

Interplanetary Space Transport System Design Document

Date: 12/18/2019

Author: Tofik Mussa

Reviewer(s): Trisha Singh, Peter Chen

Introduction

The Interplanetary Space Agency is launching a new project to open up exploring space to the bigger audience for entertainment purposes, to generate income for expansion of other projects, to save the human race from an inevitable destruction due to climate change and to aggressively facilitate the discovery of extraterrestrial beings. This design proposes a full-blown orchestration system for the new Interplanetary Space Transport System, in short ISTS.

Overview

There is a need to open a portal for customers to engage with the current happenings in space. The system should serve customers who want to take part in space explorations or donate to the cause. There must also be a monitoring system on the ground to mitigate risks associated with emergencies and to keep track of allocated resources. Since security is a major concern, customers and operators must be identified as having the correct permissions to correct tasks. There must also be a decentralized system to perform transactions instead of traditional banking.

These problems were identified, and the system needs to be highly decoupled, modular architecture with clearly defined boundaries. The interaction between the component should resemble real world interactions from the user's standpoint. It also should have an easily accessible user interface that accommodates nontechnical users.

System Architecture

A successful execution of the requirements will result in reusable systems that are easily extensible but unlikely to change. The dependencies between the modules must be leveled and cyclical dependencies are to be avoided. The system must be abstract and stable. Taking these requirements into consideration 6 modules were identified to model the system: **Flight Management System, Resource Management System, Customer Management System, Interplanetary File System, Authentication Service and Ledger Service**. The first three of the systems are discussed in these design document while the last three are previously built and are briefly discussed whenever it is necessary to mention an interaction. All the modules are illustrated in the component diagram below. Please note that cyclical dependencies are avoided, and the interactions are leveled. The clients for the services can be other services in the ISTS system or graphical user interfaces. The modules are also independently deployable restful web services

thus achieving level 5 of the modularity maturity level.

Resource Management System – Serves as a system of record of resources that enables administrators and staff manage

Customer Management System – portal for managing the relationship between customers

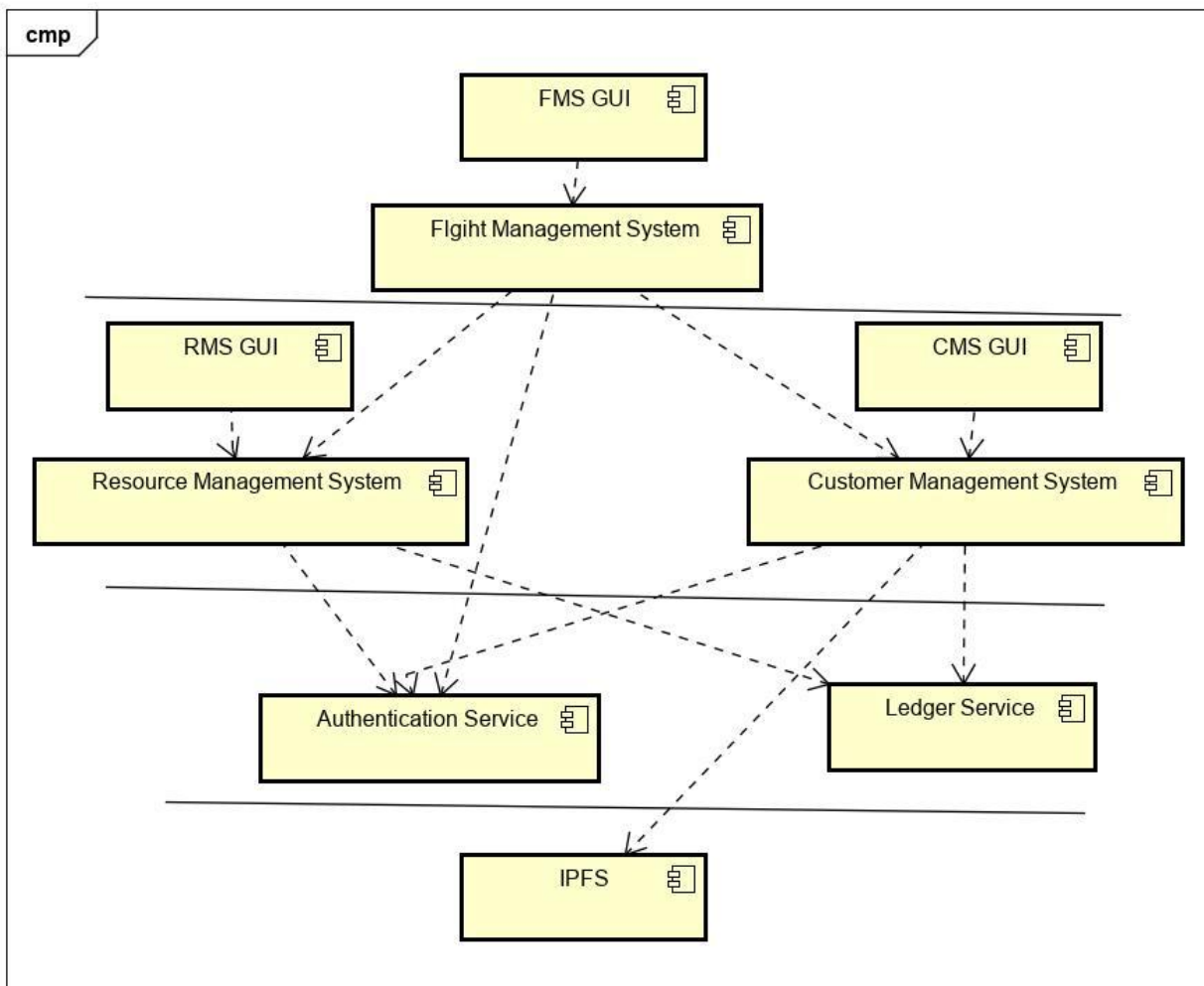
Flight Management System – real time information about the expedition is accessible here

Interplanetary File System – a decentralized system used in lieu of a database capable of storing files related customer documents, reports and discoveries. It can also stream multimedia

Authentication Service – most resources are protected and are accessible by only authorized users

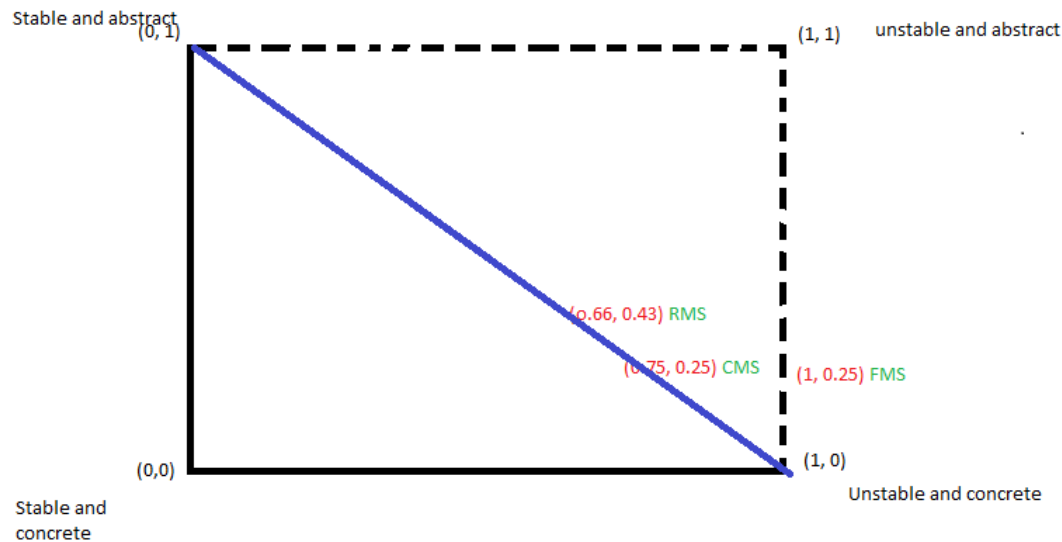
Ledger Service – a decentralized ledger system used to process transactions related to flight booking or allocating budget

The modules will be discussed in the following order: RMS -> CMS -> FMS



Instability Analysis

See the instability metrics below to measure module dependencies and the degree of abstraction



Requirements

As outlined by the System Architecture section above, level 5 modularity should be attained to promote high cohesion, loose coupling and independent deployment. The system should be able to scale horizontally based on load and the time it takes to spin up an instance of a highly frequented module should be minimal. The implementation must be performant and readable enough to be handed over a new team for maintenance. Cross cutting concerns like security must be mitigated all while handling the business requirements. Redundancy and load balancing must be put in place to achieve the highest level of fault tolerance. To rip the benefits of elegant design and to avoid reinventing the wheel, the use of design patterns must be promoted as much as possible.

Besides the nonfunctional requirements outlined above the following business workflows that are referenced in the products requirements must be incorporated into the system.

- 1) **Manage spacecrafts** – Spacecrafts must be defined, assigned flights, their routes must be planned and monitored. Infrastructure for inter-spacecraft communication must be put in place as well as for spaceships with operators and staff on the ground. Spaceships out of use can be decommissioned and spaceships can also be upgraded to use the latest appliances.
- 2) **Manage crew (Operators, staff, pilots, flight attendants and rescue team)** – Employees of the ISTS system should be empowered and effectively assigned tasks to serve customers with the highest standards and to advance mankind in the extraterrestrial world.

- 3) **Provision flights** – The goal is to have successful and strategic flights. The system must be dependable for customers to trust ISTS on their safety and emergencies must be addressed swiftly when they occur. Potential discovery sites must be prioritized to maximize the yield.
- 4) **Customer portal** – there must be a portal to register new customers and to store their payment information. Customers should also be able to express their interest on what to visit and the system should notify customers of upcoming flights aligning to their interest. As our intention is to serve customers better, their feedback is important to us and customers must be able to provide feedback on their experience.
- 5) **Reporting and analytics** – Mission reports must be stored and used to extract useful insights leading towards future discoveries. Reports can be instigated by passengers or the flight crew and permanently stored in a reliable system like IPFS or a database.
- 6) **Manage funds** – Donations, government subsidies and income generated through flights must be effectively allocated to continually sustain the project. There must be checks and balances on administering funds. Budgets must also be transparent to the public.
- 7) **User friendly GUI** – There must be no assumptions that users of our GUI are tech savvy. Our graphical user interface must be minimalist and intuitive.
- 8) **Document Management** – Reliable storage of all documents including boarding passes, passports, visas and reports
- 9) **Entertainment** – Streaming of content to keep passengers engaged

Use Cases

Interplanetary Space System supports the following use cases.

1) **Provisioning flights**

- The staff which includes operators, flight attendants, rescue team, pilots and administrators work in tandem to ensure successful takeoff and landing

2) **Managing resources**

- Resources like spaceships, documentation and all the people involved are stored in the system of records. All the CRUD (Create, Read, Update, Delete) operations are supported by the service APIs to enable the manipulation of data by authorized individuals

3) **Authentication and registering**

- It provides support to register and identify users

4) **Uploading/downloading documents related to flight/discovery**

- Using IPFS anyone authorized individual can seamlessly upload or download documents

5) **Inter-spaceship communication and the ground**

- Spaceships can communicate with each other and send health updates to monitors in the ground

6) **Booking and cancelling flights**

- Customers can book or cancel flights

7) **Reporting/addressing emergencies**

- Spaceships can report emergencies and a rescue spaceship can be assigned to address it

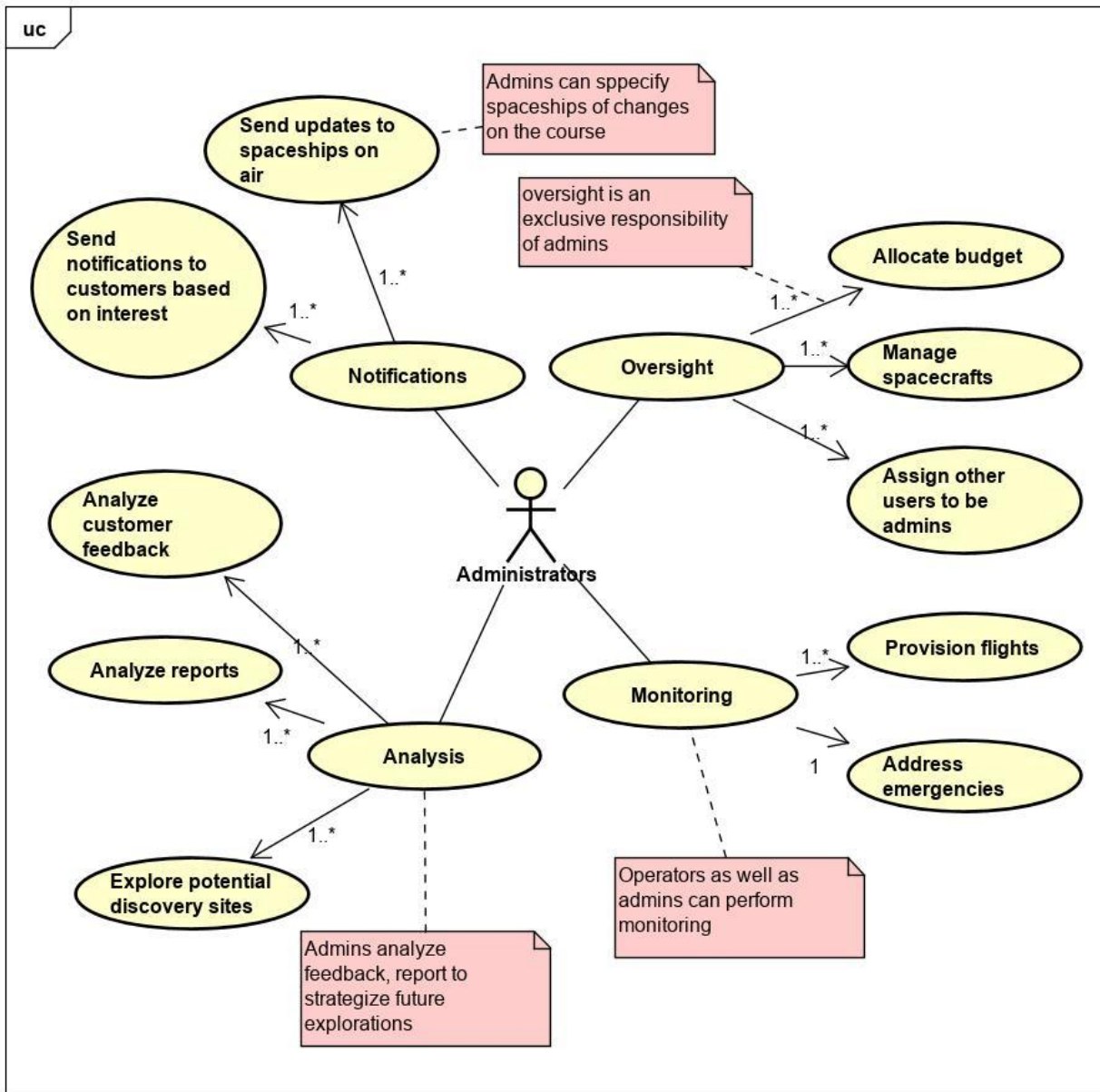
8) **Entertainment**

- Entertainment can be streamed for customers when they request it

Actors

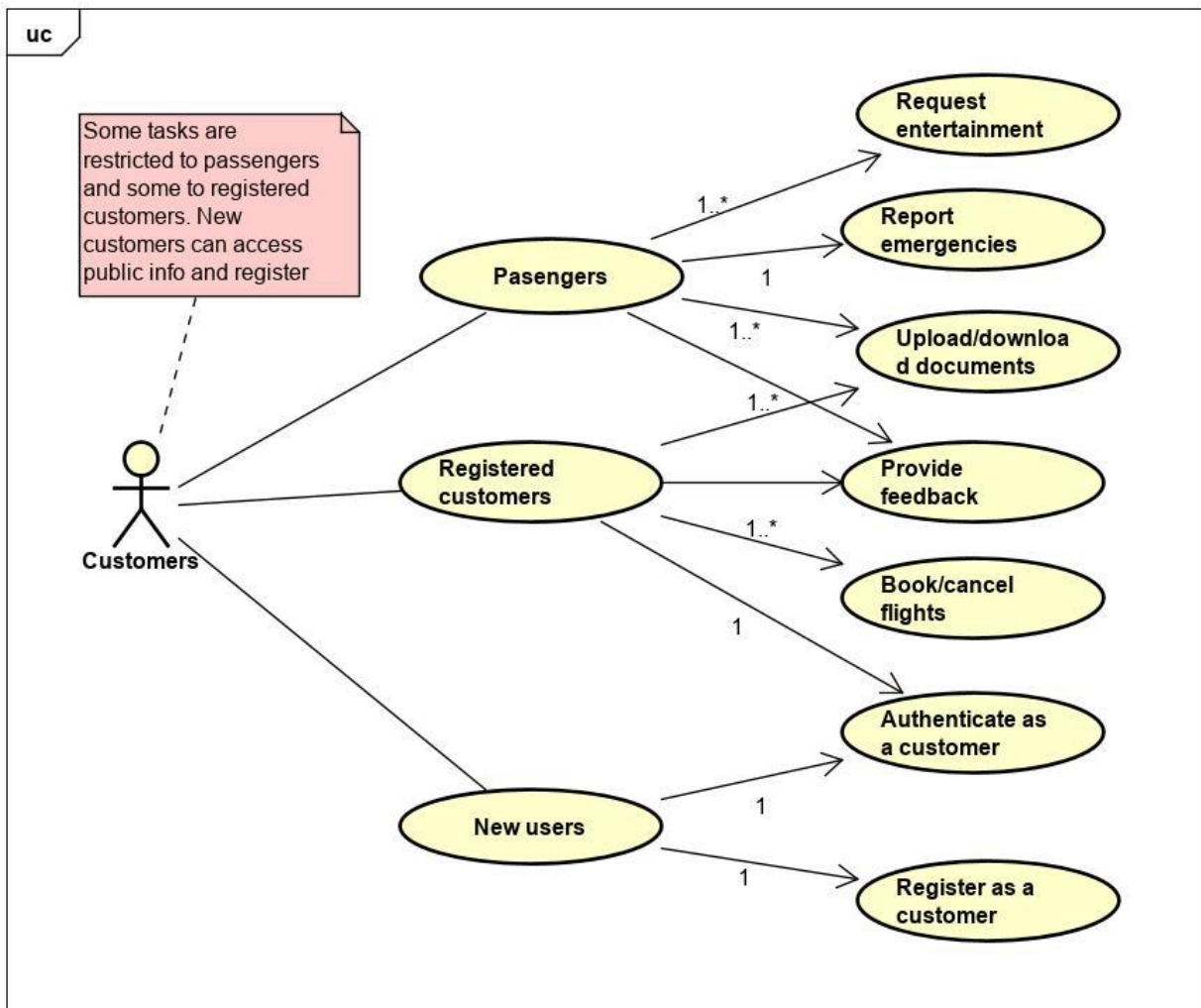
Administrators

Administrators can perform oversight, monitoring, analysis and notifications.



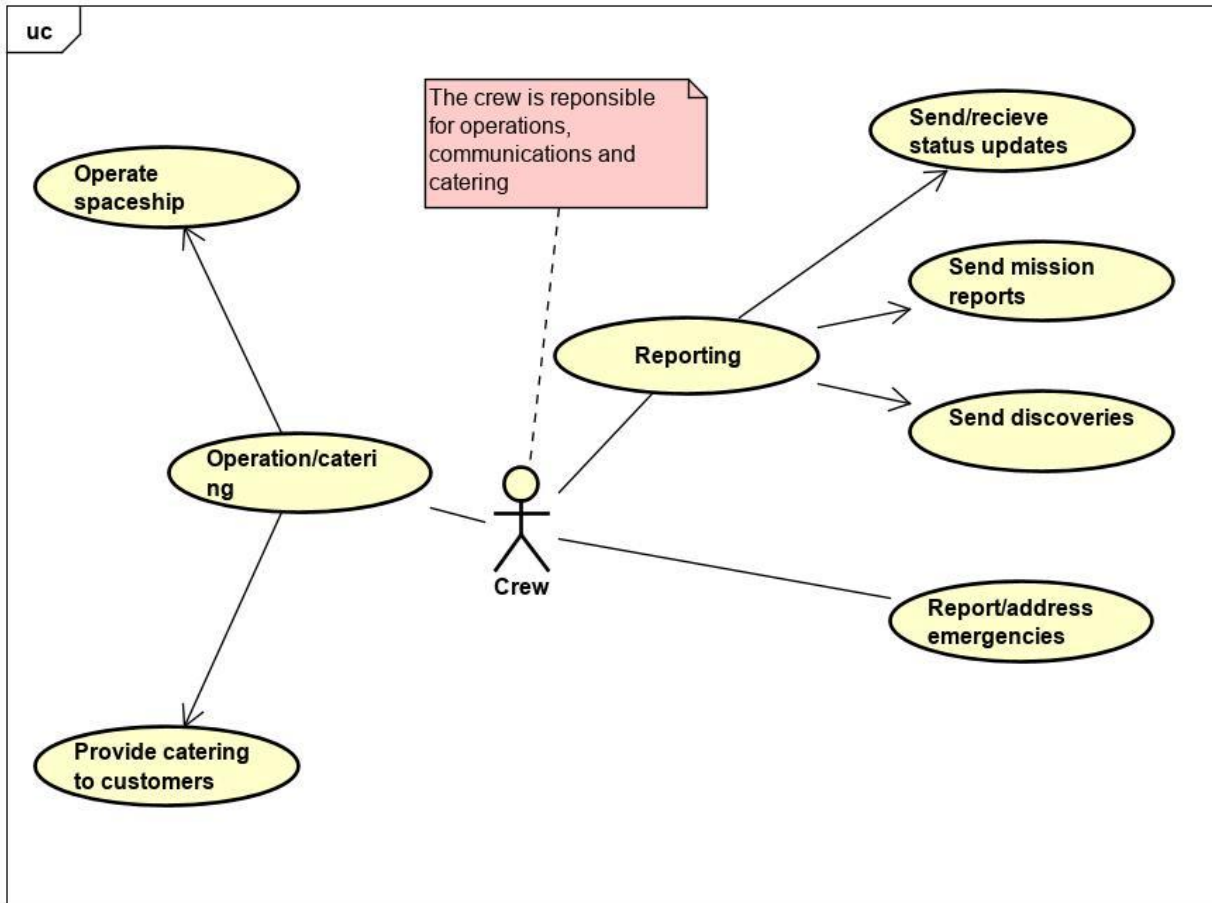
Customers

Customers may be registered users, currently in-flight passengers or new/anonymous users. They can perform the actions below.



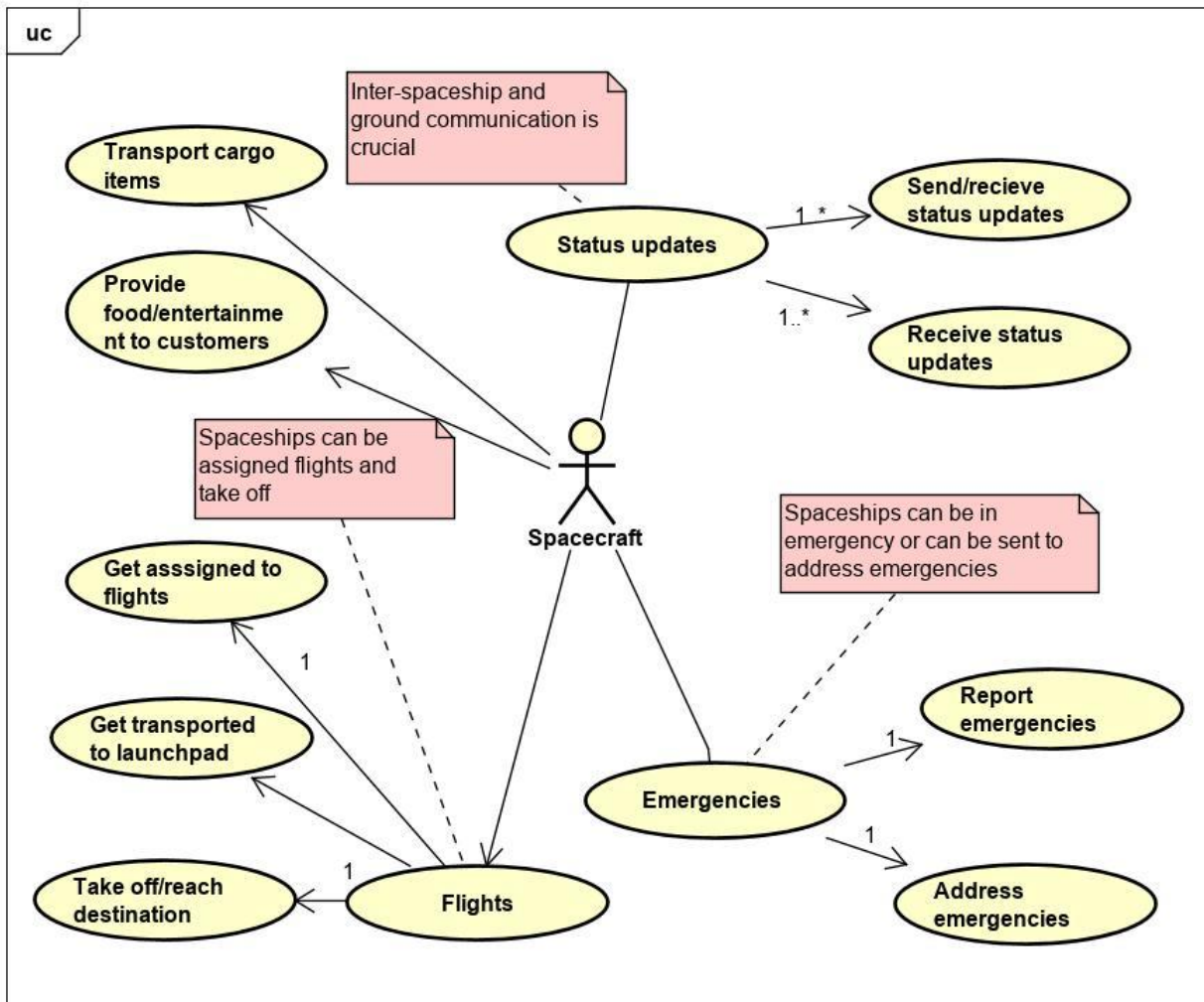
Crew

The flight crew is one of the actors in ISTS. They can perform operations, catering and reporting.



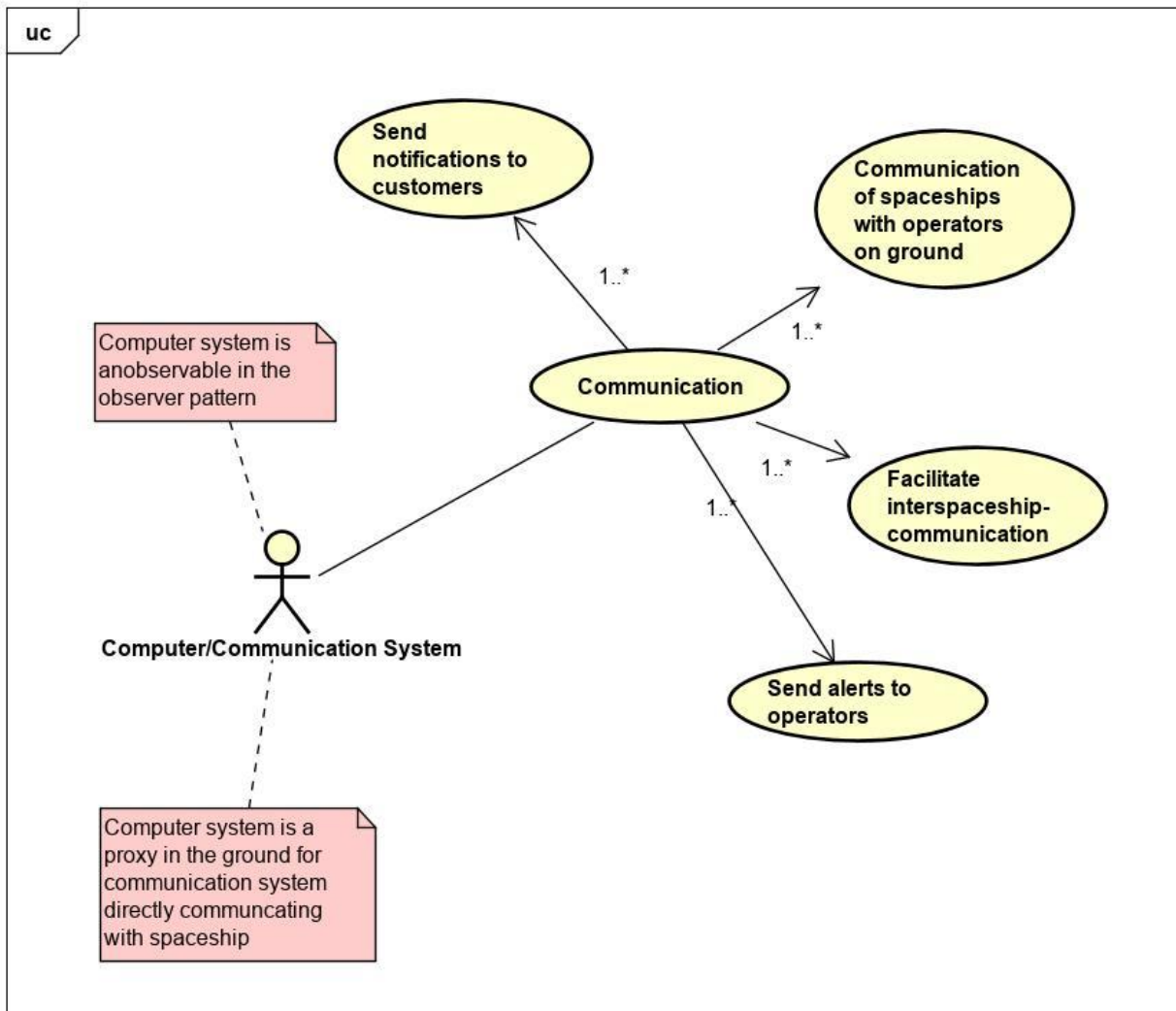
Spacecraft

Spacecrafts are one of the actors in ISTS and can affect the system by being the subject of flights, status updates and emergencies.



Computer/Communication System

The computer and communication systems send alert and health check updates. Therefore, they are one of the actors in ISTS



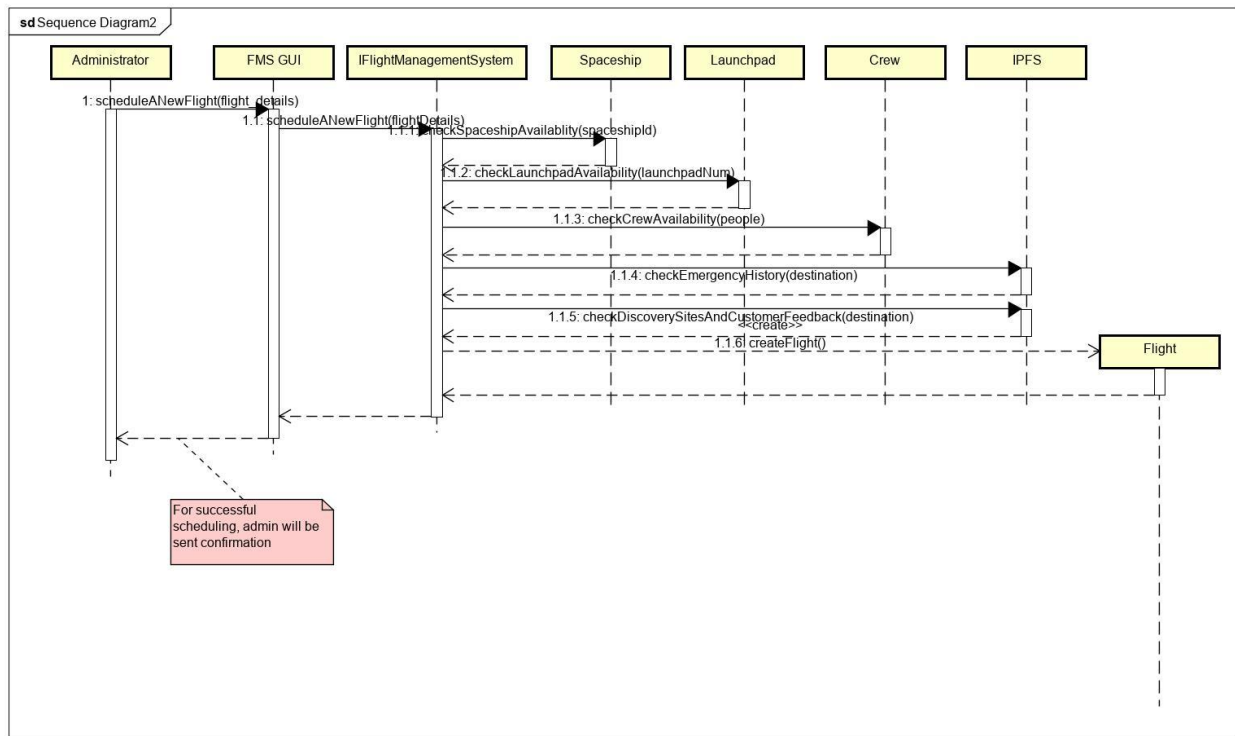
Implementation

The design heavily uses the following patterns

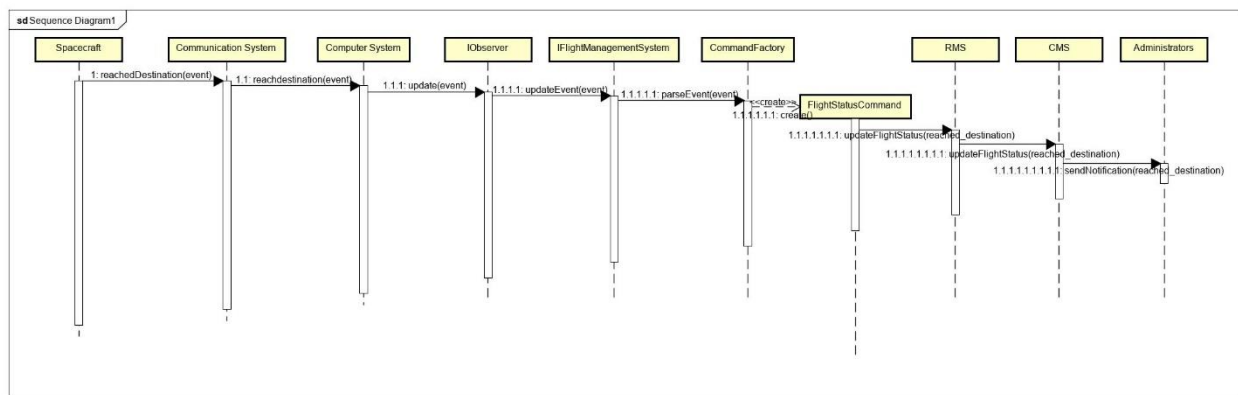
- 1) Command Pattern
- 2) Factory Pattern
- 3) Composite Pattern
- 4) Observer Pattern
- 5) Façade Pattern
- 6) Proxy Pattern
- 7) Singleton Pattern

The modules are independently deployable RESTful webservices achieving level 5 of modularity. The design also follows the underlying SOLID principles. It favors composition over inheritance and interfaces over implementation. Additional details that might obscure the functionality of the system are deliberately omitted while all the business and nonfunctional requirements are addressed. Please refer the class dictionary section of all the modules included here to get a better understanding of the class hierarchy, behavior and implementation details.

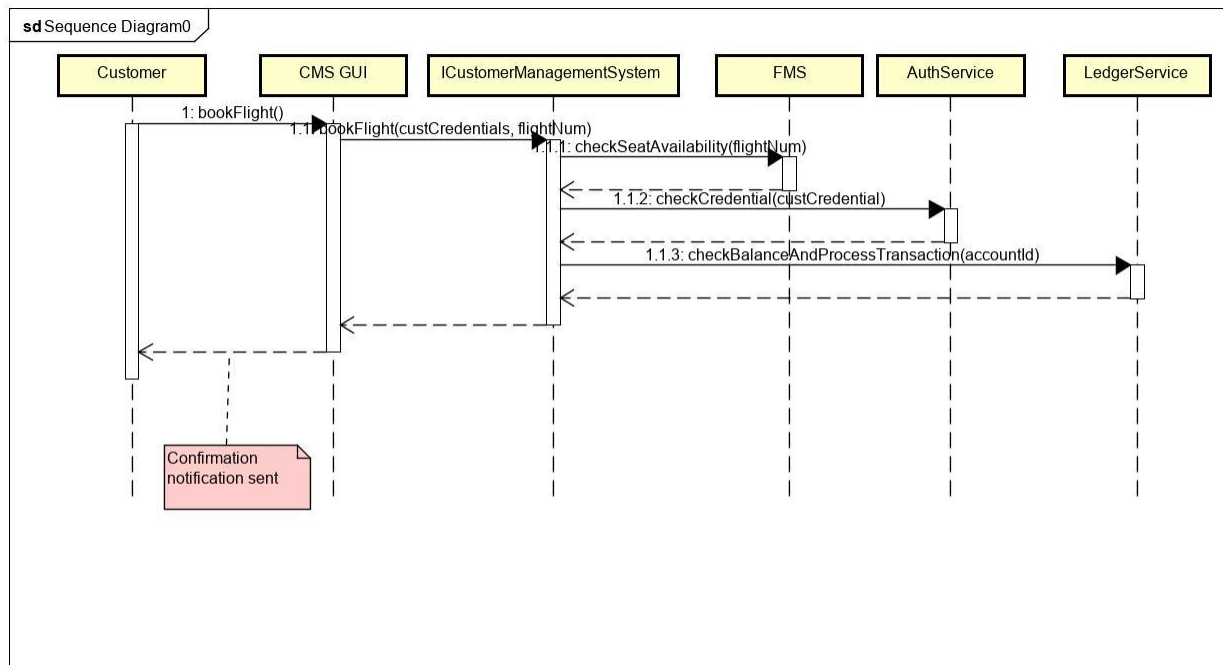
Sequence diagram for provisioning a new flight



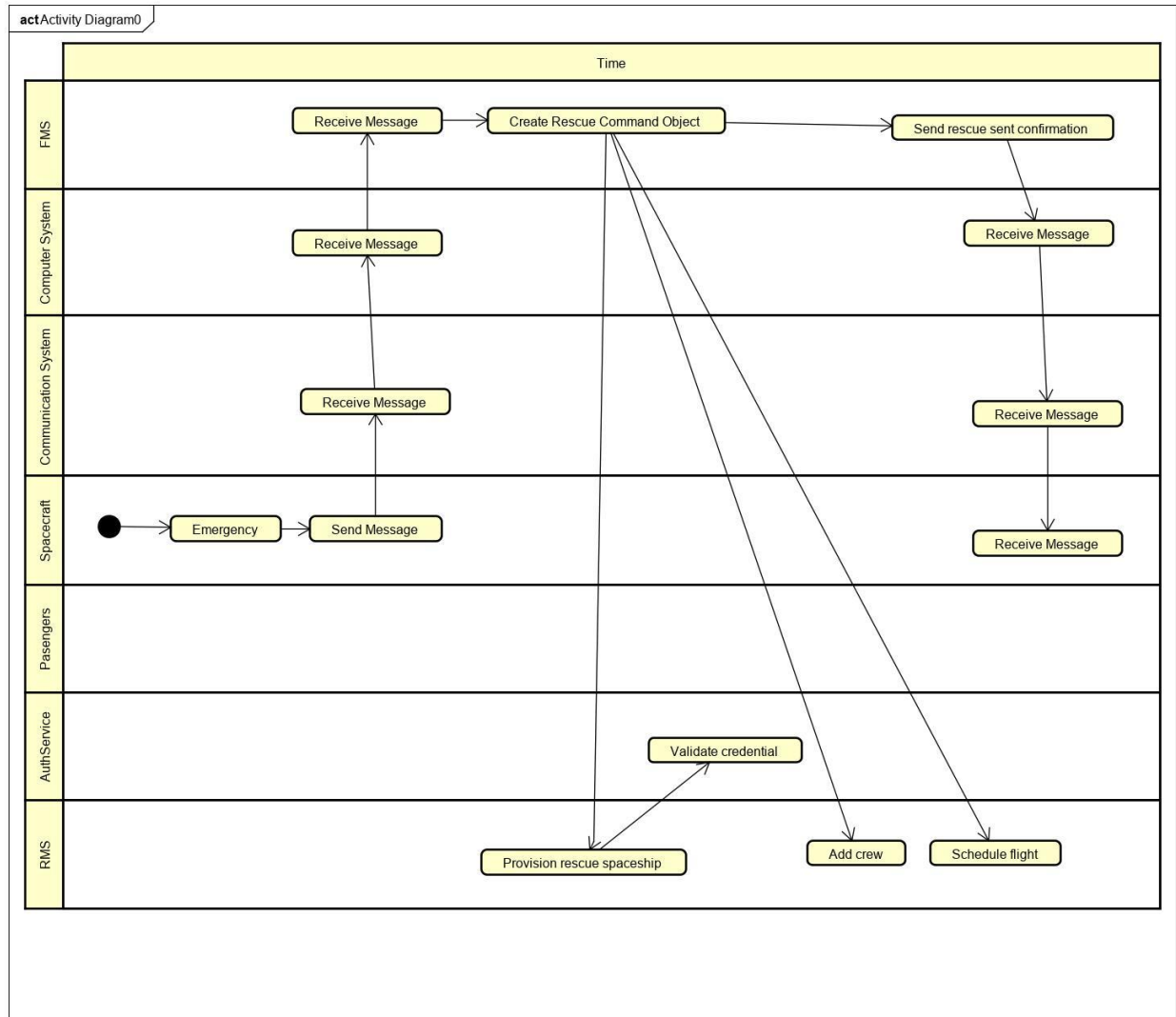
Sequence diagram when a spacecraft has reached destination



Sequence diagram when customer is booking a new flight



Activity diagram for distress signal and rescue mission



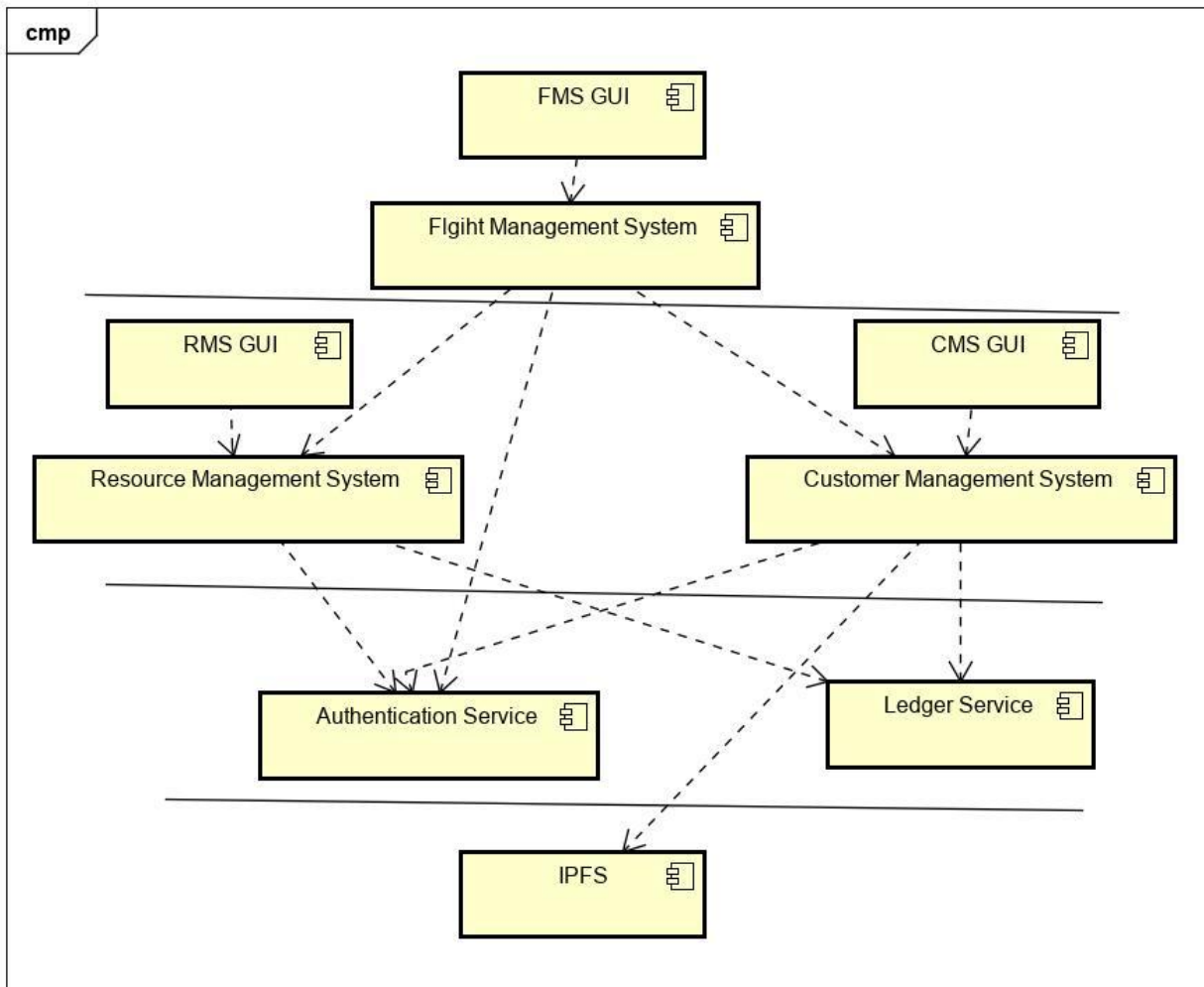
Resource Management System Design Document

Introduction

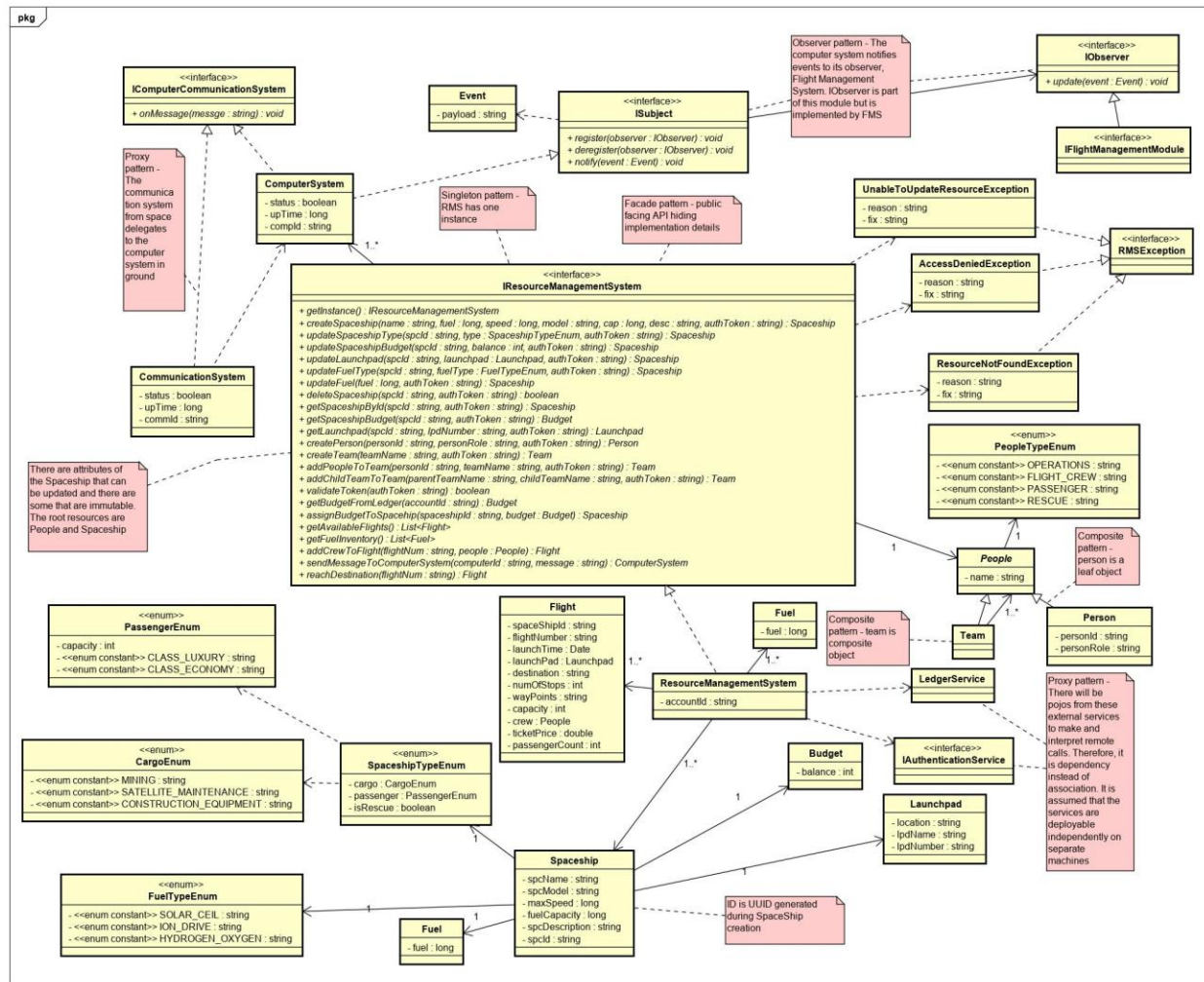
This section defines one of the modules that make up the Interplanetary Space Transport System, the Resource Management System. Resource Management System, RMS in short, keeps the inventory of resources that administrators can view and update. RMS depends on the Authentication Service and Ledger Service 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.



The following class diagram defines the classes defined in RMS module.



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

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
<code>getInstance</code>	<code>() : IResourceManagementSystem</code>	<code>/api/rms</code> POST	Singleton pattern – This method is invoked first to get a singleton instance of this service
<code>createSpaceship</code>	<code>(name: string, fuel: long, speed: long, model: string, cap: long, desc: string, authToken: string): Spaceship</code>	<code>/api/rms/spc/</code> POST	Defines a new Spaceship object
<code>updateSpaceshipType</code>	<code>(spcId : string, type : SpaceshipTypeEnum, authToken : string) : Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> PUT	Updates the spaceship type. The same spaceship can be repurposed for rescue, cargo or passenger use.
<code>updateSpaceshipBudget</code>	<code>(spcId: string, balance: int, authToken: string): Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> PUT	Administrators can allocate funds to a spaceship
<code>updateLaunchpad</code>	<code>(spcId: string, launchpad: Launchpad, authToken: string): Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> PUT	Administrators can adjust the Launchpad for a spaceship
<code>updateFuelType</code>	<code>spcId: string, fuelType: FuelTypeEnum, authToken: string): Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> PUT	Spaceships can be adjusted to be refueled by solar cell, ion drive or hydrogen/oxygen
<code>updateFuel</code>	<code>(fuel: long, authToken: string): Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> PUT	Administrators can fill up spacecrafts with fuel
<code>deleteSpaceship</code>	<code>(spcId: string, authToken: string): boolean</code>	<code>/api/rms/spc/{spcShipId}</code> DELETE	Administrators can decommission spaceships when they are out of use. Returns whether decommissioning is successful
<code>getSpaceshipById</code>	<code>(spcId: string, authToken: string): Spaceship</code>	<code>/api/rms/spc/{spcShipId}</code> GET	Gets spaceship by id
<code>getSpaceshipBudget</code>	<code>(spcId: string, authToken: string): Budget</code>	<code>/api/rms/spc/{spcShipId}/budget</code>	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

		GET	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/{teamName}/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/{teamName}/childTeam/{teamName} 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
assignBudgetToSpaceship	(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	Administrators can view the available fuel in each of the spacecrafts and in storage

		GET	
addCrewToFlight	(flightNum: string, people: People): Flight	/api/rms/flight/{flightNum}/people	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 cell, 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}/lpd/{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/{teamName}/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/{teamName}/childTeam/{teamName} 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
assignBudgetToSpaceship	(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 POST	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 is 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 cell, 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: IObservable): void	Registers observers
deregister	(observer: IObservable): 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: IObservable): void	Registers observers
deregister	(observer: IObservable): void	Deregister observers
notify	(event: Event): void	Notifies observers passing in payload

IObservable

The listener in **observer pattern**.

Methods

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

RMSEException

Marker interface all the exceptions thrown in this module implement

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

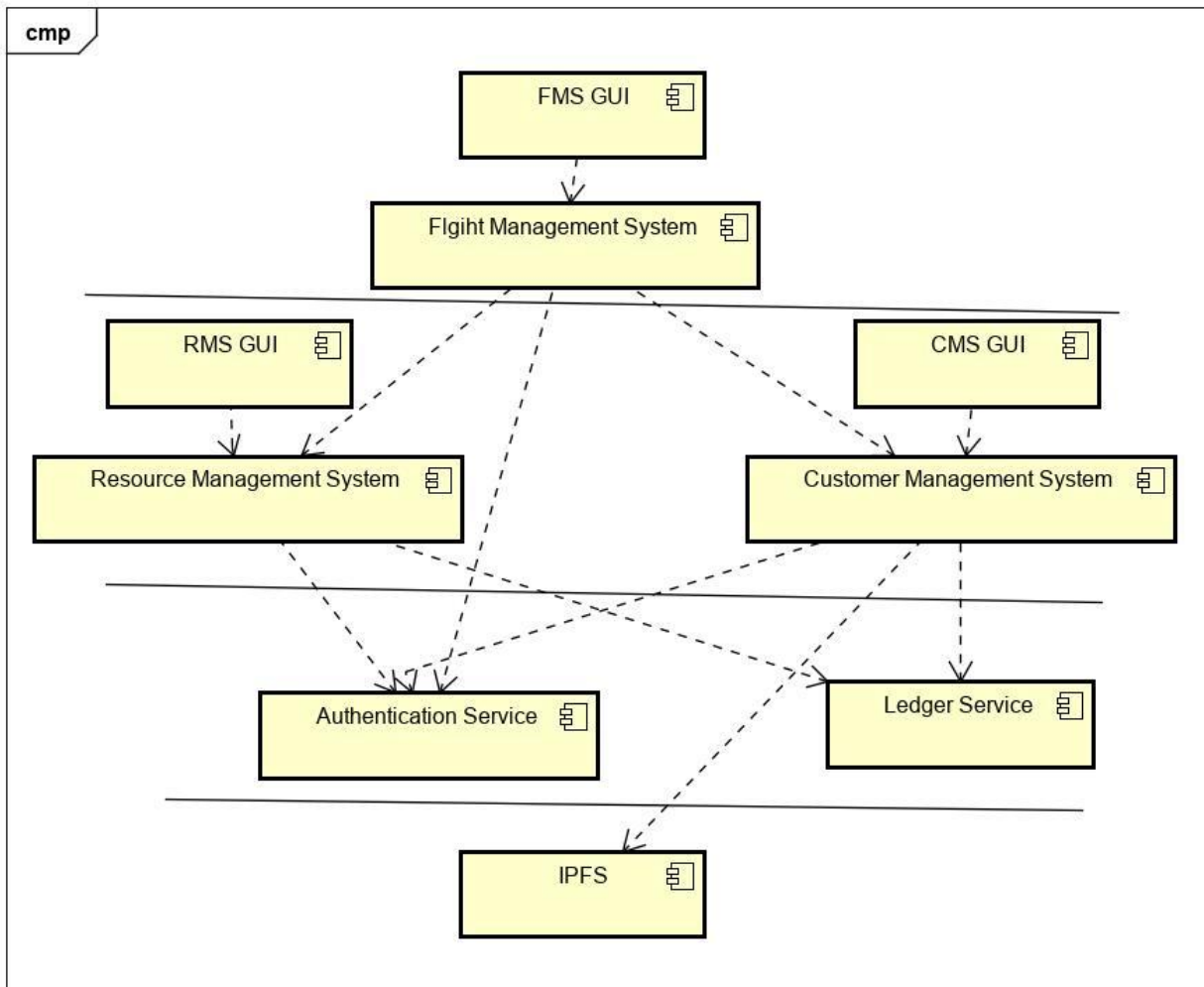
Customer Management System Design Document

Introduction

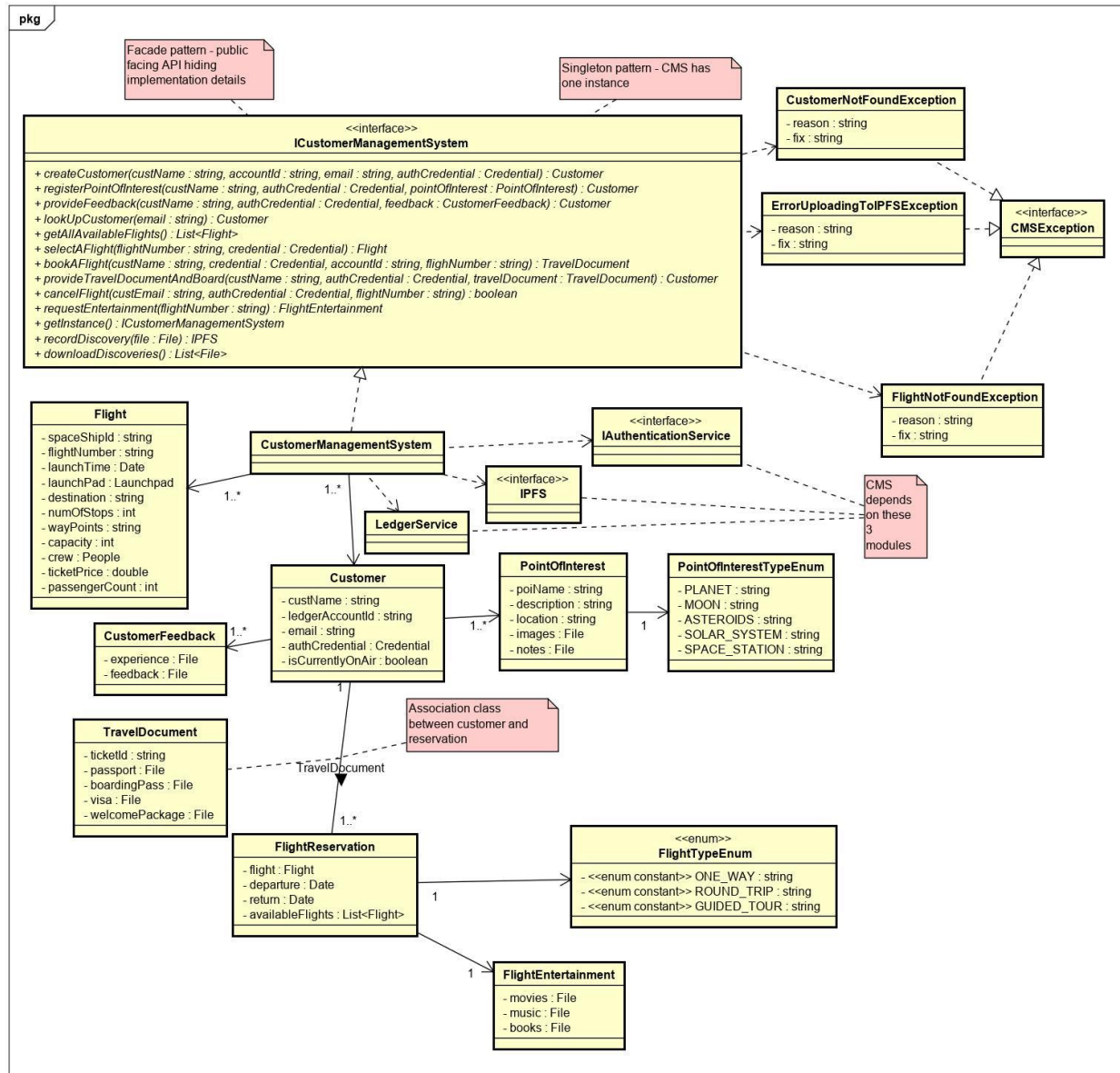
This module is one of the components in the Interplanetary Space Transport System. It is dependent on the Interplanetary File System for recreation and for the providing the ability for customers to dump any travel related or discovery related files. It depends on the Authentication Service to identify users and maintain sessions. It also depends on the Ledger Service to undertake transactions. The Customer Management System, CMS, in short has a user interface accessible to end users and administrators. Customers can perform tasks ranging from registering as a user to booking flights and recording their journeys using this module, the Customer Management System.

Overview

The Customer Management System besides other tasks is the money engine for the Interplanetary Space Transport System. The system must be able to finance itself in order to fund upcoming space explorations and the main source of income is by having passengers onboard. CMS handles the customer's interaction to booking flight including accepting payment methods besides providing a greater customer experience during flight. Customer Management System's interaction with IPFS makes documents instantly accessible even in the extraterrestrial world. CMS as discussed above is dependent on the Authentication Service, the Ledger Service and IPFS. See diagram below for clear understanding of the interaction.



The following class diagram defines the classes defined in CMS module.



Class Dictionary

This section is the class dictionary for classes to be defined under the package ‘com.cscie97.ists.cms’

ICustomerManagementSystem

This is a REST API that is accessible to other modules in the ISTS system and the CMS GUI. It defines the contract for interaction. Customers can pass in their credentials to log in, browse flights, select flights, book flights, record discoveries and request entertainment using service API. Credentials must be passed to the methods and CMS obtains an access token from the Authentication Service on behalf of the customer.

Methods

Method Name	Signature	URL	Description
getInstance	(): ICustomerManagementSystem	/api/cms GET	Singleton pattern – This gets a singleton instance of the service API
createCustomer	(custName: string, accountId: string, email: string, authCredential: Credential): Customer	/api/cms/customer/ POST	Creates a new customer. The customer must have opened a new account in the Authentication Service and obtained credentials already
registerPointOfInterest	(custName: string, authCredential: Credential, pointOfInterest: PointOfInterest): Customer	/api/cms/customer/{custId}/poi/ POST	Registered customers can express interest to explore points of interest in the space
provideFeedback	(custName: string, authCredential: Credential, feedback: CustomerFeedback): Customer	/api/cms/customer/{custId}/feedback/ POST	Registered customers can provide feedback about their flight experience and space exploration
lookUpCustomer	(email: string): Customer	/api/cms/customer/{custEmail} GET	Customers can be looked up using their email
getAllAvailableFlights	(): List<Flight>	/api/cms/flights GET	Returns upcoming flights. When Flight Management System schedules a new flight, it is dumped to CMS for storage

selectAFlight	(flightNumber: string, credential: Credential): Flight	/api/cms/flights/{flightNum} GET	Customers can choose a flight to book
bookAFlight	(custName: string, credential: Credential, accountId: string, flightNumber: string): TravelDocument	/api/cms/flights/{flightNum} POST	Customers provide their name, credentials, blockchain account ids and the flight they are interested in. CMS authenticates the customer using the Authentication Service and processes transactions using the Ledger service. If the customer's passport and visa is found in IPFS, the customer in return gets issued a travel document with welcome package and boarding pass in return from IPFS. The travel document is an association class between customer and flight reservation which contains 1 or more details of flight reservation. Flight reservations can be one way, round trip or guided trips.
provideTravelDocumentAndBoard	(custEmail: string, authCredential: Credential, travelDocument: TravelDocument): Customer	/api/cms/customer/{custEmail} POST	When the time arrives for boarding the customer provides email and other credentials with the travel documents like a boarding pass previously issued. The customer will be marked as currentlyOnAir upon successful boarding. When the flight concludes, this flag will be set to false.
cancelFlight	(custEmail: string, authCredential: Credential, flightNumber: string): boolean	/api/cms/customer/{custEmail}/cancel DELETE	Customers can cancel their flight before boarding time free of charge. Customers will be charged a fee if they cancel after boarding time and standby customers will be allowed to board.
requestEntertainment	(flightNumber: string): FlightEntertainment	/api/cms/flights/{flightNum}/customer/{custEmail}	Customers currently on air can request entertainment. CMS will query IPFS to get the list of movies, music and books

		GET	
recordDiscovery	(file: File): IPFS	/api/cms/discovery POST	Any customer or part of the crew can post discoveries. CMS sends the files to IPFS. They can also post anonymously. It returns a pointer to IPFS and is downloadable
downloadDiscoveries	() : List<File>	/api/cms/discovery GET	Customers can download all the recent discoveries. CMS queries IPFS to retrieve the files

CustomerManagementSystem

This is an implementation of the ICustomerManagementSystem. It also has associations to IAuthenticationService, LedgerService and IPFS.

Methods

Method Name	Signature	URL	Description
getInstance	() : ICustomerManagementSystem	/api/cms GET	Singleton pattern – This gets a singleton instance of the service API
createCustomer	(custName: string, accountId: string, email: string, authCredential: Credential): Customer	/api/cms/customer/ POST	Creates a new customer. The customer must have opened a new account in the Authentication Service and obtained credentials already
registerPointOfInterest	(custName: string, authCredential: Credential, pointOfInterest: PointOfInterest): Customer	/api/cms/customer/{custId}/poi/ POST	Registered customers can express interest to explore points of interest in the space
provideFeedback	(custName: string, authCredential: Credential, feedback: CustomerFeedback): Customer	/api/cms/customer/{custId}/feedback/ POST	Registered customers can provide feedback about their flight experience and space exploration
lookUpCustomer	(email: string): Customer	/api/cms/customer/{c	Customers can be looked up using their email

		ustEmail} GET	
getAllAvailableFlights	() : List<Flight>	/api/cms/flights GET	Returns upcoming flights. When Flight Management System schedules a new flight, it is dumped to CMS for storage
selectAFlight	(flightNumber: string, credential: Credential): Flight	/api/cms/flights/{flightNum} GET	Customers can choose a flight to book
bookAFlight	(custName: string, credential: Credential, accountId: string, flightNumber: string): TravelDocument	/api/cms/flights/{flightNum} POST	Customers provide their name, credentials, blockchain account ids and the flight they are interested in. CMS authenticates the customer using the Authentication Service and processes transactions using the Ledger service. If the customer's passport and visa is found in IPFS, the customer in return gets issued a travel document with welcome package and boarding pass in return from IPFS. The travel document is an association class between customer and flight reservation which contains 1 or more details of flight reservation. Flight reservations can be one way, round trip or guided trips.
provideTravelDocumentAndBoard	(custEmail: string, authCredential: Credential, travelDocument: TravelDocument): Customer	/api/cms/customer/{custEmail} POST	When the time arrives for boarding the customer provides email and other credentials with the travel documents like a boarding pass previously issued. The customer will be marked as currentlyOnAir upon successful boarding. When the flight concludes, this flag will be set to false.
cancelFlight	(custEmail: string, authCredential: Credential, flightNumber: string): boolean	/api/cms/customer/{custEmail}/cancel	Customers can cancel their flight before boarding time free of charge. Customers will be charged a fee if they cancel after

		DELETE	boarding time and standby customers will be allowed to board.
requestEntertainment	(flightNumber: string): FlightEntertainment	/api/cms/flights/{flightNum}/customer/{custEmail} GET	Customers currently on air can request entertainment. CMS will query IPFS to get the list of movies, music and books
recordDiscovery	(file: File): IPFS	/api/cms/discovery POST	Any customer or part of the crew can post discoveries. CMS sends the files to IPFS. They can also post anonymously. It returns a pointer to IPFS and is downloadable
downloadDiscoveries	() : List<File>	/api/cms/discovery GET	Customers can download all the recent discoveries. CMS queries IPFS to retrieve the files

Properties

Property Name	Type	Description
customers	List<Customer>	List of registered customers
availableFlights	List<Flight>	List of upcoming flights. Once spaceship reaches destination, flights are taken of this list

Associations

Association Name	Type	Description
authenticationService	IAuthenticationService	Used to obtain tokens on behalf of customers and maintain sessions
ledgerService	LedgerService	Used to book flights and process transactions on behalf of a customer
iPFS	IPFS	Downloadable discoveries can be posted anonymously by customers, passports and visas can be uploaded, travel documents can be issued, entertainment can be streamed through IPFS

Flight

This object is instantiated when a new flight is scheduled. The Flight Management System dumps new flights to CMS 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

CUSTOMER

A customer who registered in the Authentication Service first can register as a customer in CMS. It is associated with the customer's preferences and feedback to provide a greater customer experience. The customer must also setup an account in the ledger service to complete his/her profile and process payments.

Properties

Property Name	Type	Description
custName	string	Name of the customer
ledgerAccountId	string	Account id of the ledger service to process payments
custEmail	string	Customers can associate an email address to their account to receive notifications
credential	Credential	Credentials can be voiceprints, faceprints or username/password
isCurrentlyOnAir	boolean	Identifies whether the customer is in a current flight or on the ground

Associations

Association Name	Type	Description
feedbackList	List<CustomerFeedback>	The customer can have a history of feedback
pointOfInterestList	List<PointOfInterest>	The customer can have a list of points of interest.
travelDocument	TravelDocument	A customer can have a travel document. The travel document can be issued for multiple flight reservations

CustomerFeedback

Customers can submit feedback anonymously or as an identified customer. CMS sends the feedback files to IPFS.

Properties

Property Name	Type	Description
---------------	------	-------------

experience	File	Customers can document their experiences
feedback	File	Customers can document their feedback

PointOfInterest

Customers can register interest on what to explore. Customers can add as many points of interest as they wish

Properties

Property Name	Type	Description
poiName	string	Name of point of interest
poiDescription	string	Description of point of interest
location	string	Location of point of interest
images	File	Customers can have image files of what they want to explore. The files are stored in IPFS
notes	File	Customers can have notes of what they want to explore. The files are stored in IPFS

Associations

Association Name	Type	Description
poiType	PointOfInterestTypeEnum	Point of interest can be planet, moon, asteroids...

PointOfInterestTypeEnum

This defines types of points of interest

Properties

Property Name	Type	Description
PLANET	string	Type of point of interest
MOON	string	Type of point of interest
ASTEROIDS	string	Type of point of interest
SOLAR_SYSTE	string	Type of point of interest

M		
SPACESTATION	String	Type of point of interest

TravelDocument

Travel documents are issued to customers upon successful booking of a flight. The customer needs to provide passports and visas to be uploaded to IPFS from CMS. IPFS in return issues ticket id, boarding pass and welcoming package in return. TravelDocument is an **association class** between Customer and FlightReservation.

Properties

Property Name	Type	Description
ticketId	string	Unique UUID issued by IPFS upon successful booking of a flight
passport	File	Must be provided by customer. CMS sends it to IPFS for uploading
boardingPass	File	Issued by IPFS upon successful booking of a flight
visa	File	Must be provided by customer. CMS sends it to IPFS for uploading
welcomePackage	File	Issued by IPFS upon successful booking of a flight

Associations

Association Name	Type	Description
reservations	List<FlightReservation>	1 or more reservations associated with the travel document

FlightReservation

Since the customer selects the flightNumber when selecting a flight some of the details will be pulled down from there. However, the customer may pick a different return flight or a different departure date and those specifications will be defined here.

Properties

Property Name	Type	Description
---------------	------	-------------

flight	Flight	The flight selected by the customer
departure	Date	Departure selected by the customer
return	Date	Return date selected by the customer
availableFlights	List<Flight>	A customer can alternate between the available flights free of charge

Associations

Association Name	Type	Description
flightType	FlightTypeEnum	Flights can be one-way, round trip or guided tour
entertainment	FlightEntertainment	Entertainment streamed through IPFS

FlightTypeEnum

This defines types of flights

Properties

Property Name	Type	Description
ONEWAY	string	Passengers can pick one-way flight
ROUND_TRIP	string	Passengers can pick a round trip flight
GUIDED_TOUR	string	Passengers can pick guided tours

FlightEntertainment

Customers on air can request entertainment through CMS portal and IPFS will deliver content

Properties

Property Name	Type	Description
movies	File	Can be streamed through IPFS
books	File	Can be streamed through IPFS
music	File	Can be streamed through IPFS

CMSException

Marker interface all of the exceptions thrown in this module implement

CustomerNotFoundException

This is thrown when a customer is not registered yet

Properties

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

ErrorUploadingToIPFSException

This is thrown when there is connectivity error with IPFS

Properties

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

FlightNotFoundException

This is thrown when flight is not available due to getting cancelled for technical issues or if the boarding time has passed or if the flight was never scheduled by the Flight Management System.

Properties

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

Flight Management System Design Document

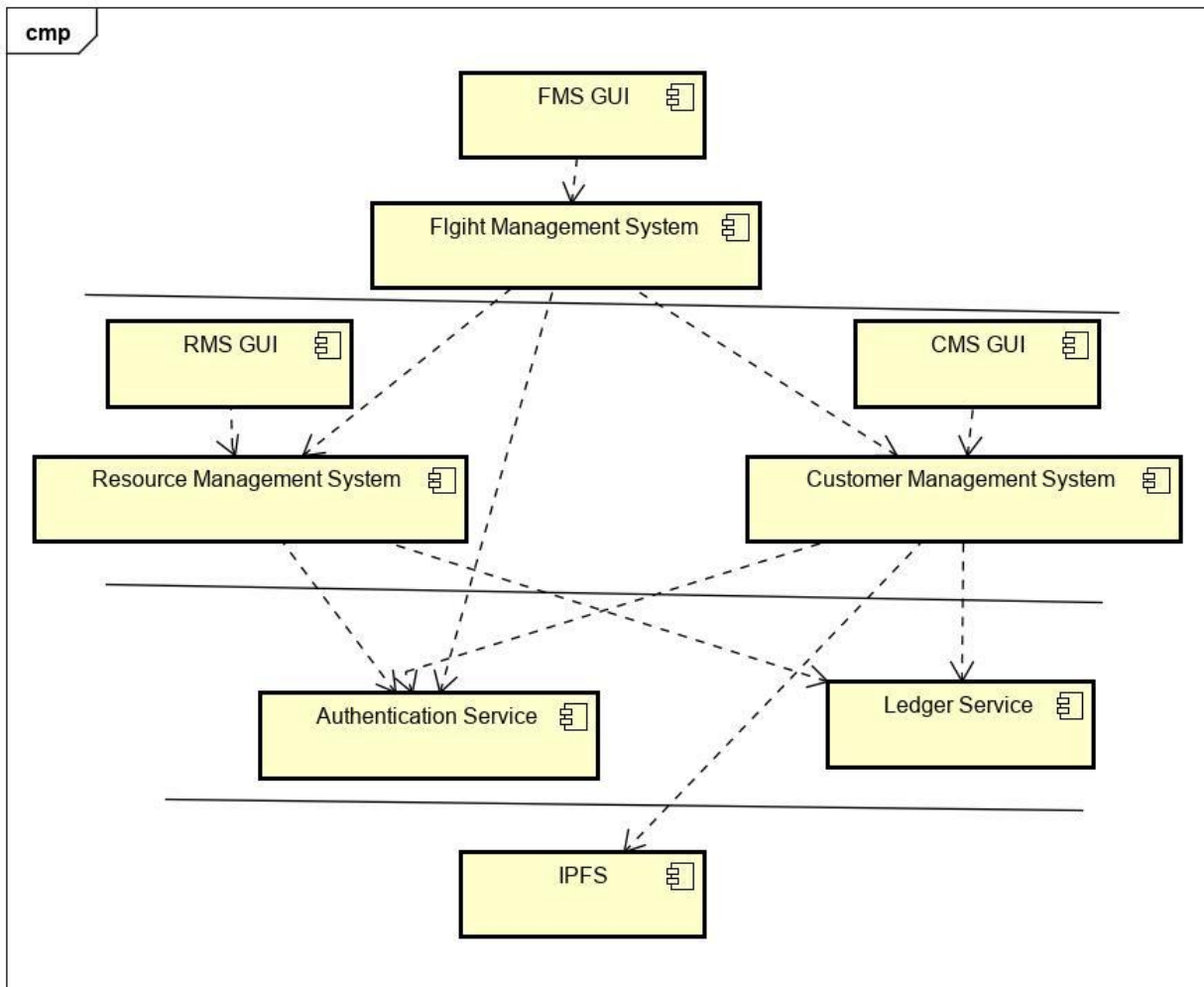
Introduction

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

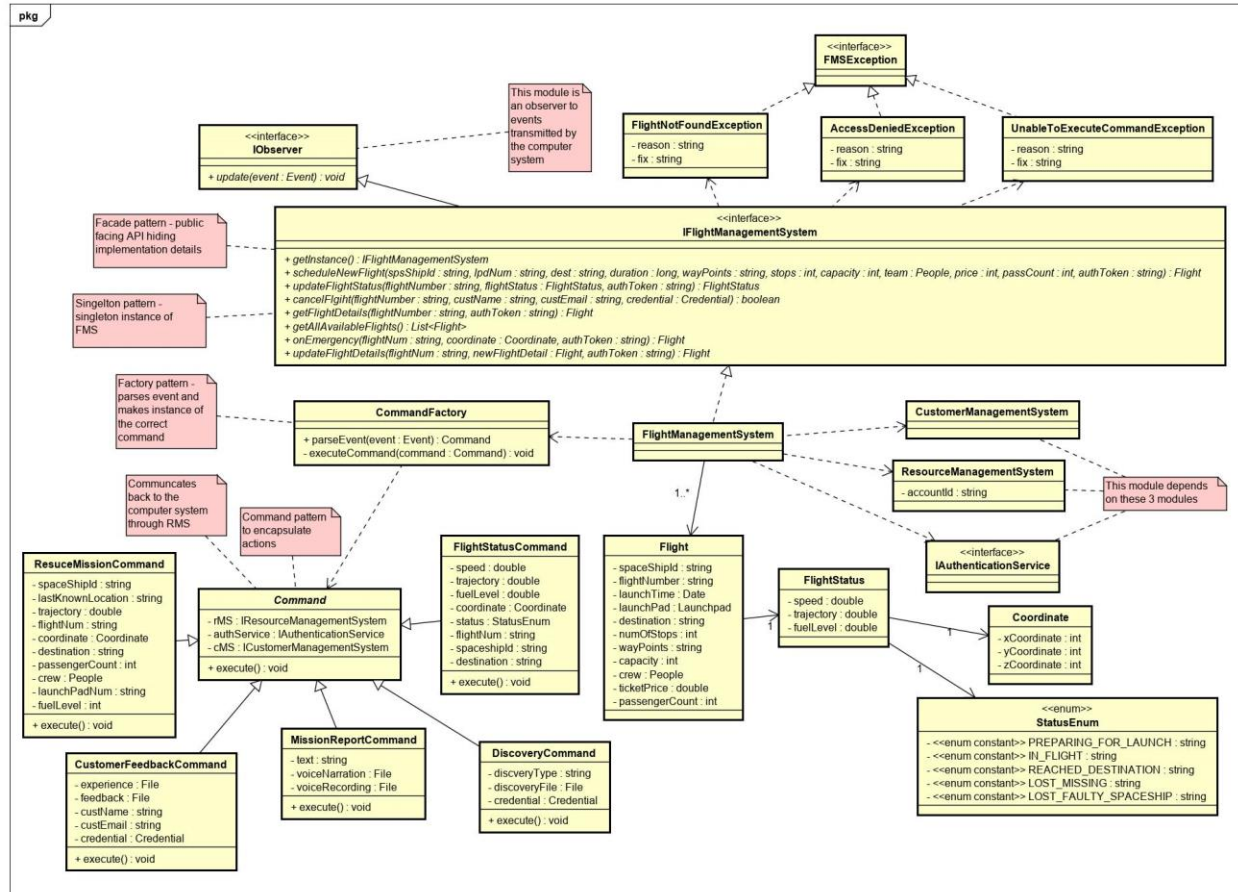
The goal of ISTS is to fly to space and everything else that comes along with it. The job of this module is to monitor flights, listen to notifications, serve as a medium to inter-spaceship and ground communications, coordinate with other modules to store reports/discoveries and orchestrate a rescue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

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



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 **Facade 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 .
cancelFlight	(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 .
cancelFlight	(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: string)	This enables admins or customers to manually

	Coordinate, authToken: string): Flight	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_FO RLAUNCH	string	Preparing for launch
INFLIGHT	string	In flight
REACHEDDESTI NATION	string	Reached destination
LOSTMISSING	string	Missing
LOSTFAULTYSP ACESHIP	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

UnableToExecuteCommandException

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

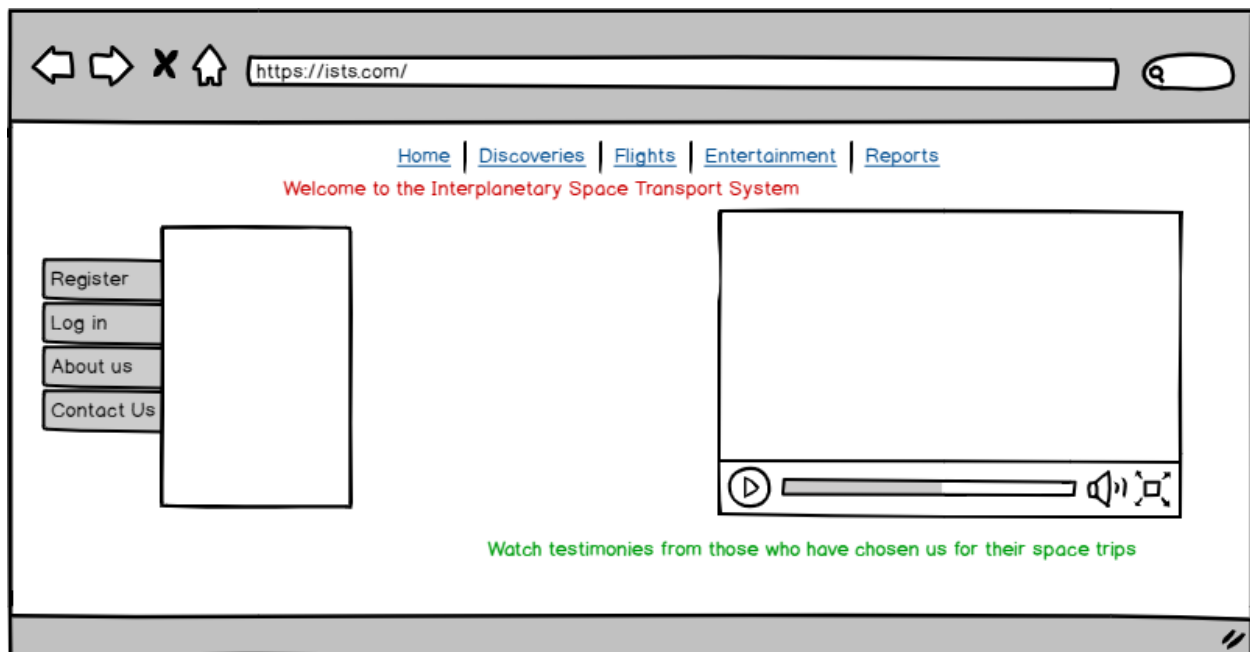
Properties

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

Graphical User Interface

The system provides a minimalist and user-friendly interface for admins, customers, the staff and the public. It is a Single Page Application despite calling multiple services to accomplish the customer journey. Please pay attention to the URLs used to navigate between pages. The user interface exercises the defined service APIs without much need for data manipulation.

Landing page



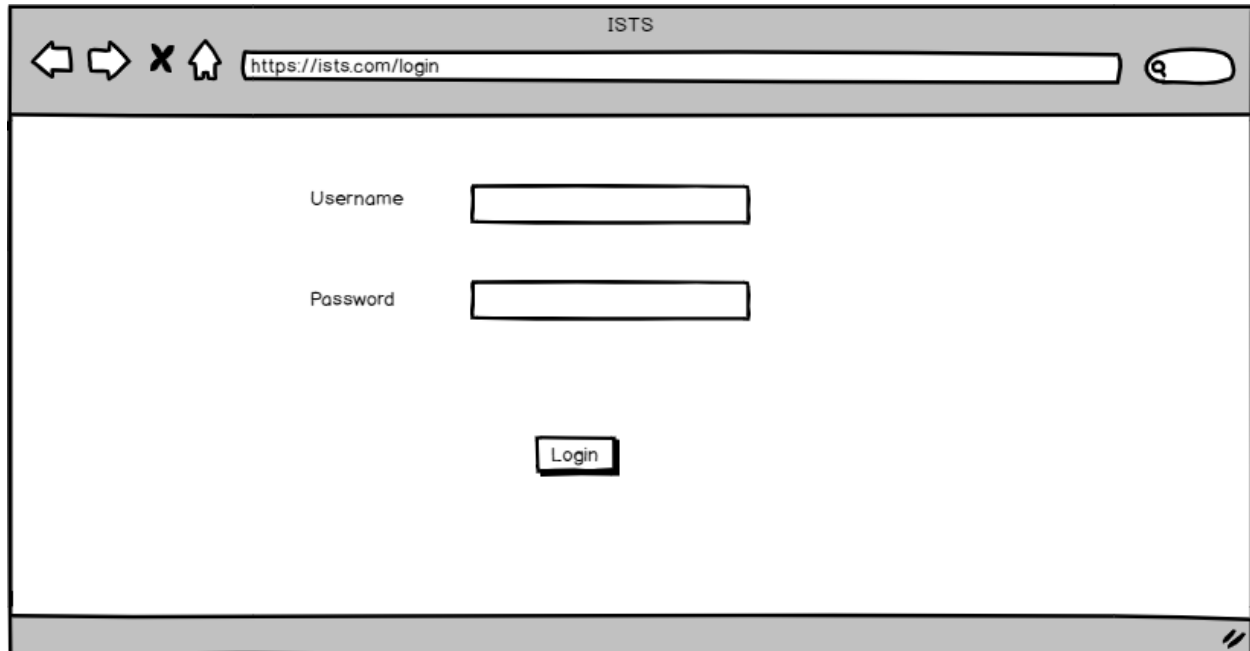
To register as a user, calls CMS and AuthService API

The diagram illustrates a web browser window for the 'ISTS' website. The browser's address bar shows the URL `https://ists.com/register`. The registration form is structured as follows:

Fullname	<input type="text"/>	Date of birth	<input type="text"/>
Email	<input type="text"/>	How did you hear about us?	<input type="text"/>
Username	<input type="text"/>	Password	<input type="text"/>

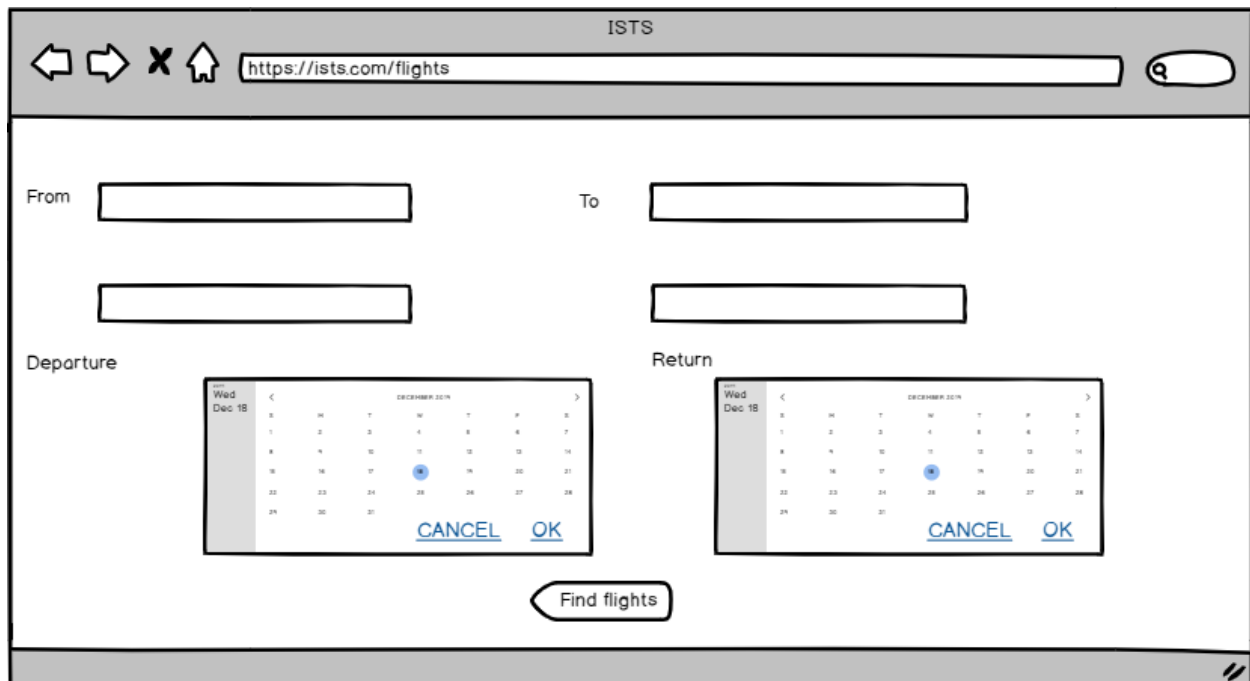
Below the form fields is a **Register** button.

To log in as a user, calls AuthService API



The screenshot shows a web browser window titled "ISTS". The address bar contains the URL "https://ists.com/login". The page has a simple login form with two input fields: "Username" and "Password". Below these fields is a "Login" button. The browser window includes standard navigation icons (back, forward, close, home) and a search icon in the address bar.

To search for flights, calls RMS API

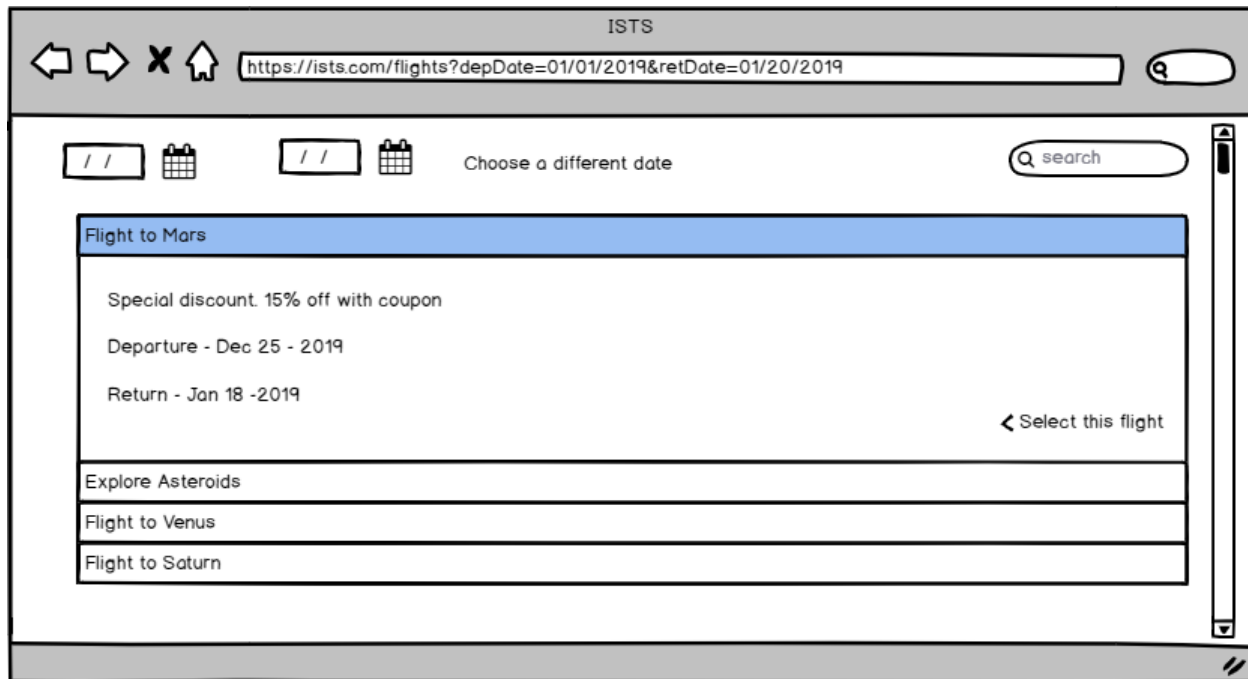


The screenshot shows a web browser window titled "ISTS". The address bar contains the URL "https://ists.com/flights". The page features a flight search form with the following elements:

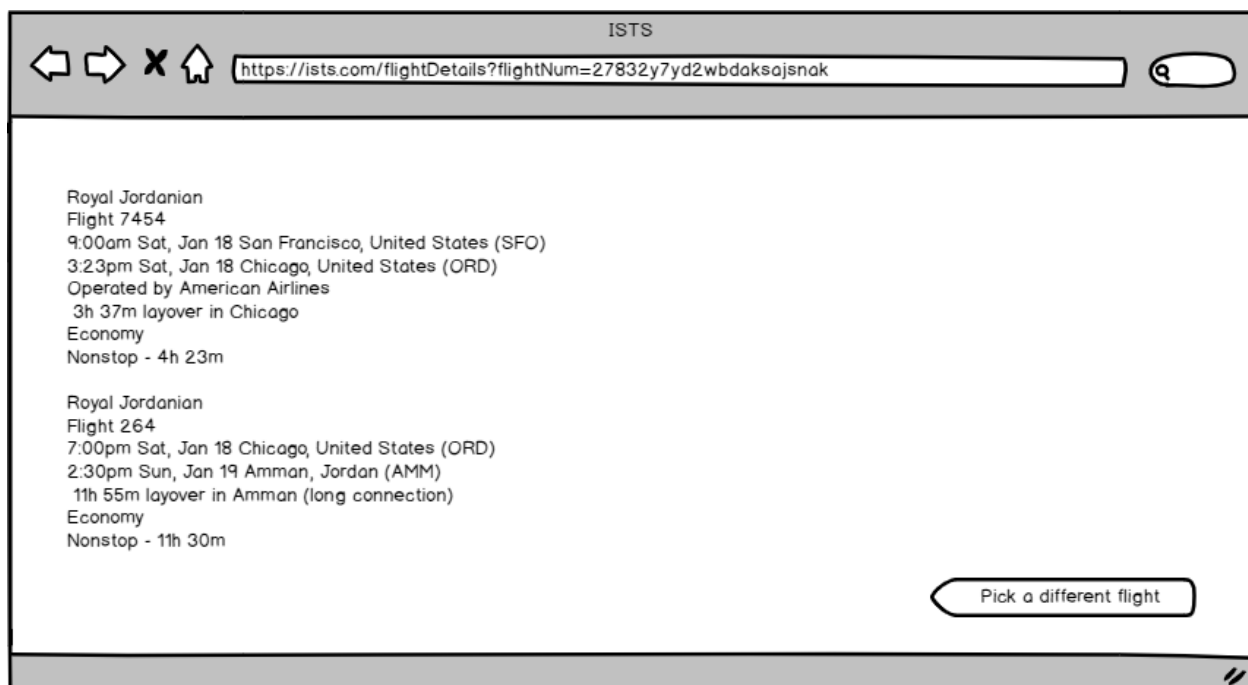
- From:** A text input field.
- To:** A text input field.
- Departure:** A date selection interface showing a calendar for December 2019. The date "Wed Dec 18" is selected. Below the calendar are "CANCEL" and "OK" buttons.
- Return:** A date selection interface showing a calendar for December 2019. The date "Wed Dec 18" is selected. Below the calendar are "CANCEL" and "OK" buttons.
- Find flights:** A button located below the date selection interfaces.

The browser window includes standard navigation icons (back, forward, close, home) and a search icon in the address bar.

List of flights found for the dates provided, calls FMS API



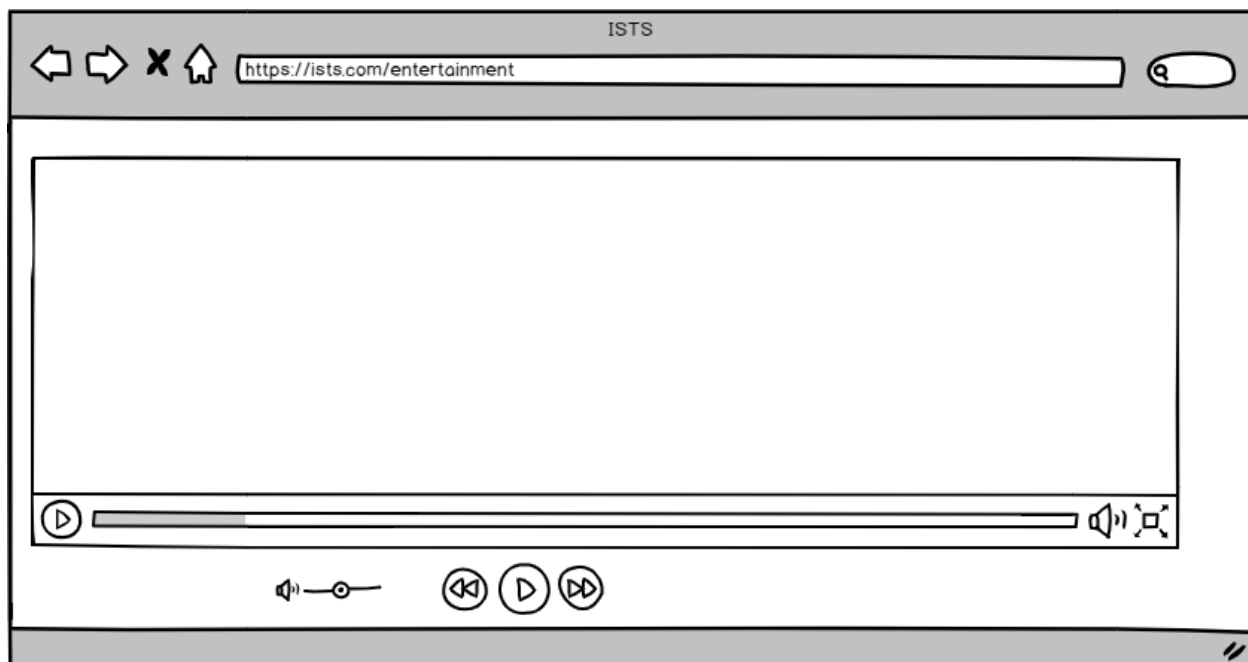
Selecting a flight and booking, calls FMS API and Ledger Service



Looking at current flight details and metrics, retrieves it from FMS API



Watching entertainment during flight, calls IPFS through CMS



To download/upload reports, discoveries and points of interest, calls CMS and IPFS

The screenshot shows a web browser window titled "ISTS" with the address bar displaying "https://ists.com/report". The main content area is titled "Upload reports or discoveries". It contains a "Category" label followed by a text input field. Below this is a "File" label, a "Browse" button, and a circular button with a plus sign. At the bottom of the form is an "Upload" button. The browser window has standard navigation icons (back, forward, stop, home) and a search icon in the top right corner.

As an admin, to define new resources, calls RMS and IPFS through CMS

The screenshot shows a web browser window titled "ISTS" with the address bar displaying "https://ists.com/admin/resources". The main content area is titled "Define resources". It contains a circular button with a plus sign. Below this is a "Quantity" label, a text input field containing the number "3", and a small up/down arrow icon. At the bottom of the form is an "Add" button. The browser window has standard navigation icons (back, forward, stop, home) and a search icon in the top right corner.

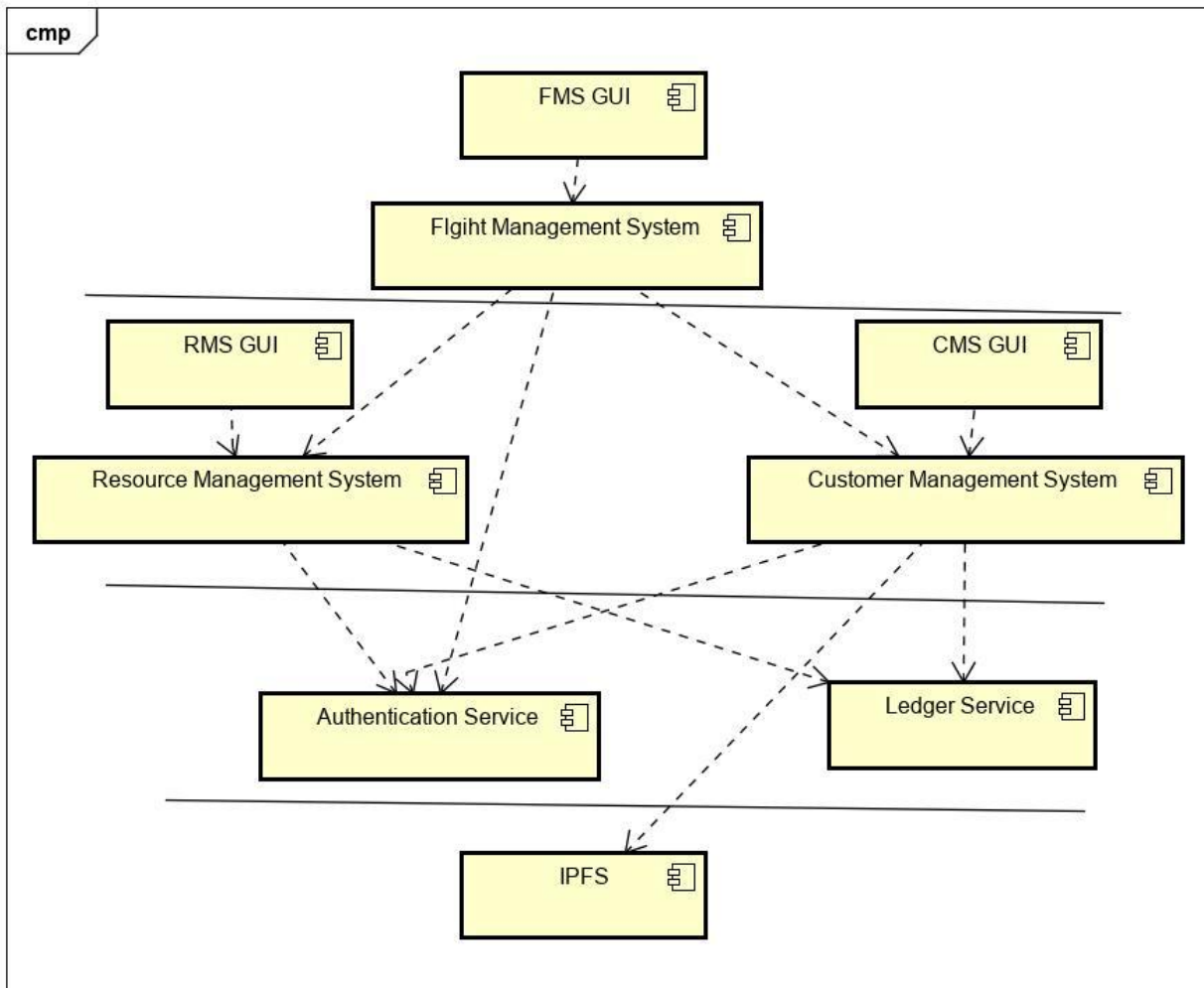
Implementation Details

The modules are independently deployable RESTful webservices achieving level 5 of modularity. The design also follows the underlying SOLID principles. It favors composition over inheritance and interfaces over implementation. Additional details that might obscure the functionality of the system are deliberately omitted while all the business and nonfunctional requirements are addressed. The design heavily uses the following patterns: Command Pattern, Factory Pattern, Composite Pattern, Observer Pattern, Façade Pattern, Proxy Pattern and Singleton Pattern.

The design fully addresses the requirements such as provisioning a flight. As the requirement says, *“The goal is to have successful and strategic flights. The system must be dependable for customers to trust ISTS on their safety and emergencies must be addressed swiftly when they occur. Potential discovery sites must be prioritized to maximize the yield”*. Strategic plans have been implemented to address emergencies by RescueMissionCommand class. The workflow in scheduling new flights also requires checking past missions and checking the history of the destination. With reporting and analytics put in place, administrators and the staff can make smarter decisions.

There is some data redundancy of the flight object to avoid inter-dependent modules. This aligns with the CAP theorem as well, making data conveniently available comes at the cost of consistency. Eventual consistency is favored in this case where FMS schedules a new flight and the list of available flights in RMS and CMS eventually stay in sync. To ensure less coupling, a messaging system maybe used to have all the modules in sync about flight details.

The component diagram included here for a recap to reinforce how the components interact with minimal dependencies.



Exception Handling

All the exceptions implement a marker interface and have two attributes.

reason – why the exception occurred

fix – how to triage it

The GUI is designed to have flash messages when there are errors. Server-side errors are not to be displayed to end users. There will be an API gateway seating in front of the APIs where all calls are propagated through and errors are handled. It is beyond the scope of this design, but it will be one more piece to consider during implementation

Testing

- **Functional** - this will be helpful as the business logic is complex. Testing a restful API with POSTMAN or a similar tool will be suitable
- **Performance** – This is imperative to resolve scalability issues. The system needs to be highly performant
- **Regression** - The system will benefit from end to end testing using automation frameworks like Selenium
- **Exception Handling** – An API gateway limiting the stack trace sent to the GUI and client side logic to handle exceptions must be put in place

Risks

These must be more thoroughly considered

- **Persistence**
- **Security**
- **Client-side exception handling**