*Flight Management System* Design Document

# **Introduction**

*This module is the flight orchestration engine. It is a listener to the events emitted by the spacecraft through the communication system and responds to events with the appropriate action. It maintains available flights, current flights and their statuses.*

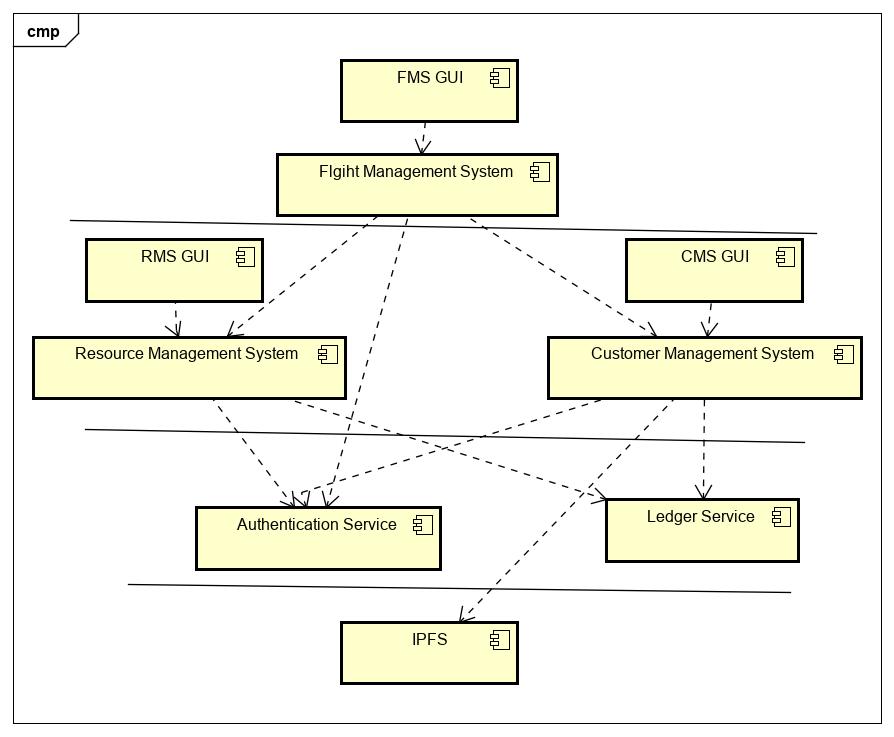
Overview

The goal of ISTS is to fly to space and everything else that comes along with it. The job of this module is to monitor flights, listen to notifications, serve as a medium to inter-spaceship and ground communications, coordinate with other modules to store reports/discoveries and orchestrate a rescue mission during emergencies.

This is an operational module and is an integral part of ISTS. FMS fits into the overall architecture as shown below.

*Overview of the problem to be solved. What is the problem and why is it being solved? How will the resulting solution provide business value?*

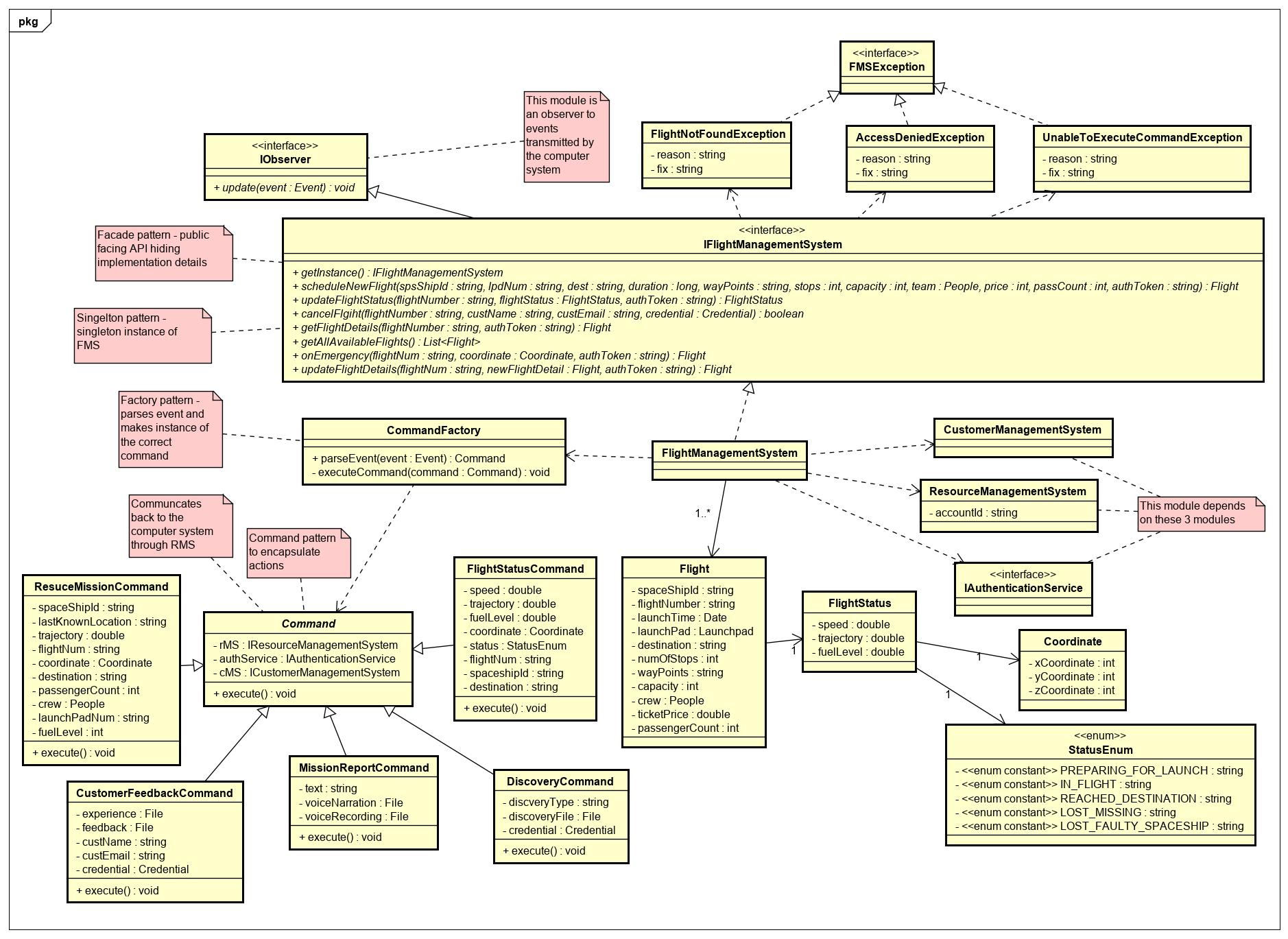
*Consider adding a diagram that explains how this component fits into the overall System with some descriptive text explaining the diagram.*



# 

# Class Diagram

*The following class diagram defines the classes defined in FMS module.*



# 

# Class Dictionary

*This section specifies the class dictionary defined under ‘com.cscie97.ists.fms’.*

## *IFlightManagementSystem*

*This is the top-level interface for the FMS module. Leveraging Proxy Pattern as an independently deployable unit and the Façade Pattern by hiding complexity of implementation details. This interface has methods to schedule and provision new flights. The methods can be called using a valid auth token or through credentials of a customer that FMS will obtain a token on behalf of.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| getInstance | (): IFlightManagementSystem | Singleton instance of FMS |
| scheduleNewFlight | (spsShipId: string, flightNum: string, lpdNum: string, dest: string, duration: long, wayPoints: string, stops: int, capacity: int, team: Team, price: int, passCount: int): Flight | Schedules a new flight when all the details are provided. The spaceship and launchpad must be available resources in RMS. This method queries RMS to find out that the resources exist. A unique UUID flight number will be generated by the system. Then it gets dumped to CMS for accessibility to customers and RMS to keep track of the available flights |
| updateFlightStatus | (flightNumber: string, flightStatus: FlightStatus): FlightStatus | Administrators can manually update the status of the current flight like fuel level and speed. The spaceship can also automatically send signals through the communication system using the observer pattern. |
| cancelFlgiht | (flightNumber: string, custName: string, custEmail: string, credential: Credential): boolean | Customers can authenticate using their credentials and pick a flight they want to cancel. If canceling is successful, customers will get confirmation email of the cancellation. Cancelling missed flights incurs penalty to customers |
| getFlightDetails | (flightNumber: string, authToken: string): Flight | Customers or admins can inquire about the details of any upcoming flight |
| getAllAvailableFlights | (): List<Flight> | List of upcoming flights is publicly available and doesn’t require authentication |
| onEmergency | (flightNum: string, coordinate: Coordinate, authToken: string): Flight | This enables admins or customers to manually report emergency. Emergency can also be reported using the communication system |
| updateFlightDetails | (flightNum: string, newFlightDetail: Flight, authToken: string): Flight | Administrators can amend the details of a flight like destination or number of stops before take off |

## *FlightManagementSystem*

*Implementation of IFlightManagementSystem discussed above. It depends on CMS and RMS to dump newly scheduled flights to make it accessible to CMS GUI or to keep track of resources for admins looking at RMS GUI. It also delegates checking for validity of tokens to Authentication Service. It keeps a list of available flights and forwards events received using the Observer Pattern to CommandFactory which parses the payload. CommandFactory to be discussed below uses the Factory Pattern to make instances of the various commands and the Command Pattern to execute queueable actions.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| getInstance | (): IFlightManagementSystem | Singleton instance of FMS |
| scheduleNewFlight | (spsShipId: string, flightNum: string, lpdNum: string, dest: string, duration: long, wayPoints: string, stops: int, capacity: int, team: Team, price: int, passCount: int): Flight | Schedules a new flight when all the details are provided. The spaceship and launchpad must be available resources in RMS. This method queries RMS to find out that the resources exist. A unique UUID flight number will be generated by the system. Then it gets dumped to CMS for accessibility to customers and RMS to keep track of the available flights |
| updateFlightStatus | (flightNumber: string, flightStatus: FlightStatus): FlightStatus | Administrators can manually update the status of the current flight like fuel level and speed. The spaceship can also automatically send signals through the communication system using the observer pattern. |
| cancelFlgiht | (flightNumber: string, custName: string, custEmail: string, credential: Credential): boolean | Customers can authenticate using their credentials and pick a flight they want to cancel. If canceling is successful, customers will get confirmation email of the cancellation. Cancelling missed flights incurs penalty to customers |
| getFlightDetails | (flightNumber: string, authToken: string): Flight | Customers or admins can inquire about the details of any upcoming flight |
| getAllAvailableFlights | (): List<Flight> | List of upcoming flights is publicly available and doesn’t require authentication |
| onEmergency | (flightNum: string, coordinate: Coordinate, authToken: string): Flight | This enables admins or customers to manually report emergency. Emergency can also be reported using the communication system |
| updateFlightDetails | (flightNum: string, newFlightDetail: Flight, authToken: string): Flight | Administrators can amend the details of a flight like destination or number of stops before taking off |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| flights | List<Flight> | List of upcoming flights |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| commandFactory | CommandFactory | Factory class to parse events and delegate to the appropriate commands |
| cms | ICustomerManagementSystem | Used to sync with CMS about the details of a newly scheduled flight |
| rms | IResourceManagementSystem | Used to store about the latest details of a current flight and inventory of flights |
| authService | IAuthenticationService | Used to check for validity of tokens and required permissions |

## *Flight*

*This object is instantiated when a new flight is scheduled.*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| spaceshipId | string | Id of the spaceship |
| flightNumber | string | Unique UUID assigned by the system when a new flight is scheduled |
| launchTime | Date | Time to launch spaceship |
| launchPad | Launchpad | Launchpad where spaceship takes off |
| destination | string | Destination of the flight |
| numOfStops | int | The number of stops before destination |
| wayPoints | string | Transit addresses |
| capacity | int | This capacity is the spaceship’s capacity less the number of persons in the crew |
| crew | People | Composite Pattern - A hierarchy of people consisting of teams and persons can be part of the crew. These can be operators, pilots and flight attendants |
| ticketPrice | double | Variable cost of ticket. Subject to change |
| passengerCount | int | Current passenger count and can be updated upon new customers booking a flight or cancelling a flight |

## *CommandFactory*

*A Factory class that parses events sent from an Observable and takes actions using the Command pattern.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| parseEvent | (event: Event): Command | Parses the payload in the event and makes instance of the appropriate command. It calls the private executeCommand method internally to execute the command. It also sets the initial state that the command needs from the events payload to be able to take actions. It can store commands in a queue and take bulk actions |
| executeCommand | (command: Command): void | A private method that just calls execute() in a generic command. It can be called in bulk of commands in queue or individually |

## *Command*

*This is an abstract class that concrete commands provide extensions to. This class contains singleton instances of RMS, CMS and AuthService that concrete commands may need to perform actions on.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | All commands provide a way to execute a command regardless of the action |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| rms | IResourceManagementSystem | Commands can execute actions on RMS |
| authService | IAuthenticationService | Commands can obtain tokens from AuthService to perform actions on protected resources from RMS and CMS |
| cms | ICustomerManagementSystem | Commands can execute actions on CMS |

## *RescueMissionCommand*

*When a distress signal is sent from a spaceship, a rescue mission is setup to address the issue and save lives.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | Schedule a new flight of a rescue team with enough capacity to accommodate passengers and crew in the emergency, with enough fuel for the trip, tailored to the needs of the crew in emergency and sent to the last reported location of the spaceship in emergency. The speed of the rescue spaceship should also be as fast as possible to minimize the risk of lost lives. The spaceship should also have rescue appliances for passengers in need, firemen who can perform CPR and should be reliable enough to withstand any dire conditions where the emergency is. RMS will be useful to lookup resources and to define new resources for rescue. CMS will be useful to identify the passengers involved in emergency. AuthService will be useful to determine the authenticity of the reporter. Nearby spaceships may also be assigned for the rescue |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| spaceShipId | string | Identifies the spaceship in emergency |
| lastKnownLocation | string | Determines where rescue should be sent to |
| trajectory | double | Makes it easy to navigate to the spaceship in emergency |
| flightNum | string | Identifies the flight and the passengers who boarded |
| coordinate | Coordinate | Helps navigate to the spaceship in emergency |
| destination | string | The spaceship may have moved closer to the destination after reporting emergency and it is important to know where the spaceship is headed |
| passengerCount | int | The number of passengers must be determined to accommodate them in a rescue spaceship |
| crew | People | The crew on the line can be identified and if their capabilities are known, the rescue team can be tailored to suit their needs |
| launchPadNum | string | The staff in the ground can navigate to the launchpad and try to trace the underlying cause |
| fuelLevel | int | Determines if the spaceship in emergency is just running out of fuel |

## *MissionReportCommand*

Passengers and crew can send mission updates anytime during the flight. The records can be text or voice narration. To ensure the credibility of the reporter, the credentials of reporter will be sent to CMS where the reporter is identified using AuthService then the file is stored in IPFS upon successful verification.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | The report is sent to CMS, CMS checks authenticity of the reporter through AuthService then report is sent as a file object to IPFS upon successful verification |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| text | string | Text to be stored in IPFS through CMS |
| voiceNarration | File | Voice narration to be stored in IPFS through CMS |
| voiceRecoring | File | Voice recording to be stored in IPFS through CMS |

## *CustomerFeedbackCommand*

*Customers can send their feedback during flight or after successful return. The feedback will be tied to the customer in CMS and a history of the customer’s experience can be stored for later retrieval or analytics. The files will also be archived and stored in IPFS. Customers can also provide feedback anonymously which will be stored in its own anonymous category not tied to any users.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | A document of experiences or feedback can be stored in IPFS through CMS anonymously or tied to the permanent record of a customer if credentials are provided |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| experience | File | Customers can provide a document of their experience to be stored as a file in IPFS through CMS |
| feedback | File | Customers can provide a document of their feedback to be stored as a file in IPFS through CMS |
| custName | string | Customers can choose to provide their name or provide feedback anonymously |
| custEmail | string | Customers can choose to provide their email or provide feedback anonymously |
| credential | Credential | Customers can choose to provide their credential or provide feedback anonymously |

## *DiscoveryCommand*

*When a discovery is reported, the authenticity of the reporter must be verified to hold any credibility. The files are then stored in IPFS through CMS but they are not permanent records associated with a customer.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | Reporter is authenticated and discovery file is sent to IPFS through CMS |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| discoveryType | string | Category of discovery |
| discoveryFile | File | File objects of discovery to be stored in IPFS |
| credential | Credential | To avoid security risks, the reporter must be verified |

## *FlightStatusCommand*

During flight, notification will be sent to the ground through notification system in a configurable interval. This is for a health check and for staying in sync with operators on the ground. Using pushing mechanism of the observer pattern instead of pushing, operators can stay abreast of flight information.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | (): void | CMS and RMS will be sent up to date information about flight statuses using this command. Concerned relatives of customers can explore the CMS portal to check the safety of their loved ones. Admins and staff members can track all the flights currently on air |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| speed | double | Current speed |
| trajectory | double | Current trajectory |
| fuelLevel | double | Current fuel level |
| coordniate | Coordinate | Coordinate of the spaceship in space |
| status | StatusEnum | The status can be preparing flight, reached destination or missing |
| flightNum | string | Identifies the flight |
| spaceshipId | string | Identifies the spaceship flight |
| destination | string | Destination may change due to emergency and those updates can be sent |

## *FlightStatus*

*Speed, trajectory and fuel level*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| speed | double | Current speed |
| trajectory | double | Current trajectory |
| fuelLevel | double | Current fuel level |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| coordinate | Coordinate | X, Y and Z coordinates of the current flight |

## *Coordinate*

*X, Y and Z coordinates*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| xCoordinate | int | X coordinate |
| yCoordinate | int | Y coordinate |
| zCoordinate | int | Z coordinate |

## *StatusEnum*

Status of flight.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| PREPARING\_FORLAUNCH | string | Preparing for launch |
| INFLIGHT | string | In flight |
| REACHEDDESTINATION | string | Reached destination |
| LOSTMISSING | string | Missing |
| LOSTFAULTYSPACESHIP | string | Faulty spaceship |

## *FMSException*

Marker interface all the exceptions thrown in this module implement

## *FlightNotFoundException*

Looking up a flight that is no longer available or never scheduled

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| reason | string | Reason for failure |
| fix | string | Hint to fix the problem |

## *AccessDeniedException*

Trying to access the service API with expired token or without required permissions

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| reason | string | Reason for failure |
| fix | string | Hint to fix the problem |

## *UnableToExceuteCommandException*

When commands fail to accomplish their tasks due to network failures during interservice communication

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| reason | string | Reason for failure |
| fix | string | Hint to fix the problem |

# Implementation Details

*Explain details of the implementation.*

*How do the various parts fit together or interact?*

*How does the design address the requirements? Justify your design decisions and how they address the requirements.*

*Some implementation details may be addressed in the class dictionary, but for things that are not, describe them here.*

*Remember to reference the requirements from the body of the design document to show how your design is addressing the requirements.*

# Exception Handling

*Provide details on your exception handling. What types of exceptions are expected and how are they handled by the design? Describe your exception classes and their properties.*

# Testing

*Provide a testing strategy for testing the component.*

* *Functional*
* *Performance*
* *Regression*
* *Exception Handling*

# Risks

*Document any risks identified during the design process.*

*Are there parts of the design that may not work or need to be implemented with special care or additional testing?*