*Resource Management System* Design Document

# Introduction

*This section defines one of the modules that make up the Interplanetary Space Transport System, the Resource Management System. Resource Management System, RMS in short, keeps the inventory of resources that administrators can view and update. RMS depends on the Authentication Service and Ledger Service previously developed for granting access to verified users and budgeting needs respectively.*

Overview

It is crucial for administrators to keep track of resources to avoid waste and to have a system of records to refer to. Administrators will also have an easily accessible user interface that accommodates non-technical audience. RMS also provides a model for the involved objects in real life making it easy to picture the interactions. A well designed and robust resources manager system saves the manual hours needed to perform bookkeeping. It is also accessible over the wire from anywhere adding to convenience and ease of use. RMS is one of the 6 modules in the ISTS system as shown below.

A screenshot of a map

Description automatically generated

# 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 defined under ‘com.cscie97.ists.rms’.*

## *IResourceManagementSystem*

*This interface defines the service API that is accessible by the GUI and other interacting services in the ISTS system. This interface serves as an orchestration engine and an entry point for the RMS module. All the methods in this interface are protected by the Authentication Service and a valid auth token must be passed to perform any task. Upon receiving a token, the methods delegate validation to the Authentication Service prior to moving forward with the action. An AccessDeniedException is thrown when tokens are invalid, or the required permissions are not granted. This is also a restful API. It is independently deployable and possibly in a different machine than the other modules that RMS interacts with. A ResourceNotFoundException is also expected when a user tries to access a non-existent resource. UnableToUpdateResourceException is also being thrown when updating a resource fails.*

***Methods***

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Signature** | **URL** | **Description** |
| getInstance | (): IResourceManagementSystem | /api/rms  POST | Singleton pattern – This method is invoked first to get a singleton instance of this service |
| createSpaceship | (name: string, fuel: long, speed: long, model: string, cap: long, desc: string, authToken: string): Spaceship | /api/rms/spc/    POST | Defines a new Spaceship object |
| updateSpaceshipType | (spcId : string, type : SpaceshipTypeEnum, authToken : string) : Spaceship | /api/rms/spc/{spcShipId}  PUT | Updates the spaceship type. The same spaceship can be repurposed for rescue, cargo or passenger use. |
| updateSpaceshipBudget | (spcId: string, balance: int, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can allocate funds to a spaceship |
| updateLaunchpad | (spcId: string, launchpad: Launchpad, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can adjust the Launchpad for a spaceship |
| updateFuelType | spcId: string, fuelType: FuelTypeEnum, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Spaceships can be adjusted to be refueled by solar ceil, ion drive or hydrogen/oxygen |
| updateFuel | (fuel: long, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can fill up spacecrafts with fuel |
| deleteSpaceship | (spcId: string, authToken: string): boolean | /api/rms/spc/{spcShipId}  DELETE | Administrators can decommission spaceships when they are out of use. Returns whether decommissioning is successful |
| getSpaceshipById | (spcId: string, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  GET | Gets spaceship by id |
| getSpaceshipBudget | (spcId: string, authToken: string): Budget | /api/rms/spc/{spcShipId}/budget  GET | Gets budget associated with a spaceship. This budget is not the overall budget that the ledger service keeps track of and there is no need to delegate to the ledger service to find out about this budget |
| getLaunchpad | (spcId : string, lpdNumber: string, authToken: string): Launchpad | /api/rms/spc/{spcShipId}/lpad/{lpdNumber}  GET | Gets the current launchpad associated a spaceship |
| createPerson | (personId: string, personRole: string, authToken: string): Person | /api/rms/ppl/person  POST | Composite pattern - Creates a new person. Person is a leaf object that can exist standalone or be part of a team |
| createTeam | (teamName: string, authToken: string): Team | /api/rms/ppl/team/  POST | Composite pattern - Defines a new team. The team doesn’t consist of any members when defined. Instances of person and other teams gradually get added to it |
| addPeopleToTeam | (personId: string, teamName: string, authToken: string): Team | /api/rms/ppl/team/{temName}/person/{personId}  PUT | Composite pattern - Members can be assigned to teams once both are defined |
| addChildTeamToTeam | (parentTeamName: string, childTeamName: string, authToken: string): Team | /api/rms/ppl/team/{temName}/childTeam/{temName}  PUT | Composite pattern – Teams can consist of other teams |
| validateToken | (authToken: string): boolean | /api/rms/token/{authToken}  GET | This method delegates to the Authentication Service to check whether the token is valid and has the required permissions. It is used internally by all of the methods in this interface |
| getBudgetFromLedger | (accountId: string): Budget | /api/rms/budget  GET | RMS is assigned an account id by the ledger service during bootstrapping session. This is the overall budget and administrators can allocate funds for specific spaceships |
| assignBudgetToSpacehip | (spaceshipId: string, budget: Budget): Spaceship | /api/rms/spc/{spaceshipId}/budget  PUT | Administrators can assign budget to spaceships preparing for flight. Note that the budget can not exceed the overall budget allocated for all the spaceships |
| getAvailableFlights | (): List<Flight> | /api/rms/spc/flights  GET | Every time Flight Management System schedules a new flight, the details are sent to RMS for storage. Administrators can look up which flights are listed. This is for a convenience to get a one stop representation in the RMS GUI. |
| getFuelInventory | (): List<Fuel> | /api/rms/spc/fuel  GET | Administrators can view the available fuel in each of the spacecrafts and in storage |
| addCrewToFlight | (flightNum: string, people: People): Flight | /api/rms/flight/{flightNum}/ppl | Administrators can assign passengers and staff to a space |
| sendMessageToComputerSystem | (computerId: string, message: string): ComputerSystem | /api/rms/computer/{computerId}  POST | Observer and proxy pattern - RMS has Computer Systems in ground that is a proxy to a Communication System in space that gets notified by the spaceship. The computer system leverages the observer pattern to notify the observers which in this case is the Flight Management System. The observers communicate back to the Computer System through RMS. This method is to send messages back to the Computer System.  It returns the computer system it communicated with future reference. |
| reachDestination | (flightNum: string): Flight | /api/rms/flight/{flightNum}/reach  DELETE | When a spaceship is done with its mission and reaches destination, it is taken off the available flight list |

## *ResourceManagementSystem*

This class is an implementation of the IResourceManagementSystem interface from above. The associations and the contained resources are discussed here.

Note that during the change of status in a spaceship the flow is

Spaceship -> Communication System -> Computer System -> FMS

And replying of messages is done through RMS

FMS -> RMS -> Computer System -> Communication System -> Spaceship

***Methods***

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Signature** | **URL** | **Description** |
| getInstance | (): IResourceManagementSystem | /api/rms  POST | Singleton pattern – This method is invoked first to get a singleton instance of this service |
| createSpaceship | (name: string, fuel: long, speed: long, model: string, cap: long, desc: string, authToken: string): Spaceship | /api/rms/spc/    POST | Defines a new Spaceship object |
| updateSpaceshipType | (spcId : string, type : SpaceshipTypeEnum, authToken : string) : Spaceship | /api/rms/spc/{spcShipId}  PUT | Updates the spaceship type. The same spaceship can be repurposed for rescue, cargo or passenger use. |
| updateSpaceshipBudget | (spcId: string, balance: int, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can allocate funds to a spaceship |
| updateLaunchpad | (spcId: string, launchpad: Launchpad, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can adjust the Launchpad for a spaceship |
| updateFuelType | spcId: string, fuelType: FuelTypeEnum, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Spaceships can be adjusted to be refueled by solar ceil, ion drive or hydrogen/oxygen |
| updateFuel | (fuel: long, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  PUT | Administrators can fill up spacecrafts with fuel |
| deleteSpaceship | (spcId: string, authToken: string): boolean | /api/rms/spc/{spcShipId}  DELETE | Administrators can decommission spaceships when they are out of use. Returns whether decommissioning is successful |
| getSpaceshipById | (spcId: string, authToken: string): Spaceship | /api/rms/spc/{spcShipId}  GET | Gets spaceship by id |
| getSpaceshipBudget | (spcId: string, authToken: string): Budget | /api/rms/spc/{spcShipId}/budget  GET | Gets budget associated with a spaceship. This budget is not the overall budget that the ledger service keeps track of and there is no need to delegate to the ledger service to find out about this budget |
| getLaunchpad | (spcId : string, lpdNumber: string, authToken: string): Launchpad | /api/rms/spc/{spcShipId}/lpad/{lpdNumber}  GET | Gets the current launchpad associated a spaceship |
| createPerson | (personId: string, personRole: string, authToken: string): Person | /api/rms/ppl/person  POST | Composite pattern - Creates a new person. Person is a leaf object that can exist standalone or be part of a team |
| createTeam | (teamName: string, authToken: string): Team | /api/rms/ppl/team/  POST | Composite pattern - Defines a new team. The team doesn’t consist of any members when defined. Instances of person and other teams gradually get added to it |
| addPeopleToTeam | (personId: string, teamName: string, authToken: string): Team | /api/rms/ppl/team/{temName}/person/{personId}  PUT | Composite pattern - Members can be assigned to teams once both are defined |
| addChildTeamToTeam | (parentTeamName: string, childTeamName: string, authToken: string): Team | /api/rms/ppl/team/{temName}/childTeam/{temName}  PUT | Composite pattern – Teams can consist of other teams |
| validateToken | (authToken: string): boolean | /api/rms/token/{authToken}  GET | This method delegates to the Authentication Service to check whether the token is valid and has the required permissions. It is used internally by all of the methods in this interface |
| getBudgetFromLedger | (accountId: string): Budget | /api/rms/budget  GET | RMS is assigned an account id by the ledger service during bootstrapping session. This is the overall budget and administrators can allocate funds for specific spaceships |
| assignBudgetToSpacehip | (spaceshipId: string, budget: Budget): Spaceship | /api/rms/spc/{spaceshipId}/budget  PUT | Administrators can assign budget to spaceships preparing for flight. Note that the budget can not exceed the overall budget allocated for all the spaceships |
| getAvailableFlights | (): List<Flight> | /api/rms/spc/flights  GET | Every time Flight Management System schedules a new flight, the details are sent to RMS for storage. Administrators can look up which flights are listed. This is for a convenience to get a one stop representation in the RMS GUI. |
| getFuelInventory | (): List<Fuel> | /api/rms/spc/fuel  GET | Administrators can view the available fuel in each of the spacecrafts and in storage |
| addCrewToFlight | (flightNum: string, people: People): Flight | /api/rms/flight/{flightNum}/ppl | Administrators can assign passengers and staff to a space |
| sendMessageToComputerSystem | (computerId: string, message: string): ComputerSystem | /api/rms/computer/{computerId}  POST | Observer and proxy pattern - RMS has Computer Systems in ground that is a proxy to a Communication System in space that gets notified by the spaceship. The computer system leverages the observer pattern to notify the observers which in this case is the Flight Management System. The observers communicate back to the Computer System through RMS. This method is to send messages back to the Computer System.  It returns the computer system it communicated with future reference. |
| reachDestination | (flightNum: string): Flight | /api/rms/flight/{flightNum}/reach  DELETE | When a spaceship is done with its mission and reaches destination, it is taken off the available flight list |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| spaceships | List<Spaceship> | Contains one to many spaceships that can be actively assigned for a flight |
| computerSystems | List<ComputerSystem> | Contains a list of computer systems on the ground that can be communicated back. This if for the purposes of replying to messages only. The Computer System notifies its observer, the Flight Management System, through the observer pattern. Reply is done like  FMS -> RMS -> Computer System-> Communication System -> Spacecraft |
| availableFlights | List<Flight> | Returns the list of available flights that have not boarded yet |
| fuelInventory | List<Fuel> | The overall fuel in RMS including all of the spaceships |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| ledgerService | LedgerService | Delegates checking the overall budget of RMS to the ledger service |
| authenticationService | IAuthenticationService | Delegates validating tokens and inquiring for permissions to the Authentication Service |

## *Spaceship*

This class represents the Spaceship object. A catalogue of spaceships both in the ground and space is maintained.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| spaceshipName | string | Name of spaceship |
| spaceshipId | string | A unique UUID is assigned to spaceship by the system when a spaceship is defined |
| spaceshipModel | string | Model of spaceship |
| maxSpeed | long | Maximum speed of spaceship |
| fuelCapacity | long | Fuel capacity of spaceship |
| spaceshipDesc | string | Description of spaceship |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| spaceshipType | SpaceshipTypeEnum | A spaceship can be of type rescue, cargo or passenger. However, it is possible to repurpose a spaceship between any of the categories |
| fuel | Fuel | Current fuel of spaceship |
| budget | Budget | Budget allocated to this specific spaceship by administrators |
| launchPad | Launchpad | The designated location where the spaceship will take off. This location won’t be specified for spaceships at rest |
| fuelType | FuelTypeEnum | Spaceships can be fueled by solar ceil, ion drive or hydrogen/oxygen |

## *SpaceshipTypeEnum*

This defines types of spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| cargo | CargoEnum | Cargo type maybe of different subtypes defined below |
| passenger | PassengerEnum | Passenger type maybe of different subtypes defined below |
| isRescue | boolean | A spaceship may or may not be used for rescue |

## *CargoEnum*

This defines types of cargo spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| MINING | string | Cargo spaceship can be used for mining |
| SATELLITE\_MAINTENANCE | string | Cargo spaceship can be used for satellite maintenance |
| CONSTRUCTION\_EQUIPMENT | string | Cargo spaceship can be used for construction |

## *PassengerEnum*

This defines types of passenger spaceships.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| capacity | int | The maximum number of passengers the spaceship can carry |
| CLASS\_LUXURY | string | Passengers may fly in luxury class |
| CLASS\_ECONOMY | string | Passengers may fly in economy class |

## *FuelTypeEnum*

This defines types of fuel types for a spaceship

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| SOLAR\_CEIL | string | Spaceship powered by solar ceil |
| IONDRIVE | string | Spaceship powered by ion drive |
| HYDROGEN\_OXYGEN | string | Spaceship powered by hydrogen/oxygen |

## *Launchpad*

This is where a spaceship is planned to take off. The service API provides method to update the launchpad associated with a spaceship

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| location | string | Location of launchpad |
| lpdName | string | Launchpad name |
| lpdNumber | string | Launchpad number |

## *Budget*

This is budget associated with a spaceship that administrators can allocate. It may not exceed the total budget allocated to RMS by the ledger service.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| budget | int | Available budget |

## *Fuel*

This is the current fuel level of the spaceship. Administrators can increase the amount of fuel using the service API.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| fuel | long | Current fuel |

## *Fuel*

This is the fuel level of the entire RMS. Administrators can increase the amount of fuel using the service API.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| fuel | long | Fuel for whole RMS |

## *Flight*

*This object is instantiated when a new flight is scheduled. The Flight Management System dumps new flights to RMS when they are 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 |

## *People*

This is the top-level abstract class leveraging the composite pattern to build a hierarchy of teams and persons.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| name | string | Name of a person or a team |
| peopleType | PeopleTypeEnum | People can be in operations, flight crew, rescue or maybe passengers |

## *Team*

This is the composite object in the hierarchy of people.

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| people | List<People> | Team consists of people. People can be teams or persons |

## *Person*

This a leaf object with id and role.

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| personId | string | ID of a person |
| personRole | string | Role of the person |

## *PeopleTypeEnum*

This defines types of people

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| OPERATIONS | string | People can be in operations |
| FLIGHT\_CREW | string | People can be members of the flight crew like pilots and flight attendants |
| PASSENGER | string | People can be passengers |
| RESCUE | string | People can be in the rescue team |

## *IComputerCommunicationSystem*

*This is a common interface for Computer System and Communication System. It leverages the Proxy pattern. Communication System is a remote object directly communicating with the spacecraft and it delegates to the Computer System on the ground which notifies its observer, the Flight Management System. The response from the ground is communicated back through the Resource Management System to the Computer System then the Communication System all the way to Spaceship.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| onMessage | (message: string): void | Listens to spaceship updates |

## *CommunicationSystem*

*It leverages the Proxy pattern to listen to aircraft messages from space and sends to the Computer System on the ground.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| onMessage | (message: string): void | Listens to spaceship updates from space and sends to the Computer System on the ground |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| status | boolean | Up or down health indicator |
| upTime | long | Time since it was last down |
| commId | string | IP address of the communication system |

## *ComputerSystem*

*It leverages the Proxy pattern to listen to aircraft messages from space and sends to the Computer System on the ground. It implements the ISubject interface of the observer pattern.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| onMessage | (message: string): void | Listens to spaceship updates from communication system. Makes an instance of Event with the payload and calls notify() internally |
| register | (observer: IObserver): void | Registers observers |
| deregister | (observer: IObserver): void | Deregister observers |
| notify | (event: Event): void | Notifies observers passing in payload |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| status | boolean | Up or down health indicator |
| upTime | long | Time since it was last down |
| compId | string | IP address of the computer system |

## *Event*

This has a payload that get sent to observers

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| payload | string | Payload sent from spaceship |

## *ISubject*

*Leverages the observer pattern to propagate messages to listening services on the ground.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| register | (observer: IObserver): void | Registers observers |
| deregister | (observer: IObserver): void | Deregister observers |
| notify | (event: Event): void | Notifies observers passing in payload |

## *IObserver*

*The listener in observer pattern.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| update | (event: Event): void | Called by the subject when there is status update |

## *UnableToUpdateResourceException*

This exception is thrown during failure to update a resource which can be people, spacecrafts or flights

***Properties***

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

## *AccessDeniedException*

This is thrown when a token is invalid or doesn’t have the required permissions.

***Properties***

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

## *ResourceNotFoundException*

This exception is thrown when the resource an administrator inquires about a resource that doesn’t exist.

***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?*