which is registration number of the airplane already saved in the system. Finally, flight is sent to the middleware through json.

## Adding flights (middleware)



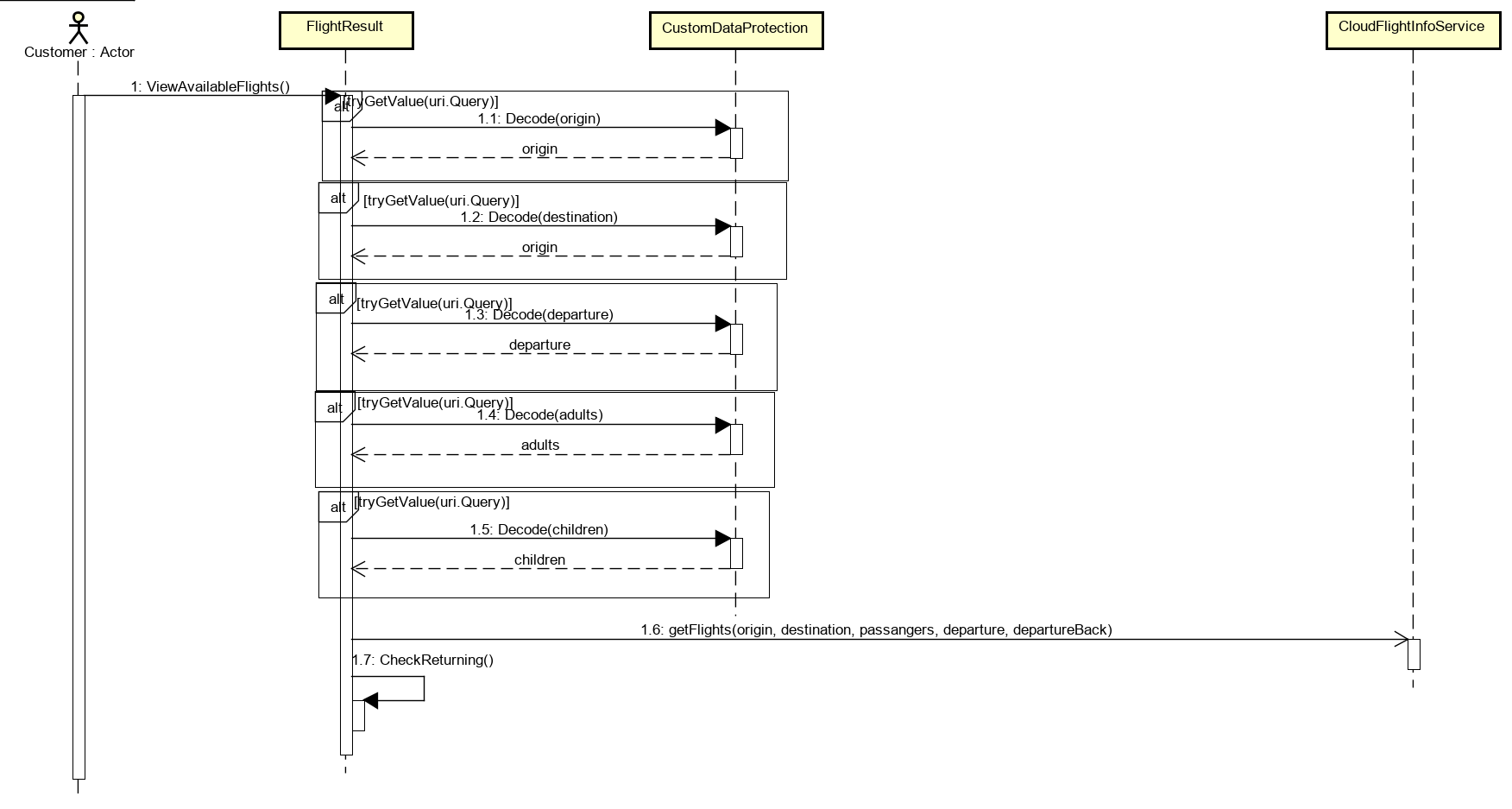
In middleware new flight is received in *FlightController*. *FlightControllers* handles this request in *FlightMiddlewareModel* in *addFlight* method. In this method arrivalTime and departureTime is set and system continues to exception. In this exception system continues to SocketClient, which is sending request to class Request with parameter *ADDFlight* and based on this parameter system distinguish, that new flight is added. Next 3 arguments newFlight, newArrival and newDepartures are data received by middleware from client with all the flight information.

## Adding flights (persistence)



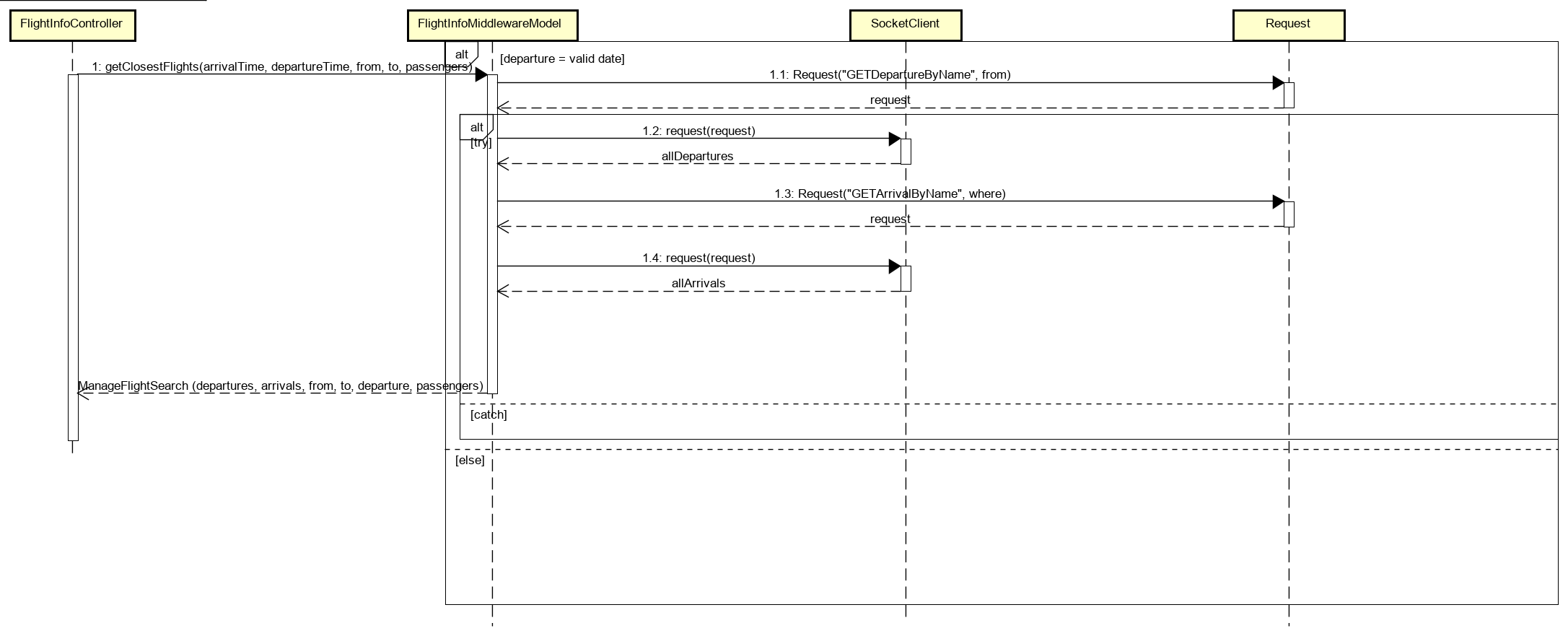
In middleware request to the Request class was sent and as this class is shared with persistence, persistence can receive it. Request is then calling method handlerRequest in PersistanceModel and based on the first argument sent from middleware “ADDFlight” method addFlight is called in class FlightsDAOImplementation. In this class persistence looks up the biggest ID saved in flights, so the new flight can be saved with the next biggest ID. Finally, flight is added to the database in class DatabaseHelper through method executeUpdate. Then class FlightsDAOImplementation is calling methods addArrival in class ArrivalDAOImplementation and method addDeparture in class DepartureDAOImplementation and saving to database departure and arrival of this flight.

## View available flights (client)



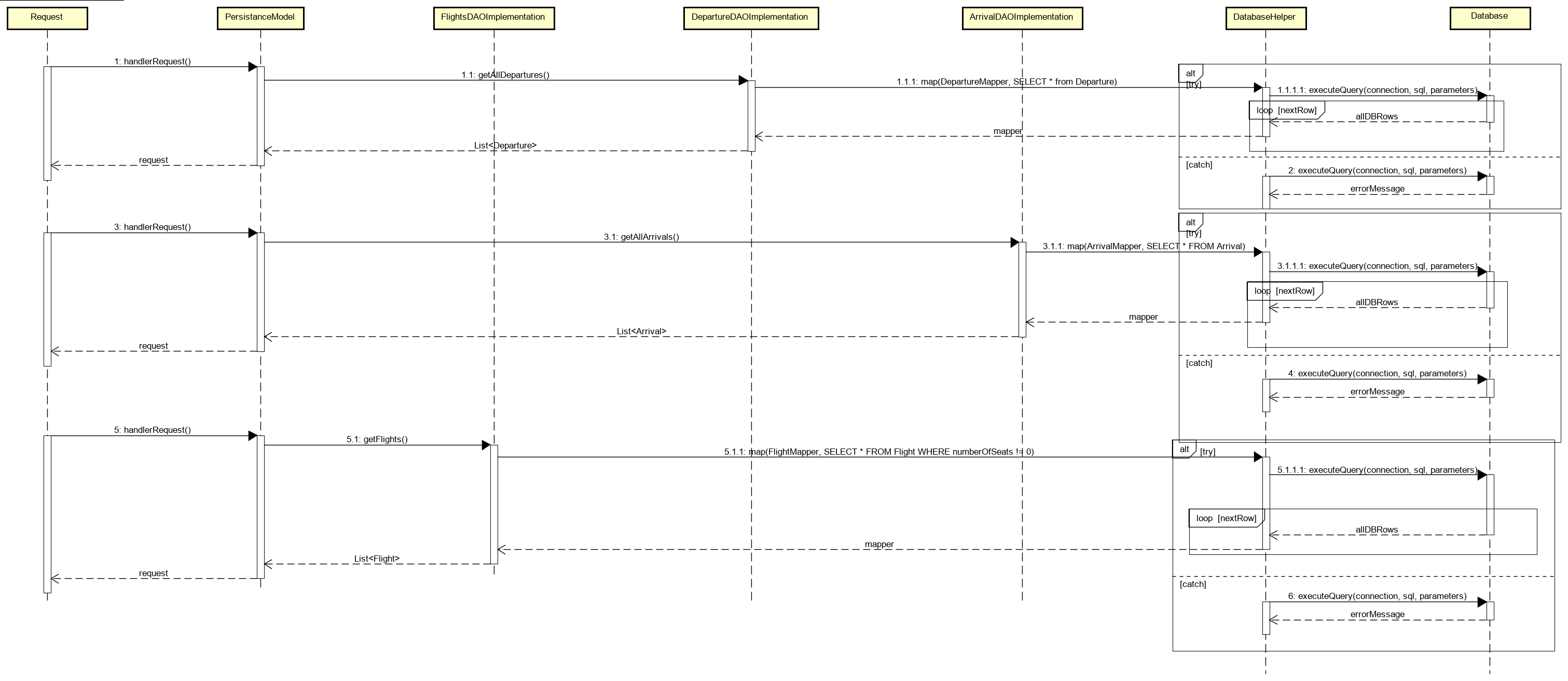
View available flights is displayed, when customer search for a flight. Firstly, FlightResults try all the arguments used in previous page during search and check them in CustomDataProtection. If all arguments are okay, client looks for all flights available in class CloudFlightInfoService, which are getting information in middleware. This will be explained bellow. Also, class FlightResult checks for return date in method CheckReturning.

## View available flights (middleware)



Client sends a message to middleware, where class FlightController receives it. Based on this request, FlightController uses method getClosestFlights from FlightInfoMiddlewareModel. In this class date of the departure is checked and request method to the Request class to get departures can be sent. This sends a request to the persistence where all the departures are saved. Then system continues and saves all the available departures. Same happens for arrivals.

## View available flights (persistence)



Request methods from middleware are received in the persistence. Then all these requests are handled in PersistanceModel and each request is handled by its own DAO class, flight request is handle in FlightsDAOImplementation, departure request is handle in DepartureDAOImplementation and arrival request is handle in ArrivalDAOImplementation. In these classes’ method map from class DatabaseHelper is called and sql statements from arguments are executed. Lists are then returned back and sent as a request back to the middleware.