Interplanetary Space Transport System (ISTS) Architecture

Date: 12/18/2019

Author: Gerald Arocena

Reviewer(s): Krithika Sundararajan

Whole document structure overview/explanation here.

# Introduction

This system design document explains how to implement an interplanetary space travel system for a program put forth by the International Space Agency. The program’s objective includes expanding knowledge of the solar system and the universe. The software system that this design document outlines includes the use of spaceships, communication systems, and distributed file systems.

Overview

Due to advances in space travel technology, humans now have the capability to travel through the universe and research it which has many benefits. For instance, space exploration can potentially save Earth from being victim to outerspace threats like asteroid impacts. In addition, exploring space can be seen as a leisure activity.

A high-level component diagram is shown on the next page that illustrates the relationship between the components that make up the Interplanetary Space Transport (IST) System.



Caption: The three main modules – Resource Management Service, Customer Service, and Flight Management Service – are shown above as well as the other system components of Ledger Service, Authentication Service, IPFS (InterPlanetary File System), and GUI. The arrows point from each component to their dependencies.

# Requirements

The IST system must comprise three main modules: Resource Management, Customer Service, and Flight Management. It must also provide a graphical user interface (GUI) for each. The GUI should allow administrators and passengers to login to access the IST system and should be implemented using the underlying services. They must support the use cases (discussed below) in this document. In addition, the IST system must use the Ledger Service implemented in Assignment 1 to process any payment transactions including ticket purchases and ISTS account management. The Authentication Service implemented in Assigment 4 must also be used to check access permissions for the GUI and service APIs including access to spaceships and when boarding passengers. Furthermore, the Interplanetary File System (IPFS) must be used for storage for flight-related activities including for travel documents, in-flight entertainment, and documention of flight experiences.

The modules will also need to be implemented independently of one another such that they exhibit level 5 of the Modularity Maturity Model: Service Oriented Architecture. Each module must define a service interface that the other modules can access and the GUI should use the service interfaces for access by users. Additionally, design patterns should be applied and highlighted where appropriate.

Persistence

The ISTS will use the InterPlanetary File System for sharing documents and long-term storage. Is is assumed that connectivity to the IPFS will not be an issue on Earth or anywhere in space.

# Use Cases

The IST system should support allowing administrators to manage, monitor, and update resources, people/teams, and flights. Administrators are tasked with managing communication and information between the resources including flight and spacecraft status, messages from spacecraft, and ground-based communication and automated control systems.

# Implementation Details

In order to achieve level 5 modularity maturity as required, each module is implemented in its own package as a microservice and any components external to a package must use its service API in order to access it. Additionally, this exclusive use of an API for access hides the underlying details of a module’s implementation. This exemplifies the Façade design pattern. In addition, each module will provide a Singleton instance of its service since multiple instances would be unnecessary and costly. This exemplifies the Singleton design pattern.

Also, it should be noted that the accessors and mutators included in this document are not comprehensive. The ones in this document were included in order to highlight the importance of its access permissions and/or the importance of its function to the overall IST system. In addition, my implementation of the Authentication Service used a class called AuthTokenTuple (which includes an AuthToken) and is included in this document. Please refer to my Authentication Service implementation for further clarification on its use.

ISTS Resource Management Service Design Document

Document structure overview here.

# Introduction

This design document explains how to implement the Resource Management Service module of the IST system. The Resource Management Service provisions and maintains the state of the domain objects of the ISTS that aren’t customer service related. Domain resources include human resources like person and team, and physical resources such as a spaceship and budget.

Overview

The Resource Management Service provides an API for ISTS administrators to interact with the ISTS resources. It supports querying their state as well as updating their state. In addition to provisioning resources, it is able to create/simulate events emitted by spaceships and the communication system.

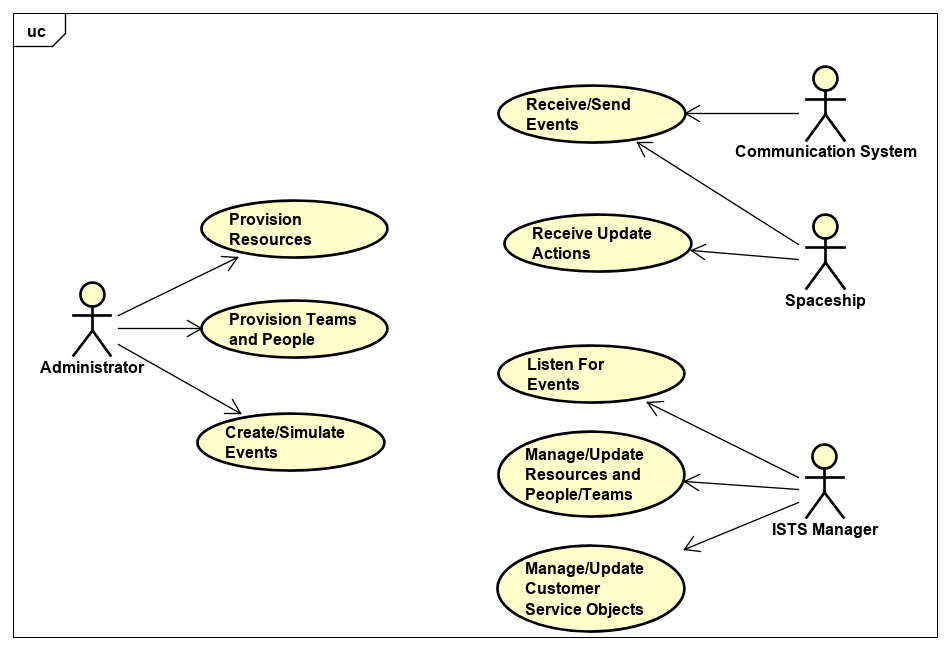
# Requirements

The Resource Management System must be able to view as well as manage teams and people where people can be part of one or more teams. It must also manage the IST system’s Ledger account since it has a budget resource that must be used to purchase resources with. All resource management including of physical assets like spaceships and launchpads need to support all four CRUD operations, i.e., create, read, update, and delete. Access to its domain objects’ information is important because the GUI and other modules rely on them to function. For example, the Flight Management System might depend on the Resource Service to find out a spaceship’s capacity for flight creation. It will also be listening for interesting events in the Resource Management Service in order to manage and update its resources. Other resources include fuel, communication system, and computer system.

The Resource Service must also be able to create spaceship and communication system events. Spaceships must be able to receive update actions in response to events such as “reached destination”. For this module’s design document, events can be thought of as opaque strings. Finally, all API methods must accept an AuthToken parameter to support access control.

# Use Cases

The diagram below illustrates the uses cases of the Resource Management Service.



Caption: The actors in the Resource Management Service. Their use cases are pointed to and state what interaction each actor has with the system.

**Actors:**

The actors of the Resource Management Service are Administrator, Communication System, Spaceship, and ISTS Manager.

**Administrator**

An Administrator is allowed to use any of the Resource Service’s methods. They can provision the ISTS system and have full access to any of its resources.

**ISTS Manager**

The Manager implements the Flight Management Service. It listens for interesting events emitted by the Resource Management Service and responds with an appropriate action including requests to the Resoure Service. For example, an event could be that a spaceship crashed which the Manager could respond to by deploying a spaceship from the Resource Management Service for a rescue flight mission.

**Communication System**

The communication system serves as an intermediate agent in the communication system of the ISTS. It is utilized in creating/simulating events.

**Spaceship**

The spaceships are able to communicate update events about their status or the status of the system to the Resource Management Service.

**Provision Resources**

Administrators can provision the IST system with resources. Provisioning resources involves buying them first so an ISTS budget must be maintained.

**Provision Teams and People**

Administrators can provision and manage human resources such as the people and teams that make up the ISTS organization hierarchy.

**Create/Simulate Events**

An administrator can create/simulate spaceship and communication system events such as a spaceship getting lost in space.

**Receive/Send Events**

Spaceships and the communication system are able to send/receive simulated events that imitate real world events that could happen. These should take the form of formatted and translatable messages.

**Receive Update Actions**

The spaceship and Resource Management Service should be able to be updated by external automatic control systems, e.g., the Flight Management Service in response to events.

**Listen For Events**

The ISTS Manager listens for interesting events that happen in the Resource Management Service. It gets notified by the Resource Management Service of such events.

**Manage/Update Resources and People/Teams**

The Manager executes update actions on Resource Management Service objects in response to events.

**Manage/Update Customer Service Objects**

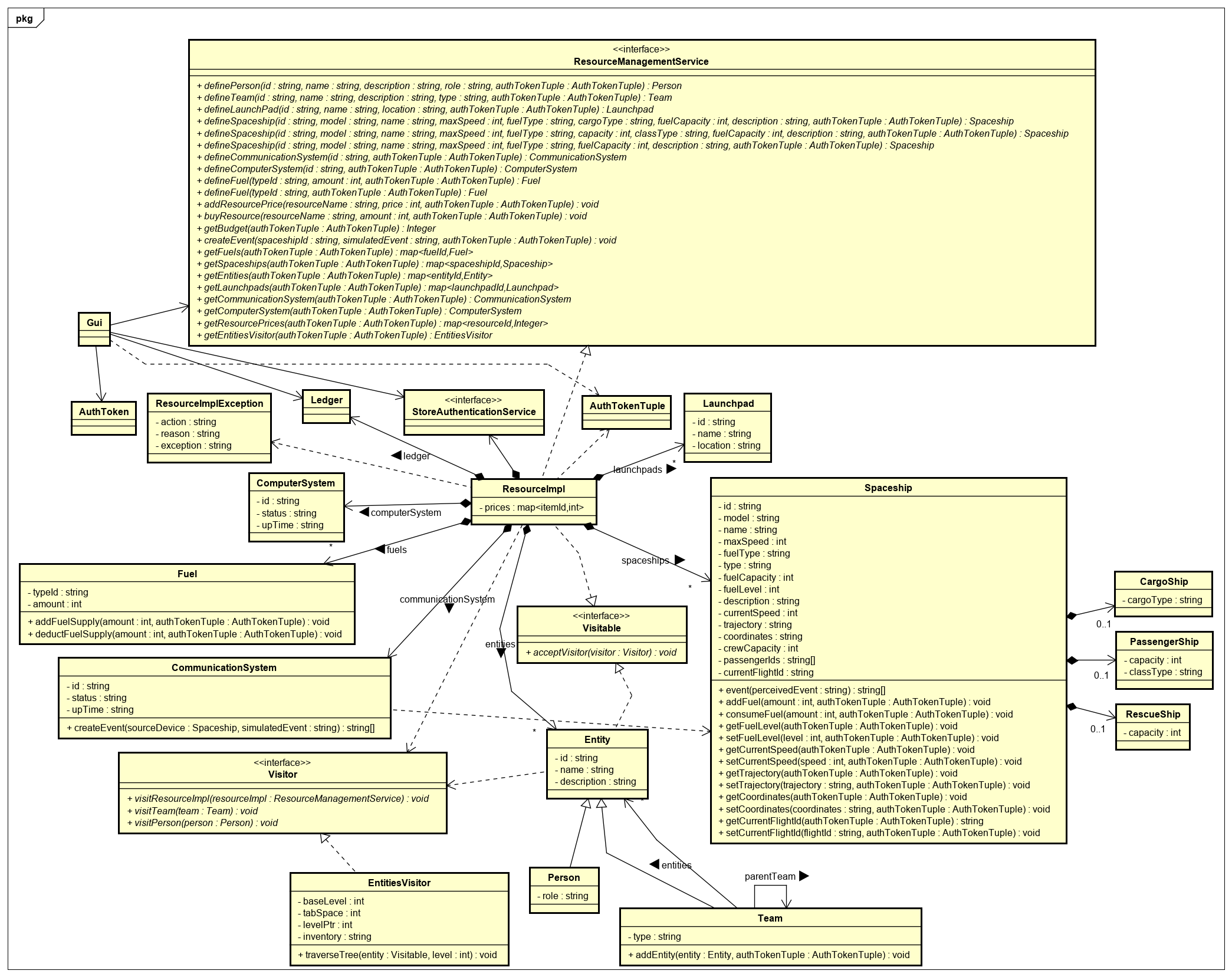
The Manager executes update actions on Customer Service objects in response to events.

# Implementation

Of note is that the Resource Management Service makes use of the Composite and Visitor design patterns with the human resources. This is shown in the following class diagram and class dictionary.

# Class Diagram

The following Resource Management Service class diagram defines the classes defined in this design that comprise the package “com.cscie97.ists.resource”.



Caption: Class diagram for the Resource Management Service.

# Class Dictionary

This section contains the class dictionary for the Resource Management Service that comprise the package “com.cscie97.ists.resource”.

## ResourceManagementService (Interface)

The Resource Management Interface defines the API for the Resource Management Service. It is the main point of access external components and entities use for interacting with its service via its exposed methods.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| definePerson | (id : string, name : string, description : string, role : string, authTokenTuple : AuthTokenTuple) : Person | Creates a Person within the IST organization. |
| defineTeam | (id : string, name : string, description : string, type : string, authTokenTuple : AuthTokenTuple) : Team | Creates a Team within the IST organization. |
| defineLaunchPad | (id : string, name : string, location : string, authTokenTuple : AuthTokenTuple) : Launchpad | Creates a launchpad for spaceships. |
| defineSpaceship | (id : string, model : string, name : string, maxSpeed : int, fuelType : string, cargoType : string, fuelCapacity : int, description : string, authTokenTuple : AuthTokenTuple) : Spaceship | Creates a spaceship used for space flight. This is the method for cargo spaceships. |
| defineSpaceship | (id : string, model : string, name : string, maxSpeed : int, fuelType : string, capacity : int, classType : string, fuelCapacity : int, description : string, authTokenTuple : AuthTokenTuple) : Spaceship | Creates a spaceship used for space flight. This is the method for passenger spaceships. |
| defineSpaceship | (id : string, model : string, name : string, maxSpeed : int, fuelType : string, fuelCapacity : int, description : string, authTokenTuple : AuthTokenTuple) : Spaceship | Creates a spaceship used for space flight. This is the method for rescue spaceships. |
| defineCommunicationSystem | (id : string, authTokenTuple : AuthTokenTuple) : CommunicationSystem | Creates a communication system used by the ISTS. |
| defineComputerSystem | (id : string, authTokenTuple : AuthTokenTuple) : ComputerSystem | Creates a computer system used by the ISTS. |
| addResourcePrice | (resourceName : string, price : int, authTokenTuple : AuthTokenTuple) : void | Creates a resource-to-price mapping for resource costs. |
| buyResource | (resourceName : string, amount : int, authTokenTuple : AuthTokenTuple) : void | Buys a resource using the Leder service. |
| getBudget | (authTokenTuple : AuthTokenTuple) : Integer | Returns the current budget of the ISTS using the Ledger service. |
| createEvent | (spaceshipId : string, simulatedEvent : string, authTokenTuple : AuthTokenTuple) : void | Creates a simulated ISTS event as occurring. |
| defineFuel | (typeId : string, amount : int, authTokenTuple : AuthTokenTuple) : Fuel | Creates a Fuel object. Types can be solar sail, ion drive, or oxygen/hydrogen. |
| defineFuel | (typeId : string, authTokenTuple : AuthTokenTuple) : Fuel | Method that creates a Fuel object without an “amount” parameter. Types can be solar sail, ion drive, or oxygen/hydrogen. |
| getFuels | (authTokenTuple : AuthTokenTuple) : map<fuelId, Fuel> | Returns the Fuels. |
| getSpaceships | (authTokenTuple : AuthTokenTuple) : map<spaceshipId : Spaceship> | Returns the Spaceships. |
| getEntities | (authTokenTuple : AuthTokenTuple) : map<entityId : Entity> | Returns the Entities (Persons and Teams). |
| getLaunchpads | (authTokenTuple : AuthTokenTuple) : map<launchpadId : Launchpad> | Returns the Launchpads. |
| getCommunicationSystem | (authTokenTuple : AuthTokenTuple) : CommunicationSystem | Returns the CommunicationSystem. |
| getComputerSystem | (authTokenTuple : AuthTokenTuple) : ComputerSystem | Returns the ComputerSystem. |
| getResourcePrices | (authTokenTuple : AuthTokenTuple) : map<resourceId, Integer> | Returns the resource-to-price mappings for the resources. |
| getEntitiesVisitor | (authTokenTuple : AuthTokenTuple) : EntitiesVisitor | Returns an EntitiesVisitor object which contains the current state of the entities in the ISTS in the form of an inventory string. |

## Visitable (Interface)

The Resource Management Service implementation also implements the Visitable interface for use in the Visitor design pattern. Objects that implement Visitable are typically one of many objects that are accessed (i.e., “visited”) in some sort of pattern in order to be read or updated for some purpose by an object that implements the Visitor interface (defined later in this document).

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| acceptVisitor | (visitor : Visitor) : void | Calls the visit method on the given visitor that corresponds to a Visitable object passing in the object as a parameter. |

## ResourceImpl

The Resource Management Service implementation that implements the ResourceManagementService and Visitable interfaces.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| prices | map<resourceId, Integer> | The resource-to-price mappings for the resource prices. |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| entities | map<entityId, Entity> | The entities (extended by Person and Team) that make up the ISTS organization hierarchy. |
| launchpads | map<launchpadId, Launchpad> | The launchpads that spaceships use. |
| spaceships | map<spaceshipId, Spaceship> | The spaceships used by the ISTS. |
| communicationSystem | CommunicationSystem | The communication system used by the ISTS. |
| computerSystem | ComputerSystem | The computer system used by the ISTS |
| fuels | map<fuelId, Fuel> | The fuels that spaceships use. |
| ledger | Ledger | The Ledger Service that is used to buy resources and manage the ISTS budget. |
| authenticator | StoreAuthenticationService | The Authentication Service that is used for checking access permissions in every method. |

## ResourceImplException

The ResourceImplException is thrown when errors occur in the ResourceImpl. It extends java.lang.Exception. The exception includes the action that was being performed and the reason for the exception.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| action | string | Command performed during exception occurrence. |
| reason | string | Reason for exception being thrown. |
| exception | string | Name of the exception being thrown, e.g., “ResourceImplException”. |

## Spaceship

Represents the physical spaceship used in space flight travel in the ISTS. It can be one of three types of ships: Cargo, Passenger, and Rescue. It can send update events about its status through the Communication System and to the Resource Service. It can also be updated by external entities like the Flight Management Service.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| event | (perceivedEvent : string) : string[] | An event perceived or happening to the spaceship. |
| addFuel | (amount : int, authTokenTuple : AuthTokenTuple) : void | Adds fuel to the spaceship by the given amount. |
| consumeFuel | (amount : int, authTokenTuple : AuthTokenTuple) : void | Deducts the amount of fuel in the spaceship by the given amount. |
| getFuelLevel | (authTokenTuple : AuthTokenTuple) : void | Returns the fuel level of the spaceship. |
| setFuelLevel | (level : int, authTokenTuple : AuthTokenTuple) : void | Sets the fuel level of the spaceship by the given amount. |
| getCurrentSpeed | (authTokenTuple : AuthTokenTuple) : void | Returns the current speed of the spaceship. |
| setCurrentSpeed | (speed : int, authTokenTuple : AuthTokenTuple) : void | Sets the current speed of the spaceship by the given amount. |
| getTrajectory | (authTokenTuple : AuthTokenTuple) : void | Returns the trajectory of the spaceship. |
| setTrajectory | (trajectory : string, authTokenTuple : AuthTokenTuple) : void | Sets the trajectory of the spaceship by the given trajectory. |
| getCoordinates | (authTokenTuple : AuthTokenTuple) : void | Returns the coordinates of the spaceship. |
| setCoordinates | (coordinates : string, authTokenTuple : AuthTokenTuple) : void | Sets the coordinates of the spaceship with the given coordinates. |
| getCurrentFlightId | (authTokenTuple : AuthTokenTuple) : string | Returns the id of the flight the spaceship is currently associated with. |
| setCurrentFlightId | (flightId : string, authTokenTuple : AuthTokenTuple) : void | Sets the current flight id associated with the spaceship with the given id. |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string | The id of the spaceship. |
| model | string | The model of the spaceship. |
| name | string | The name of the spaceship. |
| maxSpeed | int | The maximum speed of the spaceship. |
| fuelType | string | Types can be solar sail, ion drive, or oxygen/hydrogen. |
| type | string | The type of the spaceship. Types can be cargo, passenger, or rescue. |
| fuelCapacity | int | The fuel capacity of the spaceship. |
| fuelLevel | int | The current fuel level of the spaceship. |
| description | string | A description of the spaceship. |
| currentSpeed | int | The current speed of the spaceship. |
| trajectory | string | The trajectory of the spaceship. |
| coordinates | string | The coordinates of the spaceship. |
| crewCapacity | int | The crew capacity of the spaceship. |
| passengerIds | string[] | The id’s of the passengers on the spaceship. |
| currentFlightId | string | The id of the flight the spaceship is currently associated with. |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| passengerShip | PassengerShip | A nested class within the Spaceship class that contains important information if the spaceship is a passenger type. |
| cargoShip | CargoShip | A nested class within the Spaceship class that contains important information if the spaceship is a cargo type. |
| rescueShip | RescueShip | A nested class within the Spaceship class that contains important information if the spaceship is a rescue type. |

## Cargo

The Cargo class is nested in the Spaceship class and is instantiated for Spaceships that are of the cargo type. It holds extra and important information and behavior for these types of spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| cargoType | string | The type of cargo in a cargo spaceship. Types can be mining, satellite maintenance, construction equipment. |

## Passenger

The Passenger class is nested in the Spaceship class and is instantiated for Spaceships that are of the passenger type. It holds extra and important information and behavior for these types of spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| capacity | int | The capacity of a passenger spaceship. |
| classType | string | The class type of a passenger spaceship. Types can be luxury, or economy. |

## Rescue

The Rescue class is nested in the Spaceship class and is instantiated for Spaceships that are of the rescue type. It holds extra and important information and behavior for these types of spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| capacity | int | Passenger capacity for a rescue spaceship. |

## *Entity*

Entity is an abstract class that is amenable to the usage of the Composite design pattern. It also implements the Visitable interface so that it can be integrated into the Visitor design pattern.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string | Unique id of the entity, e.g, “person 1”, or “team 1”. |
| name | string | The name of the entity. |
| description | string | Description of the entity. |

## Person

The Person class extends Entity and represents a person in the IST system and also a leaf object in the Composite pattern. As such, they can be nested within teams.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| role | string | The role of the person within the IST system. |

## Team

The Team class extends Entity and represents a group of people in the ISTS and also the composite in the Composite design pattern. As such, teams can be nested within other teams, and persons can be composed within teams.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addEntity | (entity : Entity, authTokenTuple : AuthTokenTuple) : void | Adds an entity to the team’s nested entities. |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| type | string | The team type. Types can be operations, flight crew, passenger, or rescue. |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| entities | map<entityId, Entity> | The entities nested within the team. |
| parentTeam | Team | The team’s parent team (if it has one). |

## Visitor (Interface)

Per the Visitor design pattern, the Visitor interface accesses (or “visits”) each of the resourceImpl’s entities in order to do something interesting with each such as read/write information or print information to stdout.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| visitResourceImpl | (resourceImpl : ResourceManagementService) : void | Accesses the resourceImpl and does something interesting. |
| visitTeam | (team : Team) : void | Accesses a team and does something interesting. |
| visitPerson | (person : Person) : void | Accesses a person and does something interesting. |

## EntitiesVisitor

The EntitiesVisitor class implements the Visitor interface. It takes an inventory of the current entities (Persons and Teams) in the ISTS and collects interesting information in the process such as person roles and team names. This could be useful in scheduling flights (done by the Flight Management Service), among other things.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| traverseTree | (entity : Visitable, level : int) : void | Traverse ResourceImpl’s tree of entities to visit each entity and recursively on Team entities. |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| baseLevel | int | The number of space indentations from the left margin for top-level entity objects in the entities tree (for readability when printed to stdout). |
| tabSpace | int | How many spaces are in one indentation. |
| levelPtr | int | A temporary pointer that tracks the levels in the entities structure. |
| inventory | string | The inventory of the entities (including any interesting information on them). |

## Launchpad

The Launchpad class represents where a spaceship will depart and arrive from.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string | The id of the launchpad. |
| name | string | The name of the launchpad. |
| location | string | The location of the launchpad. |

## Fuel

Represents the fuel spaceships use and the supply of that fuel for the ISTS.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addFuelSupply | (amount : int, authTokeTuple : AuthTokenTuple) : void | Adds to the ISTS’ resource supply of the fuel, e.g, when more fuel is purchased, by the given amount. |
| deductFuelSupply | (amount : int, authTokeTuple : AuthTokenTuple) : void | Deducts from the ISTS’ resource supply of the fuel, e.g., when spaceships are filled with fuel, by the given amount. |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| typeId | string | The type of the fuel. Types can be solar sail, ion drive, or oxygen/hydrogen. |
| amount | int | The amount of the fuel. |

## CommunicationSystem

This class represents an abstraction for large communication facilities. For the IST system to function, it’s important that these facilities are up and available so their status must be monitored.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| createEvent | (sourceShip : Spaceship, simulatedEvent : string) : string[] | Used by the Resource Management Service in the creation/simulation of events for interfacing with spaceships. |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string | The id of the communication system. |
| status | string | The status of the communication system. Statuse values can be up, or down. |
| upTime | integer | How long the system has been up. |

## ComputerSystem

This class represents an abstraction for large computer facilities. For the IST system to function, it’s important that these facilities are up and available so their status must be monitored.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string | The id of the computer system. |
| status | string | The status of the system. Statuse values can be up, or down. |
| upTime | Integer | How long the system has been up. |

# Implementation Details

The Resource Management Service is largely a modeling exercise. It’s a very stateful service that provides and oversees the resources in the ISTS system from and on which other components can act.

* Spaceship holds a list of passenger id’s which can be added to when passengers board the ship.
* Added crewCapacity attribute to Spaceship and capacity attribute to Rescue Ship.
* Added currentFlightId attribute to Spaceship.
* Visitor design pattern

# 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

In addition, unlike GUIs, CLI allows for scripts that can create class objects and state instantly and en masse. A GUI would require lots of manual mouse clicking and input on the user’s part.

# Risks

* The in-memory implementation makes the system prone to losing the state of the reources. It is not required but perhaps the implementation can be updated to make use of the IPFS like the Customer Service module does for long-term storage.
* Hackers may attempt to access the payment system. Transaction processing should be updated to require payer account to sign transactions with a secure signature algorithm.
* The spaceship and resources don’t have any functionality. Since events are opaque strings at this point, the system is inoperable and not self-sufficient. Functionality should be implemented to correct this.
* Improperly naming variables (not using unique id’s / improper name space practices could corrupt the integrity of your data)?
* Might not integrate well with other modules? The interface may need to be extended. There may have been vagueness in areas of the requirements.
* Programming language limitations?
* Governmental regulations and standards on resources like spaceships and fuel might might change unexpectedly.
* Customer may change strategy, priorities, or scope of project?

ISTS Customer Service Design Document

Document structure overview here.

# Introduction

*A high-level description of this document, for example, “This document defines the design for the Canonical Model”.*

Overview

*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.*

# Requirements

*This section provides a summary of the requirements for the <Component Name>.*

*Provide your understanding of the requirements, both functional and nonfunctional. Reference the provided Requirements and System Architecture documents. Do not cut and paste from the requirements document.*

*Product Manager and others can read this to understand what requirements your design will support. There is already a requirements doc, so keep this brief and to the point, highlighting the important requirements that the design is addressing. Structure in a way to provide a requirements checklist for your design.*

# Use Cases

*Enumerate the use cases supported by the design,*

*This design supports the following use cases:*

*Include a Use Case Diagram.*

*Include descriptions of each of the actors and use cases.*

# Implementation

*This section of the document will describe the implementation details for ...*

*The implementation section should cover the following topics:*

* *What are the classes, and their properties, associations and methods?*
* *What are the important interfaces and how they will be implemented?*
* *How are the requirements addressed?*

# Class Diagram

*The following class diagram defines the classes defined in this design. Remember to include exception classes.*

*CLASS DIAGRAM GOES HERE*

# Class Dictionary

*This section specifies the class dictionary for the class … defined within the package …*

## CustomerService (Interface)

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| pullFromIpfsRepo | (authTokenTuple : AuthTokenTuple) : void |  |
| pushToIpfsRepo | (authTokenTuple : AuthTokenTuple) : void |  |
| getIpnsKeyName | (authTokenTuple : AuthTokenTuple) : string |  |
| getFlights | (authTokenTuple : AuthTokenTuple) : map<flightId, Flight> |  |
| registerPassenger | (id : string, name : string, account : string, email : string, authTokenTuple : AuthTokenTuple) : Passenger |  |
| getPassengers | (authTokenTuple : AuthTokenTuple) : map<passengerId, Passenger> |  |
| addPassengerCredential | (passengerId : string, type : string, value : string, authTokenTuple : AuthTokenTuple) : void |  |
| definePointOfInterest | (id : string, name : string, type : string, description : string, location : string, authTokenTuple : AuthTokenTuple) : PointOfInterest |  |
| getPointsOfInterest | (authTokenTuple : AuthTokenTuple) : map<pointOfInterestId, PointOfInterest> |  |
| defineImage | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : Image |  |
| getImages | (authTokenTuple : AuthTokenTuple) : map<imageId, Image> |  |
| defineNote | (id : string, description : string, message : string, authTokenTuple : AuthTokenTuple) : Note |  |
| getNotes | (authTokenTuple : AuthTokenTuple) : map<noteId, Note> |  |
| bookFlight | (id : string, flightId : string, destination : string, passengerId : string, price : int, type : string, departureTime : string, returnTime : string, authTokenTuple : AuthTokenTuple) : FlightBooking |  |
| getFlightBookings | (authTokenTuple : AuthTokenTuple) : map<flightBookingId, FlightBooking> |  |
| defineTravelDoc | (id : string, flightNumber : string, ticketId : string, passengerName : string, destination : string, dateTime : string, price : int, boardPassIpnsKeyName : string, passportId : string, visaId : string, welcomePackageId : string, authTokenTuple : AuthTokenTuple) : TravelDocument |  |
| getTravelDocs | (authTokenTuple : AuthTokenTuple) : map<travelDocId, TravelDocument> |  |
| defineWelcomePackage | (id : string, name : string, description : string, authTokenTuple : AuthTokenTuple) : WelcomePackage |  |
| getWelcomePackages | (authTokenTuple : AuthTokenTuple) : map<welcomePackageId, WelcomePackage> |  |
| defineMovie | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : Movie |  |
| getMovies | (authTokenTuple : AuthTokenTuple) : map<movieId, Movie> |  |
| defineAudioRecording | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : AudioRecording |  |
| getAudioRecordings | (authTokenTuple : AuthTokenTuple) : map<audioRecordingId, AudioRecording> |  |
| defineVideoRecording | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : VideoRecording |  |
| getVideoRecordings | (authTokenTuple : AuthTokenTuple) : map<videoRecordingId, VideoRecording> |  |
| defineExperienceDocument | (id : string, name : string, description : string, authTokenTuple : AuthTokenTuple) : ExperienceDocument |  |
| getExperienceDocuments | (authTokenTuple : AuthTokenTuple) : map<experienceDocumentId, ExperienceDocument> |  |
| defineFeedback | (id : string, description : string, messge : string, authTokenTuple : AuthTokenTuple) : Note |  |
| getFeedback | (authTokenTuple : AuthTokenTuple) : map<feedbackId, Note> |  |
| defineBook | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : Book |  |
| getBooks | (authTokenTuple : AuthTokenTuple) : map<bookId, Book> |  |
| defineMusic | (ipnsKeyName : string, id : string, name : string, description : string, source : string, authTokenTuple : AuthTokenTuple) : Music |  |
| getMusic | (authTokenTuple : AuthTokenTuple) : map<musicId, Music> |  |
| defineMissionReport | (id : string, name : string, description : string, authTokenTuple : AuthTokenTuple) : MissionReport |  |
| getMissionReports | (authTokenTuple : AuthTokenTuple) : map<missionReportId, MissionReport> |  |
| defineDiscovery | (id : string, name : string, description : string, type : string, authTokenTuple : AuthTokenTuple) : Discovery |  |
| getDiscoveries | (authTokenTuple : AuthTokenTuple) : map<discoveryId, Discovery> |  |

## CustomerImpl

Implements CustomerService interface.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| hashCalculator | (originalString : string) : string |  |
| objectToJson | (object : Object) : string |  |
| jsonToObject | (json : string) : Object |  |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| repositoryIpnsKeyName | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| ledger | Ledger |  |
| resourceImpl | ResourceManagementService |  |
| authenticator | StoreAuthenticationService |  |
| passengers | map<passengerId, Passenger> |  |
| credentialUserIndexes | map<id, Passenger> |  |
| pointsOfInterest | map<id, PointOfInterest> |  |
| images | map<imageId, Image> |  |
| notes | map<noteId, Note> |  |
| flightBookings | map<id, FlightBooking> |  |
| travelDocuments | map<id, TravelDocument> |  |
| welcomePackages | map<id, WelcomePackage> |  |
| experienceDocuments | map<id, ExperienceDocument> |  |
| movies | map<movieId, Movie> |  |
| videoRecordings | map<id, VideoRecording> |  |
| audioRecordings | map<id, AudioRecording> |  |
| feedback | map<feedbackId, Note> |  |
| music | map<musicId, Music> |  |
| books | map<bookId, Book> |  |
| flights | map<flightId, Flight> |  |
| missionReports | map<id, MissionReport> |  |
| discoveries | map<id, Discovery> |  |

## CustomerImplException

The CustomerImplException is thrown when errors occur in the CustomerImpl. It extends java.lang.Exception. The exception includes the action that was being performed and the reason for the exception.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| action | string | Command performed during exception occurrence. |
| reason | string | Reason for exception being thrown. |
| exception | string | Name of the exception being thrown, e.g., “CustomerImplException”. |

## Flight

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| updatePassengerCount | (amount : int, authTokenTuple : AuthTokenTuple) : void |  |
| getStatus | (authTokenTuple : AuthTokenTuple) : string |  |
| setStatus | (status : string, authTokenTuple : AuthTokenTuple) : void |  |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| number | string |  |
| time | string |  |
| location | string |  |
| destination | string |  |
| duration | string |  |
| numStops | int |  |
| capacity | int |  |
| ticketPrice | int |  |
| passengerCount | int |  |
| status | string | {preparing for launch, in-flight, reached destination, lost:mission, lost:spacecraft fault} |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| spaceship | Spacheship |  |
| crewId | Team |  |

## Passenger

*Class 1 description*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| name | string |  |
| account | string |  |
| email | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| credentials | map<credentialId, Credential> |  |

## FlightBooking

*Class 1 description*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| destination | string |  |
| price | int |  |
| type | string | {one way, round trip, guided tour} |
| departureTime | string |  |
| returnTime | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| flight | Flight |  |
| passenger | Passenger |  |

## Medium

Abstract class

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| ipnsKeyName | string |  |
| id | string |  |
| name | string |  |
| description | string |  |
| source | string |  |

## Image

Extends Medium

## Movie

Extends Medium

## Music

Extends Medium

## Book

Extends Medium

## VideoRecording

Extends Medium

## AudioRecording

Extends Medium

## Note

*Class 1 description*

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| description | string |  |
| message | string |  |

## Document

Abstract class

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addNote | (note : Note, authTokenTuple : AuthTokenTuple) : void |  |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| name | string |  |
| description | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| notes | map<noteId, Note> |  |

## WelcomePackage

Extends Document

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addImage | (image : Image, authTokenTuple : AuthTokenTuple) : void |  |
| addMovie | (movie : Movie, authTokenTuple : AuthTokenTuple) : void |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| images | map<imageId, Image> |  |
| movies | map<movieId, Movie> |  |

## ExperienceDocument

Extends Document

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addImage | (image : Image, authTokenTuple : AuthTokenTuple) : void |  |
| addVideoRecording | (videoRecording : VideoRecording, authTokenTuple : AuthTokenTuple) : void |  |
| addAudioRecording | (audioRecording : AudioRecording, authTokenTuple : AuthTokenTuple) : void |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| images | map<imageId, Image> |  |
| videoRecordings | map<id, VideoRecording> |  |
| audioRecordings | map<id, AudioRecording> |  |

## MissionReport

Extends ExperienceDocument

## Discovery

Extends ExperienceDocument

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| type | string | {life, minerals, object} |

## PointOfInterest

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addImage | (image : Image, authTokenTuple : AuthTokenTuple) : void |  |
| addNote | (note : Note, authTokenTuple : AuthTokenTuple) : void |  |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| name | string |  |
| type | string | {planet, moon, asteroid, solar system, space station} |
| description | string |  |
| location | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| images | map<imageId, Image> |  |
| notes | map<mapId, Note> |  |

## TravelDocument

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addPassportId | (passengerId : string, passportId : string, authTokenTuple : AuthTokenTuple) : void |  |
| addVisaId | (passengerId : string, passportId : string, authTokenTuple : AuthTokenTuple) : void |  |
| addWelcomePackage | (welcomePackage : WelcomePackage, authTokenTuple : AuthTokenTuple) : void |  |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | string |  |
| flightId | string |  |
| ticketId | string |  |
| passengerName | string |  |
| destination | string |  |
| dateTime | string |  |
| price | int |  |
| boardPassIpnsKeyName | string |  |
| passportId | string |  |
| visaId | string |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| welcomePackages | map<id, WelcomePackage> |  |

## ObjectFactory

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| getObject | (json : string) : Object | A static method |

# Implementation Details

* CustomerImpl has a reference to ResourceImpl so that it can add a registered passenger to the the persons in ResourcImpl if need be. Also, so it can use ResourceImpl’s createEvent or its communication system to notify observers (i.e., the manager) of interesting mission reports, discoveries, etc.?

# 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

* Hackers may try to access resources such as sending update actions to the spaceship.
* Keeping the Customer Service logged in perpetually
* Validating parameters and data before submitting to IPFS

ISTS Flight Management Service Design Document

Document structure overview here.

# Introduction

*A high-level description of this document, for example, “This document defines the design for the Canonical Model”.*

Overview

*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.*

# Requirements

*This section provides a summary of the requirements for the <Component Name>.*

*Provide your understanding of the requirements, both functional and nonfunctional. Reference the provided Requirements and System Architecture documents. Do not cut and paste from the requirements document.*

*Product Manager and others can read this to understand what requirements your design will support. There is already a requirements doc, so keep this brief and to the point, highlighting the important requirements that the design is addressing. Structure in a way to provide a requirements checklist for your design.*

# Use Cases

*Enumerate the use cases supported by the design,*

*This design supports the following use cases:*

*Include a Use Case Diagram.*

*Include descriptions of each of the actors and use cases.*

# Implementation

*This section of the document will describe the implementation details for ...*

*The implementation section should cover the following topics:*

* *What are the classes, and their properties, associations and methods?*
* *What are the important interfaces and how they will be implemented?*
* *How are the requirements addressed?*

# Class Diagram

*The following class diagram defines the classes defined in this design. Remember to include exception classes.*

*CLASS DIAGRAM GOES HERE*

# Class Dictionary

*This section specifies the class dictionary for the class … defined within the package …*

## FlightManagementService (Interface)

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| defineFlight | (id : string, number : string, spaceshipId : string, time : string, location : string, destination : string, duration : string, numStops : int, capacity : int, crewId : string, ticketPrice : int, passengerCount : int, authTokenTuple : AuthTokenTuple) : Flight |  |

## Observer (Interface)

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| update | (event UpdateEvent): void |  |

## Manager

Implements FlightManagementService and Observer interfaces.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| handleEvent | (event : UpdateEvent) : void |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| resourceImpl | ResourceManagementService; Subject |  |
| customerImpl | CustomerService |  |
| authenticator | StoreAuthenticationService |  |
| authToken | AuthToken | The Manager needs its own AuthToken with the proper access permissions to interface with the other modules such as when defining flights. |

## ManagerException

The ManagerException is thrown when errors occur in the Manager. It extends java.lang.Exception. The exception includes the action that was being performed and the reason for the exception.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| action | string | Command performed during exception occurrence. |
| reason | string | Reason for exception being thrown. |
| exception | string | Name of the exception being thrown, e.g., “ManagerException”. |

## Subject (Interface)

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| registerObserver | (observer : Observer) : void |  |
| deregisterObserver | (observer : Observer) : void |  |
| notifyObservers | (sourceDevice : Spaceship, eventToSend : string[]) : void |  |

## ResourceImpl

Implements Subject

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| createEvent | (spaceshipId : string, simulatedEvent : string, authTokenTuple : AuthTokenTuple) : void |  |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| observers | Observer[] |  |
| spaceships | map<spaceshipId, Spaceship> |  |
| communicationSystem | CommunicationSystem |  |
| authenticator | StoreAuthenticationService | Used to check access permissions on the ResourceImpl object’s methods. |

## CommunicationSystem

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| createEvent | (sourceDevice : Spaceship, simulatedEvent : string) : string[] |  |

## Spaceship

*Class 1 description*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| event | (perceivedEvent : string) : string[] |  |

## UpdateEvent

The UpdateEvent class represents an event that is sent from a Subject (e.g., the Resource Management Service) to an Observer (e.g., the Manager) defined by the Observer pattern. It is created in the notifyObservers method of the Subject when it calls an Observer’s update method.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| perceivedEvent | string[] | The event that the source Spaceship perceived and sent. |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| sourceShip | Spaceship | The Spaceship that sent the originating event to the Communication System (and ultimately to the Resource Management Service) for notifying observers. |

## *Action*

The Action class encapsulates the actions that need to be performed in its execute method in response to an event including the update actions sent to a spaceship, and the Resource Management and Customer services. It is an abstract class that each event extends with a corresponding Action subclass. When a new Action is made it is executed right away.

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | () : void | Defines the executable actions that are needed in response to an event. |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| sourceShip | Spaceship | The source spaceship that perceived/sent the original event. |
| resourceImpl | ResourceManagementService | A reference to the Resource Management Service that the self-sufficient Action classes depend on. |
| customerImpl | CustomerService |  |
| manager | FlightManagementService |  |
| authToken | AuthToken | To perform their functions the Action classes need an authToken with the appropriate access permissions. |

## EmergencyAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| emergencyType | string |  |

## StatusUpdateAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| status | string |  |

## ReachedDestinationAction

Extends Action.

## LocationUpdateAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| speed | int |  |
| trajectory | string |  |
| coordinates | string |  |

## FuelLevelAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| fuelLevel | int |  |

## MissionReportAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| missionReportInfo | string |  |

## DiscoveryAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| discoveryInfo | string |  |

## CustomerFeedbackAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| feedbackInfo | string |  |

## BoardPassengerAction

Extends Action.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| passengerId | string |  |

# Implementation Details

* Resources are mostly physical in nature
* Mostly a modeling exercise

*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

* Keeping the Customer Service logged in perpetually
* Validating parameters and data before submitting to IPFS