Interplanetary Space Transport System

Design Document

Date: 12/18/2019
Author: Evelyn Taylor-McGregor
Reviewer(s): Shantha Kumari Rajendran, Ian Dinwoodie

Table of Contents

I. ISTS

Introduction

The document details the design of the Interplanetary Space Transport System (ISTS). The ISTS is composed of five separate modules: 1) Resource Service, 2) Customer Service, 3) Flight Service, 4) Authentication Service, 5) Ledger Service, 6) Interplanetary File System (IPFS). This design document includes designs for the first three aforementioned modules.

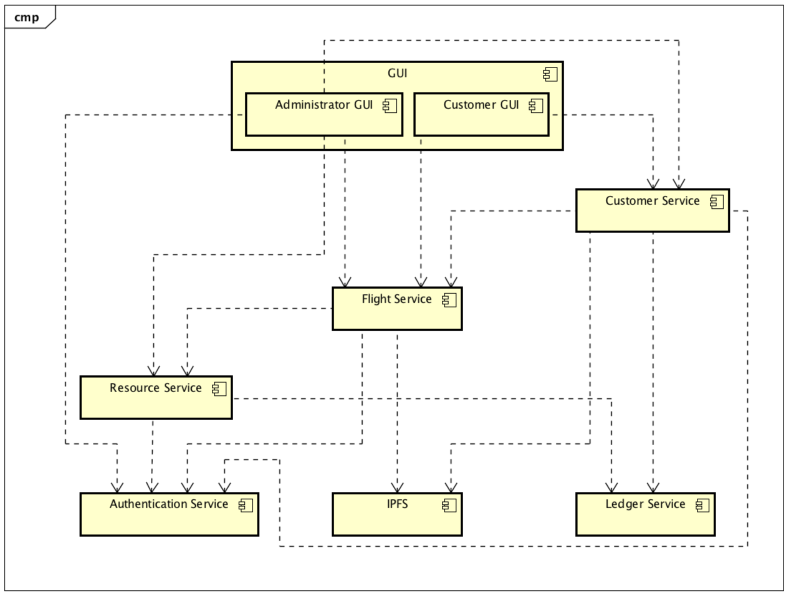
Overview

Product

The International Space Agency has invested in a space transport program, the Interplanetary Space Transport System detailed in this document. The purpose of the transport system is to enable scientific discoveries as well as space tourism. The system is expected to grow in coming years and support a significant number of spacecraft and passengers as space travel becomes more affordable and commonplace.

Design

The software for the ISTS will manage the resources required for space travel, coordinate the booking and monitoring of flights, and provide the ability to record scientific findings and customer feedback using the Interplanetary File System (IPFS). The IPFS is also used for in-flight entertainment and managing travel documents. Each module within the ISTS system has restricted access, enable by an Authentication Service. A blockchain Ledger Service is used to manage payments for flights. The ISTS system is usable through a graphical user interface (GUI) for both administrators and customers.



Component diagram: the ISTS is comprised of 6 modules and is accessible to administrators and customers through a GUI. The modules are organized into four levels, with the GUIs at the *highest level with no dependencies, to the Authentication Service, the IPFS, and the Ledger at the fourth level with only modules dependent on them.*

Requirements

General requirements:

- The ISTS relies on 6 modules: Ledger Service, Authentication Service, IPFS, Resource Service, Customer Service, and Flight Service
- The Resource Service, Customer Service, and Flight Service have service interfaces (APIs)
- Users and administrators are able to interact with the ISTS through a GUI

Ledger Service requirements:

- Manage the ISTS account
- Manage passenger accounts
- Record the transactions for purchasing flights
- Deposit revenue from flight purchases in the ISTS account

Authentication Service requirements:

- Enable admin and customer log in through the GUI (using bio prints)
- Control access to flights
- Control access to APIs

IPFS requirements:

- Store travel documents such as flight information and boarding passes
- Store in-flight entertainment
- Store information about points of interest
- Store discoveries
- Store customer feedback
- Store mission reports
- Enable retrieval of data stored in the IPFS

Resource Service requirements:

- Support creation and management of resources
- Serve as a record of state for the ISTS system
- Enable monitoring of objects within the ISTS system
- Report events issued from hardware within the ISTS system using a communication/computer system
- Maintain the budget of the ISTS
- Allow resources to be viewed through the GUI

Flight Service requirements:

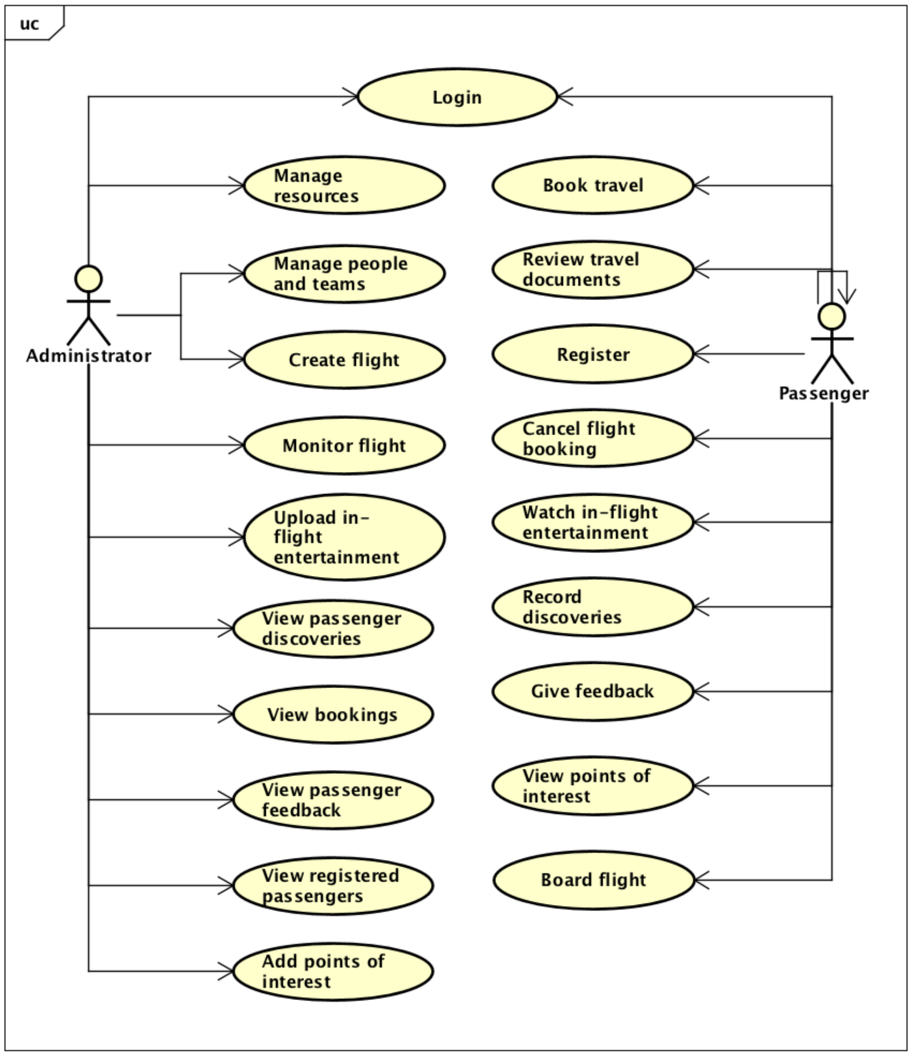
- Manage flights
- Maintain the current status of each flight
- Respond to events reported to the Flight Service system:

  - Flight status updates
  - Mission report requests
  - Send rescue missions
  - Record new discoveries
- Allow flight details to be viewed through the GUI

Customer Service requirements:

- Support booking of travel
- Allow passengers to view travel documents
- Allow passengers to purchase tickets and record the transactions using the Ledger Service
- Allow passengers to cancel their bookings
- Allow passengers to provide feedback
- Allow passengers to report discoveries
- Allow passengers to view in-flight entertainment
- Allow passengers to view points of interest while in-flight
- Maintain records of bookings
- Maintain records of tickets for booked flights
- Maintain record of registered passengers, including Ledger Service account IDs and travel credentials
- Use the IPFS API to store travel documents, passenger feedback and discoveries, in-flight entertainment, and points of interest

Use Cases



Use case diagram: there are two actors on the ISTS, administrators and passengers, each with their own, separate set of use cases of the ISTS. The one overlapping use case is that both administrators and passengers must log in to use the GUI for the ISTS

Actors

- Admin
    - An administrator is a user of the ISTS that has specialized privileges that allow the user to alter the state of the system. The administrator must log in to the administrator GUI first.
- Passenger
    - A passenger is a customer of the ISTS. Passengers log in to and then use the customer GUI, which allows primarily for booking flights and interacting with the IPFS (either through uploading information about flights or requesting to view information such as in-flight entertainment or the customer's boarding pass)

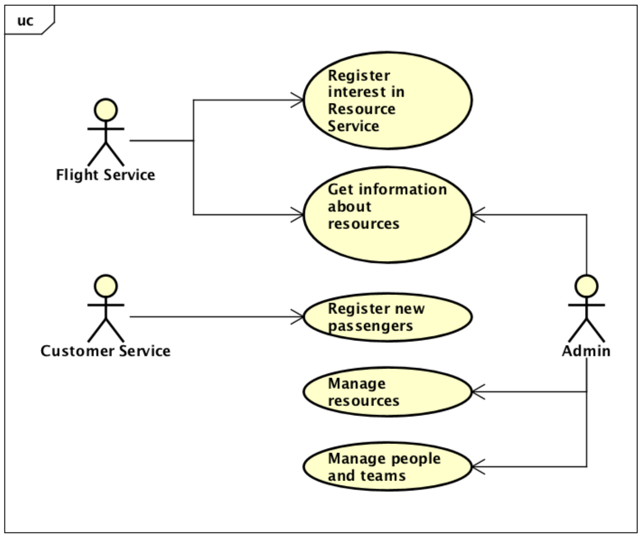
Use Cases

- Login
  - Customers and Administrators must both log in to the GUI in order to interact with the ISTS. Authentication is managed by the Authentication Service.
- Manage resources
  - The Administrator may use the Admin GUI to manage resources within the ISTS system. This includes adding resources (such as spacecraft, launchpads, and fuel) and updating resources (such as the ISTS budget and fuel stores).
- Manage people and teams
  - Admins can add people resources (people and teams) and update the structures of teams.
- Create flight
  - Admins can create (i.e. schedule) a flight by specifying the resources (such as the spaceship, launchpad, and crew) and the flight plan (departure time, destination, stops, etc.).
- Monitor flight
  - Admins can view the current status of any flight through the Admin GUI.
- Book travel
  - Customers may book travel by selecting a flight on the Customer GUI and paying for the flight using their Ledger account. Customers must be registered with the ISTS in order to book and pay for flights.
- Review travel documents
  - Customers may review travel documents for their upcoming flights using the Customer GUI.
- Register
  - In order book flights, Customers must register with the Customer Service. To register, Customers must have a Ledger account and provide their name and a unique username.
- Board flight
  - Customers may board flights if they possess a boarding pass.
- View in-flight entertainment
  - Customers may view in-flight entertainment when on flights.
- Give feedback/record discovery
  - While on flights, Customers can give feedback on their flight and record any discoveries along the way.
- Upload in-flight entertainment
  - Admins must upload files for in-flight entertainment in order for it to be accessible for passengers to watch on their flights
- View passenger discoveries
  - Admins may view passenger discoveries through the Admin GUI
- View passenger feedback
  - Admins may view passenger feedback through the Admin GUI
- View registered passengers
  - Admins may view all registered passengers known to the ISTS through the Admin GUI

- Add points of interest
    - Admins may upload new points of interest for passengers to view during their flights

II. Resource Service

Use Case



Resource Service use case diagram: the three actors on the Resource Service are the Customer Service, Flight Service, and Administrators

Actors

- Customer Service
    - The Customer Service uses the Resource Service API to create records of new passengers that register through the Customer Service GUI.
- Flight Service
    - The Flight Service uses the Resource Service API to validate flights scheduled through the Admin GUI, and to get information about space craft, launch pads, and fuel stores.
    - The Flight Service also registers interest in the Resource Service API and monitors it for events.
- Administrators
    - Administrators act on the Resource Service API through the Admin GUI to provision the ISTS with resources and update these resources.

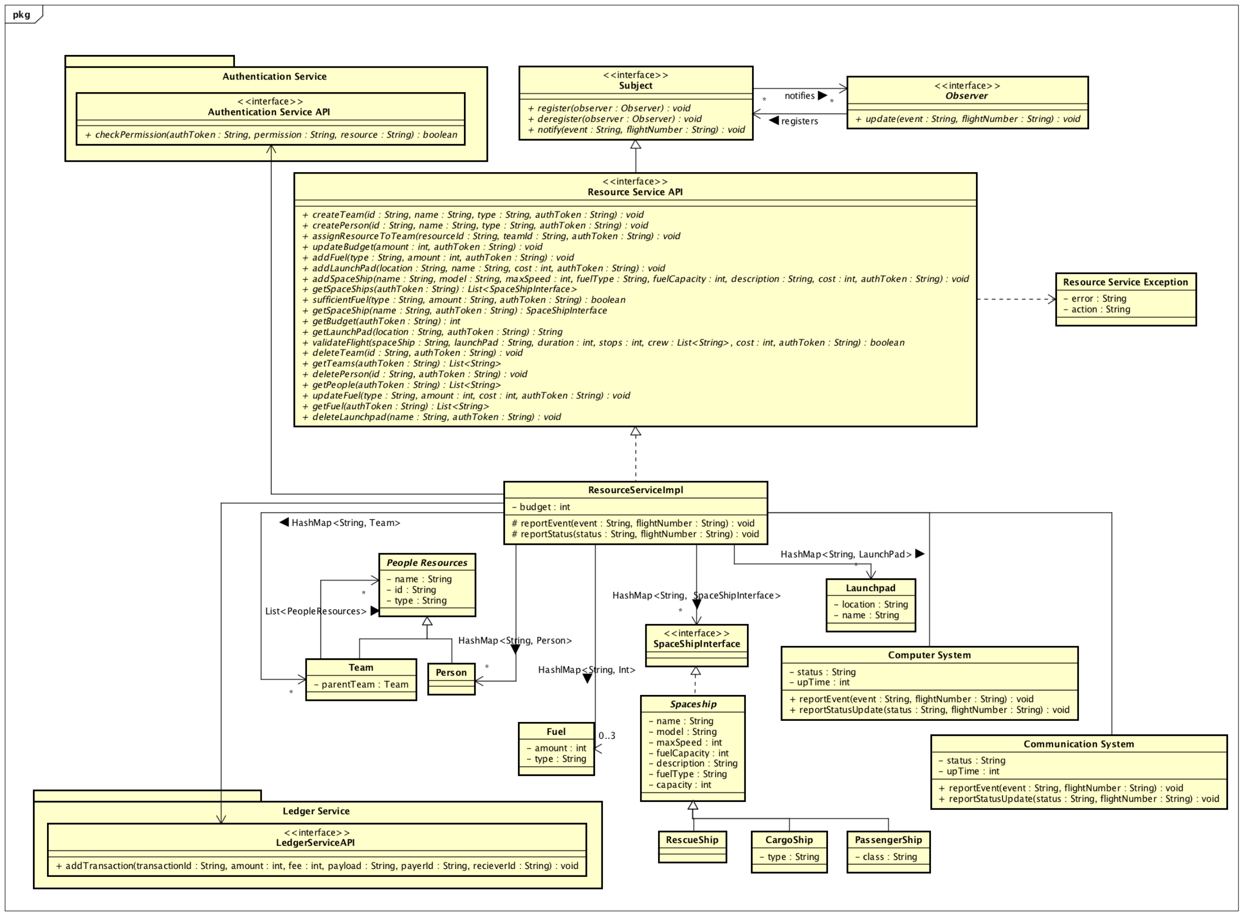
Use cases

- Register interest in Resource Service
    - The Flight Service acts as an Observer (using the Observer Design Pattern) of the Resource Service. The Flight Service registers interest in the Resource

Service so it is notified of events transpiring on any of the objects within the ISTS.

- Get information about resources
    - The Flight Service uses the Resource Service API to get information about resources (space craft, landing pads, fuel) to validate scheduled flights and to maintain details of these resources (for example, the capacity of a passenger space craft is used to determine how many tickets are available for booking).
- Register new passengers
    - When new passengers register with the Customer Service, records of these passengers are created in the Resource Service module.
- Create resources
    - The Administrator may use the Admin GUI to manage resources within the ISTS system. This includes adding resources (such as spacecraft, launchpads, and fuel
- Update state of resources
    - The Administrator may use the Admin GUI to manage resources within the ISTS system. This includes updating resources (such as the ISTS budget and fuel stores).
- Manage people and teams
    - Admins can add people resources (people and teams) and update the structures of teams.

Class Diagram


pkg
Authentication Service
<<interface>>
Authentication Service API
+ checkPermission(authToken : String, permission : String, resource : String) : boolean
<<interface>>
Subject
+ register(observer : Observer) : void
+ deregister(observer : Observer) : void
+ notify(event : String, flightNumber : String) : void
notifies ▶
◀ registers
*
*
<<interface>>
Observer
+ update(event : String, flightNumber : String) : void
<<interface>>
Resource Service API
+ createTeam(id : String, name : String, type : String, authToken : String) : void
+ createPerson(id : String, name : String, type : String, authToken : String) : void
+ assignResourceToTeam(resourceId : String, teamId : String, authToken : String) : void
+ updateBudget(amount : int, authToken : String) : void
+ addFuel(type : String, amount : int, authToken : String) : void
+ addLaunchPad(location : String, name : String, cost : int, authToken : String) : void
+ addSpaceShip(name : String, model : String, maxSpeed : int, fuelType : String, fuelCapacity : int, description : String, cost : int, authToken : String) : void
+ getSpaceShips(authToken : String) : List<SpaceShipInterface>
+ sufficientFuel(type : String, amount : String, authToken : String) : boolean
+ getSpaceShip(name : String, authToken : String) : SpaceShipInterface
+ getBudget(authToken : String) : int
+ getLaunchPad(location : String, authToken : String) : String
+ validateFlight(spaceShip : String, launchPad : String, duration : int, stops : int, crew : List<String>, cost : int, authToken : String) : boolean
+ deleteTeam(id : String, authToken : String) : void
+ getTeams(authToken : String) : List<String>
+ deletePerson(id : String, authToken : String) : void
+ getPeople(authToken : String) : List<String>
+ updateFuel(type : String, amount : int, cost : int, authToken : String) : void
+ getFuel(authToken : String) : List<String>
+ deleteLaunchpad(name : String, authToken : String) : void
Resource Service Exception
- error : String
- action : String
ResourceServiceImpl
- budget : int
# reportEvent(event : String, flightNumber : String) : void
# reportStatus(status : String, flightNumber : String) : void
◀ HashMap<String, Team>
HashMap<String, LaunchPad> ▶
People Resources
- name : String
- id : String
- type : String
List<PeopleResources> ▶
HashMap<String, Person>
HashMap<String, SpaceShipInterface>
Launchpad
- location : String
- name : String
<<interface>>
SpaceShipInterface
Team
- parentTeam : Team
Person
HashlMap<String, Int>
Computer System
- status : String
- upTime : int
+ reportEvent(event : String, flightNumber : String) : void
+ reportStatusUpdate(status : String, flightNumber : String) : void
Fuel
- amount : int
- type : String
0..3
Spaceship
- name : String
- model : String
- maxSpeed : int
- fuelCapacity : int
- description : String
- fuelType : String
- capacity : int
Communication System
- status : String
- upTime : int
+ reportEvent(event : String, flightNumber : String) : void
+ reportStatusUpdate(status : String, flightNumber : String) : void
Ledger Service
<<interface>>
LedgerServiceAPI
+ addTransaction(transactionId : String, amount : int, fee : int, payload : String, payerId : String, recieverId : String) : void
RescueShip
CargoShip
- type : String
PassengerShip
- class : String

Class diagram: Diagram detailing the relationship between classes within the Resource Service module. The Resource Module uses the Façade Design Pattern, the Singleton Pattern, the Composite Pattern, and the Observer Pattern.

Class Dictionary

Resource Service

The Resource Service class implements the Resource Service API interface, which extends the Subject API interface. The Resource Service class is the entry point for users into the Resource Service module and the interface is used by the Flight Service, the Customer Service, and the Admin GUI. The **Façade Pattern** is used as the implementation of the Resource Service is hidden by the interface. The Resource Service has a factory method for generating a singleton instance, which follows the **Singleton Pattern.** The Resource Service also uses the **Observer Pattern** to allow the Flight Service module to register interest in it. The **Observer Pattern** is useful because it enables the Flight Service to receive and act on messages from the Resource Service such as SOS messages and other status updates, which is a requirement of the ISTS.

Properties

Property Name	Type	Description

budget	int	The current account balance of the ISTS.

Associations

Association Name	**Type**	**Description**
fuelMap	HashMap<String, Fuel>	A mapping of fuel type to fuel objects. This association keeps track of how much fuel the ISTS has available to use for flights.
launchpadMap	HashMap<String, Launchpad>	A mapping of launch pad name to the launchpad object.
teamMap	HashMap<String, Team>	A mapping of team id to a team object. This map keeps track of all teams created within the ISTS.
personMap	HashMap<String, Person>	A mapping of person id to person object. This map keeps track of all people known to the ISTS.
computerSystem	ComputerSystem	The central computer system for the ISTS.
communicationSystem	CommunicationSystem	The central communication system for the ISTS.
spaceshipMap	HashMap<String, SpaceshipInterface>	A mapping of spaceship ID to an interface for the spaceship.
authenticationService	AuthenticationService	The authentication service used to control access to restricted resources. The Resource Service module uses the Authentication Service to verify that auth tokens passed to the API are attached to roles that have sufficient permissions to call the restricted method.
ledgerService	LedgerService	The Ledger Service is used to record payments for resources: spaceships, launchpads, and fuel. The payments for these resources come out of the ISTS account in the ledger. The budget property is maintained by checknig the account balance of the ISTS account and updating the budget proprety.

Methods

Method Name	Signature	Description
createTeam	(id : String, name : String, type : String, authToken : String) : void	Creates a team within the Resource Service.
deleteTeam	(id: String, authToken: String): void	Removes a record of a team within the Resource Service. If the team does not exist, a ResourceServiceException is thrown.
getTeams	(authToken: String): List<String>	Returns a list of Strings detailing the teams known to the Resource Service.
createPerson	(id : String, name : String, type : String, authToken : String) : void	Creates a record of a person within the Resource Service.
deletePerson	(id: String, authToken: String): void	Removes a record of a person within the Resource Service. If the person does not exist, a ResourceServiceException is thrown.
getPeople	(authToken: String): List<String>	Returns a list of Strings providing details on all the people known to the Resource Service.
assignResource ToTeam	(resourceId : String, teamId : String, authToken : String) : void	Assigns a resource (either team or person) to a team. Both team and resource must be already defined or a ResourceServiceException is thrown.
updateBudget	(amount : int, authToken : String) : void	Adds budget to the ISTS account in the Ledger Service.
getBudget	(authToken : String) : int	Returns the current budget of the ISTS.
addFuel	(type : String, amount : int, cost: int, authToken : String) : void	Adds fuel of a given type to the ISTS. The cost of the fuel is subtracted from the ISTS's ledger account.
updateFuel	(type : String, amount : int, authToken : String) : void	Updates the amount of fuel of a given type available to the ISTS.

getFuel	(authToken: String): List<String>	Returns a list of Strings detailing the current fuel stores of the ISTS.
addLaunchpad	(location : String, name : String, cost: int, authToken : String) : void	Adds a launch pad to the resources of the ISTS. The cost of the launchpad is subtracted from the ISTS account in the ledger.
deleteLaunchpad	(name : String, authToken : String) : void	Removes a record of a launch pad from the ISTS. If the launchpad is not known, a ResourceServiceException is thrown.
getLaunchpad	(location : String, authToken : String) : String	Returns a String of details about a given launchpad. If the launchpad is not known, a ResourceServiceException is thrown.
getLaunchpads	(authToken: String): List<String>	Returns a list of Strings with details about all the launchpads known to the ISTS.
addSpaceship	(name : String, model : String, maxSpeed : int, fuelType : String, fuelCapacity : int, description : String, cost: int, authToken : String) : void	Adds a new spaceship to the ISTS resources. The cost of the spaceship is subtracted from the ISTS account in the ledger.
getSpaceships	(authToken : String) : List<SpaceShipInterface>	Returns a list of Spaceship interfaces for all spaceships known to the ISTS.
getSpaceShip	(name : String, authToken : String) : SpaceShipInterface	Returns a single Spaceship interface. If the spaceship does not exist, throws a ResourceServiceException.
deleteSpaceship	(name: String, authToken: String): void	Removes a record of a spaceship. If the spaceship does not exist, throws a ResourceServiceException.
validateFlight	(spaceShip : String, launchPad : String, duration : int, stops : int, crew : List<String>, cost : int, authToken : String) : boolean	Validates whether a proposed flight is possible given the current resources available to the ISTS. This method is used by the Flight Service to validate flights provisioned by administrators through the GUI.
reportEvent	(event: String, flightNumber:	Private method called by the communication

	String): void	or computer systems to report any detected event on a flight. Calls the notify method.
reportStatus	(status: String, flightNumber: String): void	Private method called by the communication or computer systems to report any status changes on a flight. Calls the notify method.
notify	(event: String, flightNumber: String): void	Called by the Resource Service to notify its observers of events. Loops through all currently registered observers and calls the update method on the observers. This method is part of the **Observer Pattern**.
register	(observer: Observer): void	Called by an observer to register interest in the Resource Service (which extends the Subject interface). This method is part of the **Observer Pattern**.
deregister	(observer: Observer): void	Called by an observer to deregister interest in the Resource Service (which extends the Subject interface). This method is part of the **Observer Pattern**.

Spaceship

The spaceship abstract class implements the SpaceShipInterface. The SpaceShipInterface has only get methods to show details of the spaceship without exposing the spaceship object itself. The spaceship class contains all the properties of the given spaceship, depending on the type of spaceship.

Properties

Property Name	**Type**	**Description**
name	String	The current account balance of the ISTS.
model	String	The alphanumeric model name of the spaceship.
maxSpeed	int	The maximum speed of the spaceship.
fuelCapacity	int	The maximum amount of fuel that can fit in the spaceship.
fuelType	String	The type of fuel the spaceship requires.
description	String	Text description of the spaceship.

capacity	int	Passenger capacity of the ship.

Methods

Method Name	**Signature**	**Description**
getName	(authToken: String): String	Getter for the spaceship's name
getModel	(authToken: String): String	Getter for the spaceship's model
getMaxSpeed	(authToken: String): int	Getter for the spaceship's max speed
getCapacity	(authToken: String): int	Getter for the spaceship's passenger capacity
getFuelCapacity	(authToken: String): int	Getter for the spaceship's fuel capacity
getDescription	(authToken: String): String	Getter for the spaceship's description
getFuelType	(authToken: String): String	Getter for the spaceship's fuel type
getType	(authToken: String): String	Getter for the spaceship's type: either Passenger, Cargo, or Rescue.
getClass	(authToken: String): String	If the spaceship is a passenger ship, getter for the class of the passenger ship.
getCargoType	(authToken: String): String	If the spaceship is a cargo ship, getter for the type of cargo ship.

PassengerShip

The PassengerShip class extends the Spaceship class and has an additional property: class, which is either Luxury or Economy.

Properties

Property Name	**Type**	**Description**
name	String	The current account balance of the ISTS.
model	String	The alphanumeric model name of the spaceship.
maxSpeed	int	The maximum speed of the spaceship.
fuelCapacity	int	The maximum amount of fuel that can fit in the spaceship.

fuelType	String	The type of fuel the spaceship requires.
description	String	Text description of the spaceship.
capacity	int	Passenger capacity of the ship.
class	String	Luxury of Economy.

CargoShip

The CargoShip class extends the Spaceship class and has one additional property: type, which is one of Mining, Satellite Maintenance, or Construction Equipment.

Properties

Property Name	Type	Description
name	String	The current account balance of the ISTS.
model	String	The alphanumeric model name of the spaceship.
maxSpeed	int	The maximum speed of the spaceship.
fuelCapacity	int	The maximum amount of fuel that can fit in the spaceship.
fuelType	String	The type of fuel the spaceship requires.
description	String	Text description of the spaceship.
capacity	int	Passenger capacity of the ship.
type	String	Type of cargo ship.

RescueShip

The CargoShip class extends the Spaceship class and has no additional properties.

Properties

Property Name	Type	Description
name	String	The current account balance of the ISTS.
model	String	The alphanumeric model name of the spaceship.

maxSpeed	int	The maximum speed of the spaceship.
fuelCapacity	int	The maximum amount of fuel that can fit in the spaceship.
fuelType	String	The type of fuel the spaceship requires.
description	String	Text description of the spaceship.
capacity	int	Passenger capacity of the ship.

Computer System

The Computer System represents the central computer system of the ISTS. This system is connected to the hardware items known to the ISTS and is connected to a network on earth. The Computer System reports events and status changes to the Resource Service.

Properties

Property Name	Type	Description
status	String	The current status of the computer system.
upTime	int	The up-time of the computer system (in seconds).

Associations

Property Name	Type	Description
resourceService	ResourceServiceImpl	An association with the resource service, which the computer system calls to report an event or a change of status.

Methods

Method Name	Signature	Description
reportEvent	(event: String, flightNumber: String): void	Reports an event on a flight to the Resource Service by calling the reportEvent method on the Resource Service API.
reportStatusUpdate	(status: String): void	Reports a status update to the Resource Service by calling the reportStatus method on the Resource Service API.

Communication System

The Communication System represents the central communication system of the ISTS. This system is connected to the hardware items known to the ISTS and is connected to a network on earth. The Communication System reports events and status changes to the Resource Service.

Properties

Property Name	Type	Description
status	String	The current status of the computer system.
upTime	int	The up-time of the computer system (in seconds).

Associations

Property Name	Type	Description
resourceService	ResourceServiceImpl	An association with the resource service, which the communication system calls to report an event or a change of status.

Methods

Method Name	Signature	Description
reportEvent	(event: String, flightNumber: String): void	Reports an event to the Resource Service by calling the reportEvent method on the Resource Service API.
reportStatusUpdate	(status: String, flightNumber: String): void	Reports a status update to the Resource Service by calling the reportStatus method on the Resource Service API.

Launchpad

The launchpad class is a representation of a physical launchpad on earth or any other location within the solar system. A launchpad is needed for a spaceship to take off from.

Properties

Property Name	Type	Description
name	String	The unique name of the launchpad.
location	String	The location of the launchpad.

Fuel

The fuel class is a representation of a type and amount of fuel stored somewhere within the ISTS system. There are three types of fuel: Ion Drive, Solar Sail, and Hydrogen/Oxygen. The Resource Service maintains a fuel object represents each of these types of fuel, and the amount specified is how much of that fuel type the ISTS has in total. The fuel class is also used by the Spaceship class to record how much fuel a Spaceship has within it at a given time.

Properties

Property Name	**Type**	**Description**
type	String	The type of fuel (Ion Drive, Solar Sail, or Hydrogen/Oxygen)
amount	String	The amount of fuel.

PeopleResources

The PeopleResources class is an abstract class that represents either a team or a person within the ISTS. The relationship between the PeopleResources class, the Team class, and the Person class follows the **Composite Pattern**. This fulfils the requirement to manage an arbitrary number of people, teams, and potentially subteams as the ISTS grows.

Properties

Property Name	**Type**	**Description**
name	String	Unique name of the team or person.
Id	String	Unique ID of the team or person.
type	String	Type of team or person (one of: Operations, Flight Crew, Passenger or Rescue)

Team

A representation of a team within the ISTS. The Team class extends the PeopleResources abstract class. In addition to the properties of the PeopleResources class, the Team class has associations with its parent team, and composes PeopleResources (Composite Pattern). A team object can have associations with other teams (sub-teams) or team members, or both.

Properties

Property Name	**Type**	**Description**
name	String	Unique name of the team.

id	String	Unique ID of the team.

Associations

Association Name	Type	Description
parentTeam	PeopleResource	Reference to a team's parent team, if it exists.
peopleResources	List<PeopleResources>	A list of sub-teams or people within the team.

Person

The Person class extends the abstract PeopleResources class.

Properties

Property Name	Type	Description
name	String	Unique name of the team or person.
Id	String	Unique ID of the team or person.
type	String	Type of team or person (one of: Operations, Flight Crew, Passenger or Rescue)

Subject

The Subject interface is used by classes that other classes may want to monitor. The Subject interface is part of the Observer pattern.

Methods

Method Name	Type	Description
register	(observer: Observer): void	Called by an observer to register interest in the subject and receive notifications.
deregister	(observer: Observer): void	Called by an observer to un-register its interest in the subject and stop receiving notifications.
notify	(event: String, flightNumber: String): void	Method called by the concrete subject to notify any observers of changes.

Observer

An interface used for observing subjects. Part of the Observer Pattern. The Observer interface has one method: update, which is called by concrete Subjects to notify their Observers of events.

Methods

Method Name	Type	Description
update	(event: String, flightNumber: String): void	Updates an observer of an event within a subject.

ResourceServiceException

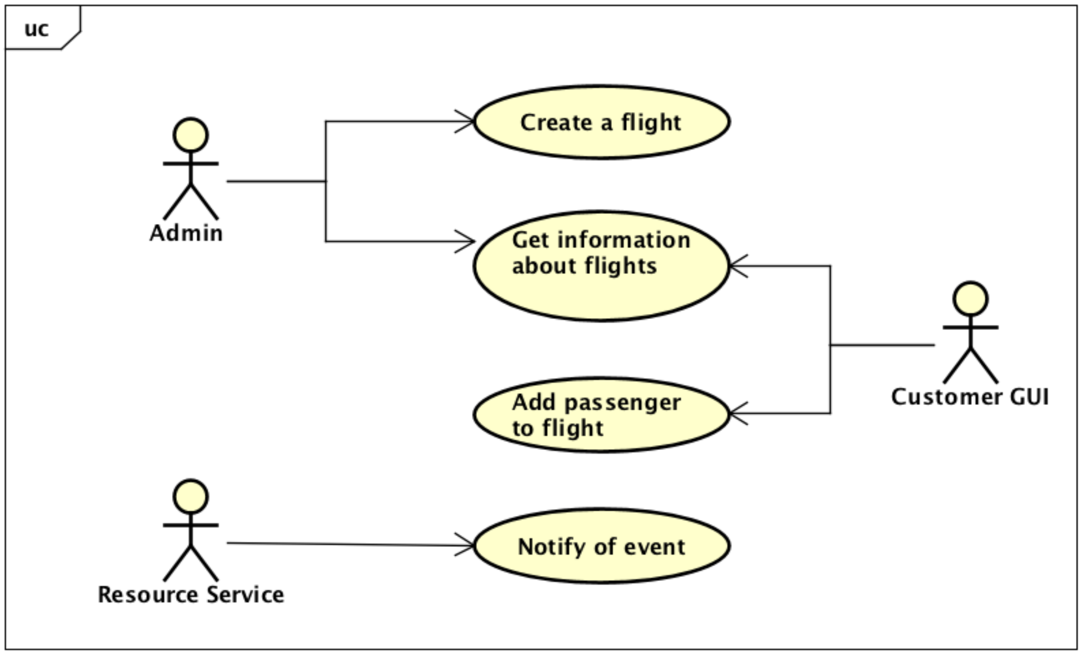
An exception class for the ResourceService.

Properties

Property Name	Type	Description
error	String	The error raised.
action	String	The action that raised the error.

III. Flight Service

Use Case



Flight Service use case diagram: the three actors on the Resource Service are the Customer Service, Customer GUI, and Administrators

Actors

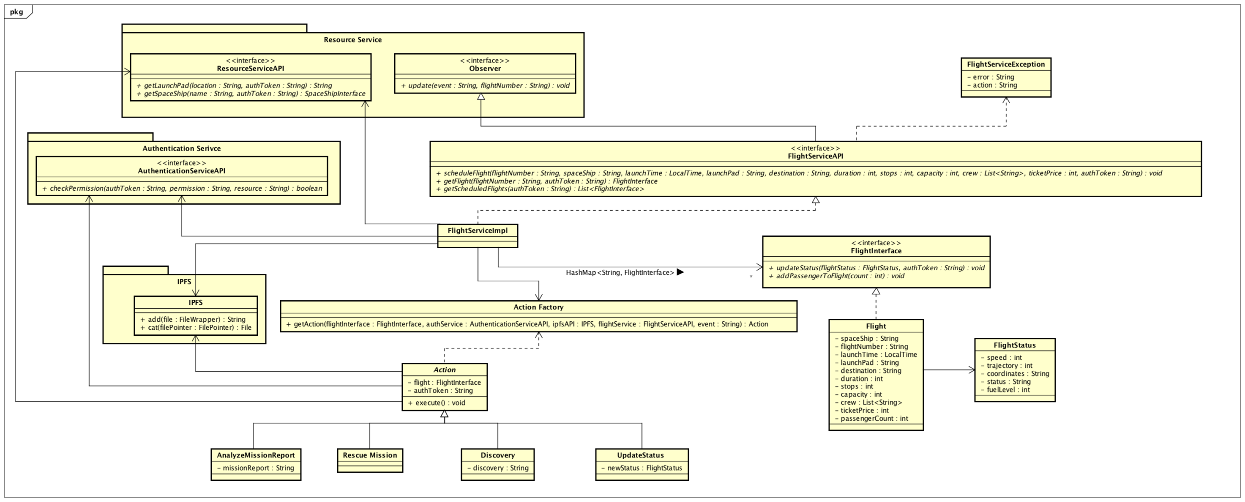
- Customer GUI
  - Flights are displayed to customers through the customer GUI
  - When a customer books a flight, the passenger count is incremented through the Flight Service API
- Resource Service
  - The Resource Service is a Subject and the Flight Service is an Observer, therefore the Resource Service notifies the Flight Service of events.
- Administrators
  - Administrators, through the Admin GUI, can schedule new flights, monitor the status of flights, and view all scheduled flights.

Use cases

- Create a flight
  - Admins can create (i.e. schedule) a flight by specifying the resources (such as the spaceship, launchpad, and crew) and the flight plan (departure time, destination, stops, etc.).
- Monitor/view flights
  - Admins can view the current status of any flight through the Admin GUI
  - Customers can browse flights to decide whether to book travel
- Add passenger to flight
  - When Customers book a flight, the passenger count (maintained by the Flight Service module) is incremented
- Notify of an event

- When an event is reported through the computer or communication systems within the Resource Service module, the Flight Service is notified.

Class Diagram


pkg
Resource Service
<<interface>>
ResourceServiceAPI
+ getLaunchPad(location : String, authToken : String) : String
+ getSpaceShip(name : String, authToken : String) : SpaceShipInterface
<<interface>>
Observer
+ update(event : String, flightNumber : String) : void
FlightServiceException
- error : String
- action : String
Authentication Serivce
<<interface>>
AuthenticationServiceAPI
+ checkPermission(authToken : String, permission : String, resource : String) : boolean
<<interface>>
FlightServiceAPI
+ scheduleFlight(flightNumber : String, spaceShip : String, launchTime : LocalTime, launchPad : String, destination : String, duration : int, stops : int, capacity : int, crew : List<String>, ticketPrice : int, authToken : String) : void
+ getFlight(flightNumber : String, authToken : String) : FlightInterface
+ getScheduledFlights(authToken : String) : List<FlightInterface>
FlightServiceImpl
HashMap<String, FlightInterface> ▶
*
<<interface>>
FlightInterface
+ updateStatus(flightStatus : FlightStatus, authToken : String) : void
+ addPassengerToFlight(count : int) : void
IPFS
IPFS
+ add(file : FileWrapper) : String
+ cat(filePointer : FilePointer) : File
Action Factory
+ getAction(flightInterface : FlightInterface, authService : AuthenticationServiceAPI, ipfsAPI : IPFS, flightService : FlightServiceAPI, event : String) : Action
Flight
- spaceShip : String
- flightNumber : String
- launchTime : LocalTime
- launchPad : String
- destination : String
- duration : int
- stops : int
- capacity : int
- crew : List<String>
- ticketPrice : int
- passengerCount : int
FlightStatus
- speed : int
- trajectory : int
- coordinates : String
- status : String
- fuelLevel : int
Action
- flight : FlightInterface
- authToken : String
+ execute() : void
AnalyzeMissionReport
- missionReport : String
Rescue Mission
Discovery
- discovery : String
UpdateStatus
- newStatus : FlightStatus

Class diagram: Diagram detailing the relationship between classes within the Flight Service module. The Flight Service uses the Observer Pattern, the Factory Pattern, and the Command Pattern.

Class Dictionary

Flight Service Implementation

The Flight Service implements the Flight Service API interface, which extends the Observer interface from the Resource Service package. The Flight Service is responsible for the scheduling of flights within the ISTS. The Flight Service maintains records of all scheduled flights. The Flight Service is an Observer (**Observer Pattern)** of the Resource Service and receives updates from the Resource Service, which the Flight Service executes actions in response to (using the Command Pattern).

Associations

Association Name	**Type**	**Description**
actionFactory	ActionFactory	A reference to an Action Factory that parses events from Subjects and creates the appropriate action class.
flights	HashMap<String, FlightInterface>	A mapping of flight number to the corresponding flight interface.
authenticationSe	AuthenticationService	The Authentications Service that is used to

rvice		control access to restricted methods within the ISTS. The Flight Service uses the Authentication Service to validate auth tokens passed to the API to ensure that they are associated with a role that has permission to execute the restricted method.
resourceService	ResourceService	The Resource Service that maintains the state of all the resources within the ISTS. The Flight Service relies on the Resource Service API to validate flights and get information about spaceships.
ipfs	IPFS	The file system that the ISTS uses. The Flight Service uses the IPFS to store discovery details.

Methods

Method Name	**Signature**	**Description**
scheduleFlight	(flightNumber : String, spaceShip : String, launchTime : LocalTime, launchPad : String, destination : String, duration : int, stops : int, capacity : int, crew : List<String>, ticketPrice : int, cost : int, authToken : String) : void	Method called to schedule a flight. The details of the flight must be specified in order to schedule the flight and validate that there are enough resources to complete the flight. The Flight Service calls the Resource Service API's validateFlight method to validate the scheduled flight.
getFlight	(flightNumber : String, authToken : String) : FlightInterface	Returns a Flight Interface for the specified flight number. If the flight does not exist, a FlightServiceException is thrown.
getScheduledFlights	(authToken : String) : List<FlightInterface>	Returns a list of Flight Interfaces for the scheduled flights.

Flight

The Flight class represents a scheduled flight within the ISTS. The Flight class implements the FlightInterface.

Properties

Property Name	**Type**	**Description**

spaceship	String	The unique ID of the spaceship completing the flight.
flightNumber	String	The unique flight number.
launchTime	Time	The time of departure for the flight.
launchPad	String	The launchpad the spaceship will take off from.
destination	String	The destination of the flight.
duration	int	The fly time of the flight.
stops	int	The number of stops the flight will make throughout the route.
capacity	int	The maximum number of people that can be on the flight.
crew	List<String>	The crew members scheduled to manage the flight.
ticketPrice	int	The price of tickets for the flight.
passengerCount	int	The current number of passengers.

Associations

Association Name	**Type**	**Description**
flightStatus	FlightStatus	The current status of the flight, including its speed, trajectory, coordinates, status, and fuel level.

Methods

Method Name	**Signature**	**Description**
updateStatus	(flightStatus: FlightStatus): void	Method called to schedule a flight. The details of the flight must be specified in order to schedule the flight and validate that there are enough resources to complete the flight. The Flight Service calls the Resource Service API's validateFlight method to

		validate the scheduled flight.
addPassengerToFlight	(count: int): void	Returns a Flight Interface for the specified flight number. If the flight does not exist, a FlightServiceException is thrown.
getSpaceship	(authToken : String) : String	Getter for the flight's spaceship name
getFlightNumber	(authToken : String) : String	Getter for the flight's number
getLaunchTime	(authToken : String) : String	Getter for the flight's launch time
getLaunchpad	(authToken : String) : String	Getter for the flight's launch pad name
getDuration	(authToken : String) : String	Getter for the flight's duration
getDestination	(authToken : String) : String	Getter for the flight's destination
getStops	(authToken : String) : String	Getter for the flight's number of stops
getCapacity	(authToken : String) : String	Getter for the flight's total passenger capacity
getCrew	(authToken : String) : List<String>	Getter for a list of the flight's crew
getTicketPrice	(authToken : String) : String	Getter for the flight's ticket price
getPassengerCount	(authToken : String) : String	Getter for the flight's current passenger count.

FlightStatus

The FlightStatus class represents the current state of a flight. The status object is immutable and is replaced when a new status is reported, which occurs periodically.

Properties

Property Name	**Type**	**Description**
speed	int	The unique ID of the spaceship completing the flight.
trajectory	int	The unique flight number.
coordinates	String	The time of departure for the flight.
status	String	The current status of the flight (Preparing for

		launch, In-flight, Reached Destination, or Lost)
fuelLevel	int	The current amount of fuel left in the spaceship.

ActionFactory

The ActionFactory is used by the Flight Service to parse an event that the Flight Service was notified of and determine the correct action to take. The ActionFactory has one method, which parses the event and creates an Action object, which it returns to the caller. This class uses the **Factory Pattern** which fulfils requirements as it simplifies the process of creating commands in response to events the Flight Service is notified of and makes extension to new events and corresponding commands easier as the ISTS scales.

Methods

Method Name	**Type**	**Description**
getAction	(flightInterface : FlightInterface, authService : AuthenticationServiceAPI, ipfsAPI : IPFSAPI, flightService : FlightServiceAPI, event : String) : Action	Parses the event and creates an Action object with the appropriate properties.

Action

Abstract class with one method, execute. The Action class is part of the **Command Pattern.**

Properties

Property Name	**Type**	**Description**
flight	FlightInterface	The flight on which the event occurred.
authToken	String	The authentication token.

Methods

Method Name	**Type**	**Description**
execute	(): void	Executes the action.

AnalyzeMissionReport

The AnalyzeMissioReport takes a flight interface, a report, an auth token, and a reference to the IPFS instance. When the execute method is called, additional details about the flight in question are gathered, the report is analyzed, and the results are saved in the IPFS.

Properties

Property Name	Type	Description
flight	FlightInterface	The flight to report upon.
authToken	String	The authentication token.
report	String	The report to analyze.

Associations

Association Name	Type	Description
ipfs	IPFS	A reference to an IPFS instance.

Method

Method Name	Type	Description
execute	(): void	Analyzes the report, adds details about the flight, and saves the results in the IPFS.

RescueMission

The RescueMission class extends the Action abstract class. The RescueMission class takes the distressed flight's FlightInterface, an auth token, and a reference to the Flight Service as inputs to its constructor. When the execute method is called, it retrieves information about the distressed flight using its interface then schedules a rescue mission to leave immediately using the Flight Service API.

Properties

Property Name	Type	Description
flight	FlightInterface	The flight that sent the distress signal.
authToken	String	The authentication token.

Associations

Association Name	Type	Description
flightService	FlightServiceAPI	A reference to the FlightServiceAPI.

Method

Method Name	Type	Description
execute	(): void	Gets information about the distressed flight using the distressed flight's FlightInterface and then schedules a rescue flight leaving immediately using the FlightServiceAPI.

Discovery

Concrete class that extends the Action abstract class. The Discovery action's one method, execute, saves the discovery in the IPFS. The discovery is saved with the number of the flight it was discovered on as well by calling the getFlightNumber method on the flight interface.

Properties

Property Name	Type	Description
flight	flightInterface	The flight on which the discovery was discovered.
authToken	String	The authentication token.
discovery	String	The discovery, described in text.

Associations

Association Name	Type	Description
ipfs	IPFS	A reference to an IPFS instance.

Method

Method Name	Type	Description
execute	(): void	Gets the flight number from the flight interface and saves the discovery (with the flight number) in the IPFS.

UpdateStatus

Concrete class that extends the abstract Action class. The UpdateStatus action takes a flight status, flight interface, and auth token as parameters to its constructor; it has one method, execute, which updates the status of the flight by calling the updateStatus method on the flight interface and passing the auth token.

Properties

Property Name	Type	Description
flight	FlightInterface	The interface for the flight to update the status of.
newStatus	FlightStatus	The new status of the flight, as an object.
authToken	String	The authentication token required to update the status of a flight.

Method

Method Name	Type	Description
execute	(): void	Calls the updateStatus method on the flight interface.

FlightServiceException

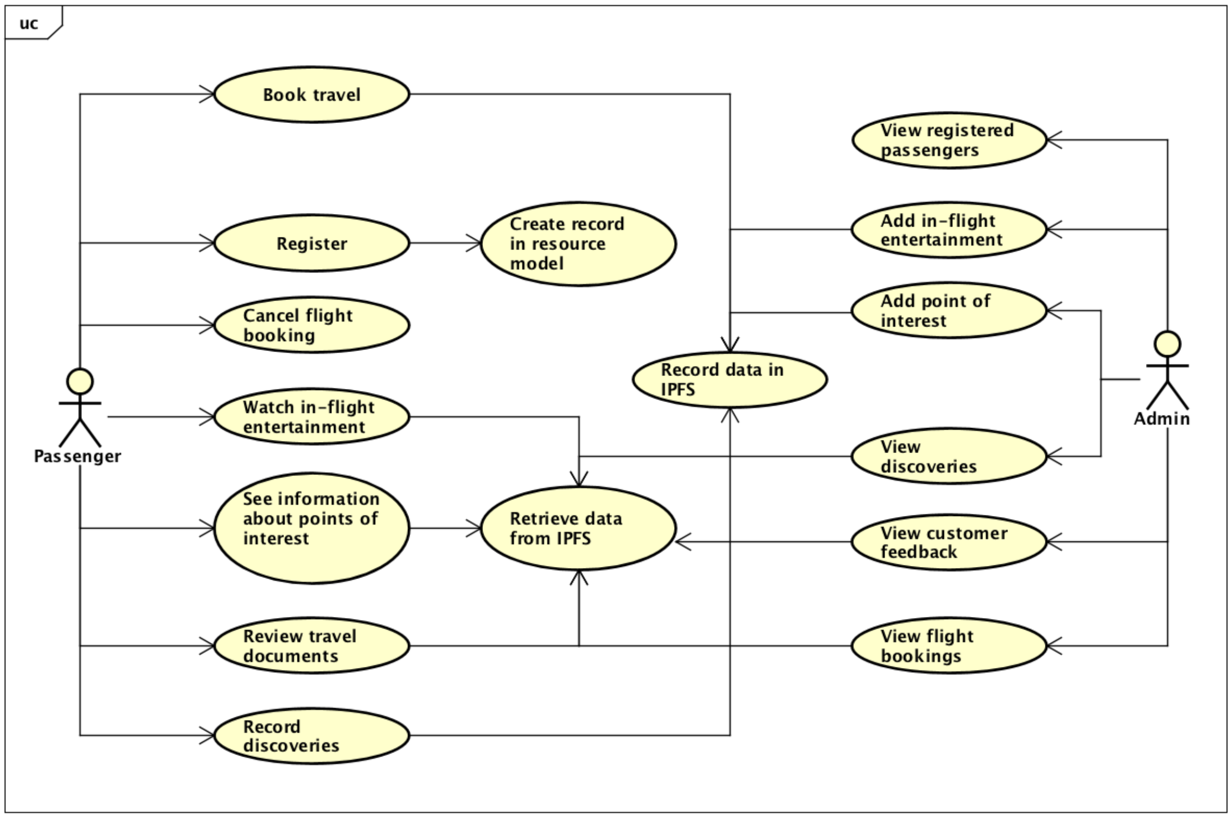
Exeption class for the Flight Service.

Properties

Property Name	Type	Description
error	String	The error raised.
action	String	The action that raised the error.

IV. Customer Service

Use Case



Customer service module use case diagram: the actor on the Customer Service is only a passenger (i.e. customer)

Actors

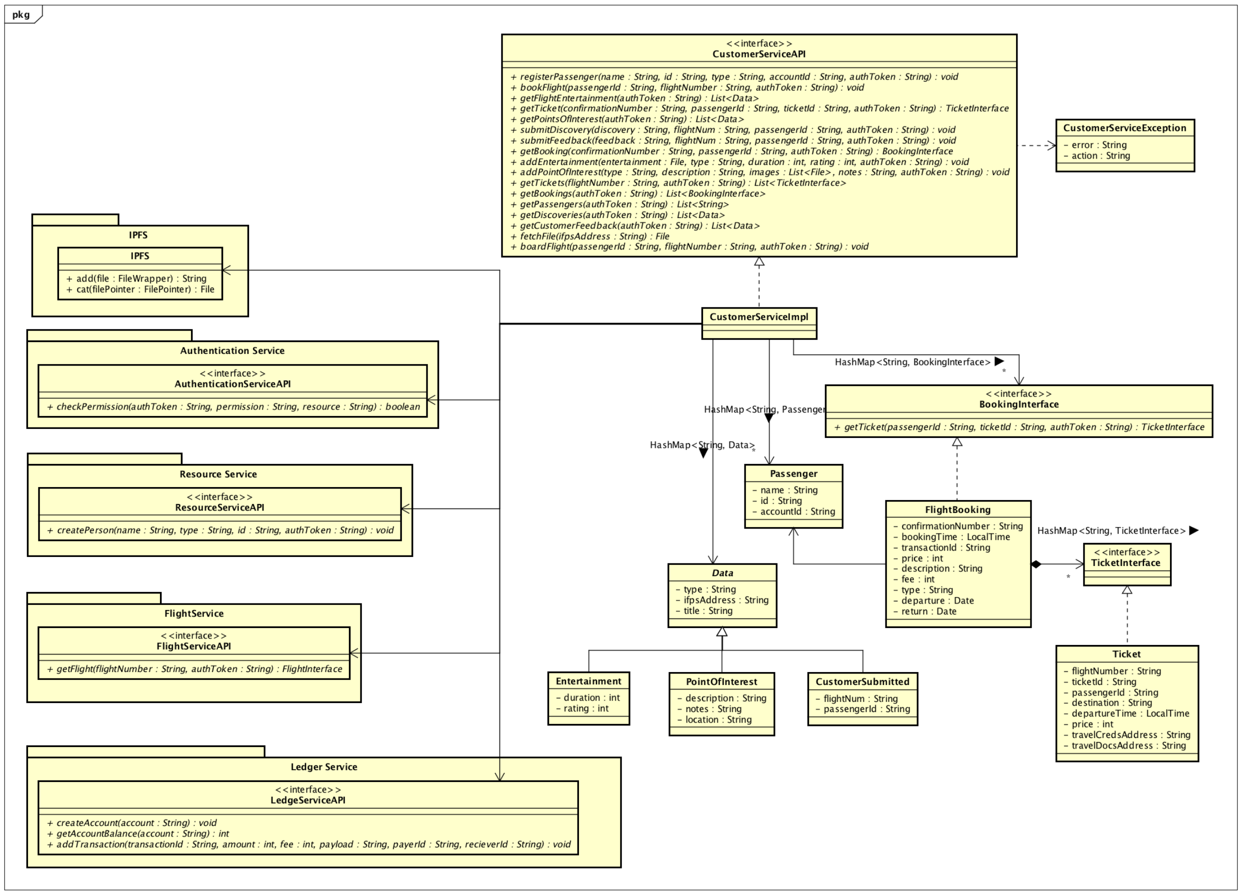
- Passengers
  - Passengers interact with the Customer Service through the Customer GUI
- Administrators
  - Administrators interact with the Customer Service through the Administrator GUI to provide content for passengers and view passenger actions on the Customer GUI.

Use cases

- Book travel
  - Customers may browse flights and book travel through the GUI
- Register
  - Customers must be registered with the ISTS in order to book travel using the Customer GUI
- Board flight
  - Once a customer has booked a flight, they have the authorization to book that flight.
- Watch in-flight entertainment

  - When on flights, customers can view in-flight entertainment such as movies, books, and music.
- Record discoveries
  - While on flights, customers can use the Customer GUI to record discoveries.
- Provide feedback
  - While on flights customers can use the Customer GUI to provide feedback on their flight.
- Review travel documents
  - After booking a flight, customers may log into the Customer GUI and using their confirmation number, review their travel documents.
- Cancel booking
  - Customers may cancel their flight booking if they so desire
- See information about points of interest
  - Passengers can view information and images of points of interest while in-flight
- View registered passengers
  - Administrators may view all registered passengers known to the ISTS through the Admin GUI
- Add in-flight entertainment
  - Administrators must upload entertainment using the Admin GUI in order for customers to have entertainment to view in-flight
- Add points of interest
  - Administrators must upload points of interest using the Admin GUI in order for customers to have points of interest to view in-flight
- View discoveries
  - Administrators may log on and view customer discoveries logged in-flight
- View customer feedback
  - Administrators may log on and view customer feedback provided in-flight
- View flight bookings
  - Administrators can view the passengers booked on a given flight through the Administrator GUI

Class Diagram


pkg
<<interface>> CustomerServiceAPI
+ registerPassenger(name : String, id : String, type : String, accountId : String, authToken : String) : void
+ bookFlight(passengerId : String, flightNumber : String, authToken : String) : void
+ getFlightEntertainment(authToken : String) : List<Data>
+ getTicket(confirmationNumber : String, passengerId : String, ticketId : String, authToken : String) : TicketInterface
+ getPointsOfInterest(authToken : String) : List<Data>
+ submitDiscovery(discovery : String, flightNum : String, passengerId : String, authToken : String) : void
+ submitFeedback(feedback : String, flightNum : String, passengerId : String, authToken : String) : void
+ getBooking(confirmationNumber : String, passengerId : String, authToken : String) : BookingInterface
+ addEntertainment(entertainment : File, type : String, duration : int, rating : int, authToken : String) : void
+ addPointOfInterest(type : String, description : String, images : List<File>, notes : String, authToken : String) : void
+ getTickets(flightNumber : String, authToken : String) : List<TicketInterface>
+ getBookings(authToken : String) : List<BookingInterface>
+ getPassengers(authToken : String) : List<String>
+ getDiscoveries(authToken : String) : List<Data>
+ getCustomerFeedback(authToken : String) : List<Data>
+ fetchFile(ifpsAddress : String) : File
+ boardFlight(passengerId : String, flightNumber : String, authToken : String) : void
CustomerServiceException
- error : String
- action : String
IPFS
IPFS
+ add(file : FileWrapper) : String
+ cat(filePointer : FilePointer) : File
Authentication Service
<<interface>> AuthenticationServiceAPI
+ checkPermission(authToken : String, permission : String, resource : String) : boolean
Resource Service
<<interface>> ResourceServiceAPI
+ createPerson(name : String, type : String, id : String, authToken : String) : void
FlightService
<<interface>> FlightServiceAPI
+ getFlight(flightNumber : String, authToken : String) : FlightInterface
Ledger Service
<<interface>> LedgeServiceAPI
+ createAccount(account : String) : void
+ getAccountBalance(account : String) : int
+ addTransaction(transactionId : String, amount : int, fee : int, payload : String, payerId : String, recieverId : String) : void
CustomerServiceImpl
HashMap<String, BookingInterface>
HashMap<String, Passenger>
HashMap<String, Data>
<<interface>> BookingInterface
+ getTicket(passengerId : String, ticketId : String, authToken : String) : TicketInterface
Passenger
- name : String
- id : String
- accountId : String
FlightBooking
- confirmationNumber : String
- bookingTime : LocalTime
- transactionId : String
- price : int
- description : String
- fee : int
- type : String
- departure : Date
- return : Date
HashMap<String, TicketInterface>
<<interface>> TicketInterface
Data
- type : String
- ifpsAddress : String
- title : String
Entertainment
- duration : int
- rating : int
PointOfInterest
- description : String
- notes : String
- location : String
CustomerSubmitted
- flightNum : String
- passengerId : String
Ticket
- flightNumber : String
- ticketId : String
- passengerId : String
- destination : String
- departureTime : LocalTime
- price : int
- travelCredsAddress : String
- travelDocsAddress : String

Customer Service class diagram: the Customer Service module depends on the IPFS, Authentication Service, Resource Service, Flight Service, and Ledger Service.

Class Dictionary

CustomerServiceImpl

The Customer Service Implementation implements the Customer Service API. The Customer Service API is used by the Admin GUI and the Customer GUI. Customers use the Customer Service API (through the GUI) to book flights, view travel documents, register as passengers, and submit feedback and discoveries once on the flight. Customers can also view points of interest and in-flight entertainment while on flights. Administrators interact with the Customer Service API (through the Admin GUI) to add entertainment, points of interest, view customer-submitted data (discoveries and feedback), and view flight bookings, registered passengers, and tickets.

Associations

Association Name	Type	Description
flightBookingMa	HashMap<String,	A mapping of booking confirmation numbers

p	BookingInterface>	to the appropriate booking interface.
passengerMap	HashMap<String, Passenger>	A mapping of passenger ID to passenger object.
entertainmentMap	HashMap<String, Entertainment>	A mapping of entertainment ID to the entertainment it represents.
pointOfInterestMap	HashMap<String, PointOfInterest>	A mapping of point of interest ID to the point of interest it represents.
customerDataMap	HashMap<String, CustomerSubmitted>	A mapping of customer submission ID to the submission it represents.
ledgerService	LedgerServiceAPI	Reference to the ledger service.
flightService	FlightServiceAPI	Reference to the flight service.
resourceService	ResourceServiceAPI	Reference to the resource service.
authService	AuthenticationServiceAPI	Reference to the authentication service.
ipfs	IPFS	Reference to the IPFS instance used by the ISTS.

Method

Method Name	**Type**	**Description**
registerPassenger	(name : String, id : String, type : String, accountId : String, authToken : String) : void	Register a new passenger. The passenger must have already registered for a ledger account or a CustomerServiceException is thrown. If the passenger is already registered, a CustomerServiceException is thrown.
getPassengers	(authToken : String) : List<String>	Get a list of all currently registered passengers.
bookFlight	(passengerId : String, flightNum : String, authToken : String) : void	The passenger must be registered and have sufficient funds in their ledger account in order to book the flight. If not, a CustomerServiceException is thrown. If they do, a flight booking is created, with as many individual tickets as required (i.e. if the booking is for a round-trip flight). See the

		Sequence Diagram section for more detail on booking a flight.
getFlightEntertainment	(authToken : String) : List<Entertainment>	Returns a list of in-flight entertainment objects that can be viewed on the Customer GUI.
getTicket	(confirmationNumber : String, passengerId : String, ticketId : String, authToken : String) : TicketInterface	Returns a TicketInterface for the specified ticket. If the confirmation number is invalid, or does not correspond to the correct passenger, or the ticket ID is not accurate, a CustomerServiceException is thrown.
getTickets	(flightNumber: String, authToken: String): List<TicketInterface>	Returns a list of TicketInterfaces for all tickets booked on a given flight. If the flight does not exist or has no current passengers, an empty list is returned.
getPointsOfInterest	(authToken : String) : List<PointOfInterest>	Returns a list of points of interests that can be browsed on the Customer GUI.
submitDiscovery	(discovery : String, flightNum : String, passengerId : String, authToken : String) : void	Method called when a customer submits a discovery on a flight. The discovery is saved in the IPFS.
submitFeedback	(feedback : String, flightNum : String, passengerId : String, authToken : String) : void	Method called when a customer submits feedback while on a flight. The feedback is saved in the IPFS.
getBooking	(confirmationNumber : String, passengerId : String, authToken : String) : BookingInterface	Returns the booking corresponding to the confirmation number and passenger ID. If the confirmation number of passenger ID do not exist or do not match, a CustomerServiceException is thrown.
cancelBooking	(confirmationNumber : String, passengerId : String, authToken : String) : void	Cancels the specified booking and the corresponding tickets. If the confirmation number of passenger ID do not exist or do not match, a CustomerServiceException is thrown.
getBookings	(authToken : String) : List<BookingInterface>	Returns a list of booking interfaces for all current bookings saved in the Customer

		Service module. Returns an empty list if there are no bookings.
addEntertainment	(entertainment: File, type : String, duration : int, rating : int, authToken : String) : void	Method called to upload entertainment. The entertainment is saved to the IPFS.
addPointOfInterest	(type : String, description : String, images : List<File>, notes : List<String>, authToken : String) : void	Method called to upload points of interest. The point of interest is saved to the IPFS.
getDiscoveries	(authToken: String): List<CustomerSubmitted>	Returns a list of customer-submitted discoveries, which are retrieved from the IPFS.
getCustomerFeedback	(authToken : String) : List<CustomerSubmitted>	Returns a list of customer feedback, which are retrieved from the IPFS.
fetchFile	(ipfsAddress: String): File	Returns the actual file for any data object. This method is called when a user chooses to view in-flight entertainment, a point of interest, or an administrator wants to view customer feedback or discoveries. The method fetches the data from the IPFS and returns the file to the GUI.
boardFlight	(passengerId : String, flightNumber : String, authToken : String) : void	Method called when a passenger attempts to board a flight. If the passenger is not authorized to board the flight, an PermissionDeniedException is thrown.

Flight Booking

A representation of a flight booking. This class serves as record of a passenger purchasing a trip and an association class between passengers and their tickets for the particular journey. The FlightBooking class also records the transaction ID of the payment, stored in the Ledger Service. The FlightBooking class implements the BookingInterface.

Properties

Property Name	**Type**	**Description**
confirmationNumber	String	The unique confirmation number of the booking.

bookingTime	LocalTime	The time the booking was completed.
transactionId	String	The transaction ID for the ticket purchase stored in the Ledger Service.
price	int	The price payed for the ticket.
description	String	The description of the booking.
fee	int	The fee corresponding to the purchase of the ticket.
type	String	The type of booking: One-Way, Round-Trip, or Guided Tour.
departure	Date	The departure date of the journey.
return	Date	The return date of the journey.

Associations

Association Name	**Type**	**Description**
passenger	Passenger	The passenger that booked the journey.
tickets	List<TicketInterface >	A list of tickets, as interfaces, corresponding to the journey.

Method

Method Name	**Type**	**Description**
getTicket	(passengerId : String, ticketId : String, authToken: String) : TicketInterface	Returns the interface for the specified ticket. If the ticket does not exist, a CustomerServiceException is raised.
getBookingTime	(authToken : String) : LocalTime	Get a list of all currently registered passengers.
getTransactionId	(authToken: String): String	Getter for the booking transaction ID.
getPrice	(authToken: String): String	Getter for the booking price.
getDescripton	(authToken: String): String	Getter for the booking description.
getFee	(authToken: String): String	Getter for the booking fee.

getType	(authToken: String): String	Getter for the booking type.
getDepartureDate	(authToken: String): Date	Getter for the booking departure date.
getReturnDate	(authToken: String): Date	Getter for the booking return date.

Ticket
A representation of a ticket. A ticket is a record that a passenger has a seat on a single flight within a potentially multi-leg journey. The Ticket class implements the TicketInterface.

Properties

Property Name	**Type**	**Description**
flightNumber	String	The unique flight number the ticket corresponds to.
ticketId	String	The unique ID of the ticket.
passegerId	String	The passenger ID of the passenger the ticket is for.
destination	int	The destination of the flight.
departureTime	String	The departure time of the flight.
price	int	The price of the flight.
travelCredsAddress	String	The IPFS address where the passenger's relevant travel credentials (visas, passports) are stored.
travelDocsAddress	String	The IPFS address where the passengers relevant travel documents (boarding pass, information package) are stored.

Method

Method Name	**Type**	**Description**
getFlightNumber	(authToken: String): String	Getter for the flight number.
getTicketId	(authToken: String): String	Getter for the ticket ID.
getPassengerId	(authToken: String): String	Getter for the passenger ID.

getDestination	(authToken: String): String	Getter for the flight destination.
getDepartureTime	(authToken: String): LocalTime	Getter for the flight departure time.
getPrice	(authToken: String): int	Getter for the price of the ticket.
getTravelCredsAddress	(authToken: String): String	Getter for the IPFS address of the passenger's travel credentials (visas, passports).
getTravelDocsAddress	(authToken: String): String	Getter for the IPFS address of the passenger's travel documents (boarding pass, information package).

Passenger
A representation of a passenger known to the ISTS. The Passenger class has a unique ID for the passenger and the passenger's unique Ledger account ID.

Properties

Property Name	**Type**	**Description**
name	String	The full name of the passenger.
id	String	The unique ID of the passenger.
accountId	String	The Ledger ID of the passenger.

Data
A representation of a Data object that is stored in the IPFS and accessed through the Admin GUI or Customer GUI. The Data class is abstract.

Properties

Property Name	**Type**	**Description**
type	String	The type of data stored, which takes different values depending on the type of data.
ipfsAddress	String	The address within the IPFS where the data is stored.
title	String	The identifying title or name of the data.

Entertainment

The Entertainment class extends the abstract Data class. The class represents in-flight entertainment such as books, movies, and music. The content of the entertainment is stored in the IPFS.

Properties

Property Name	Type	Description
type	String	Type of entertainment: Book, Movie, or Music.
ipfsAddress	String	The address within the IPFS where the entertainment is stored.
title	String	Name of the entertainment.

PointOfInterest

The PointOfInterest class extends the Data abstract class. A point of interest contains a description and images, which are stored in the IPFS.

Properties

Property Name	Type	Description
type	String	The type of point of interest: Planet, Moon, Asteroids, Solar System, or Space Station.
ipfsAddress	String	The address within the IPFS where the images corresponding to the point of interest are stored.
title	String	Name of the point of interest.
description	String	Brief description of the point of interest.
notes	String	Notes on the point of interest.

CustomerSubmitted

The CustomerSubmitted class extends the Data abstract class and represents customer-submitted data, such as discoveries and customer feedback. The content of the submission is stored in the IPFS.

Properties

Property Name	Type	Description
type	String	The type of customer submitted data, either Discovery or Feedback.
ipfsAddress	String	The address within the IPFS where the content of customer submission is stored, along with the passenger's ID and the number of the flight they were on.
title	String	The subject of the customer submitted data.
flightNum	String	The flight the passenger was on when they submitted the data.
passengerId	String	The ID of the passenger submitting the feedback or discovery.

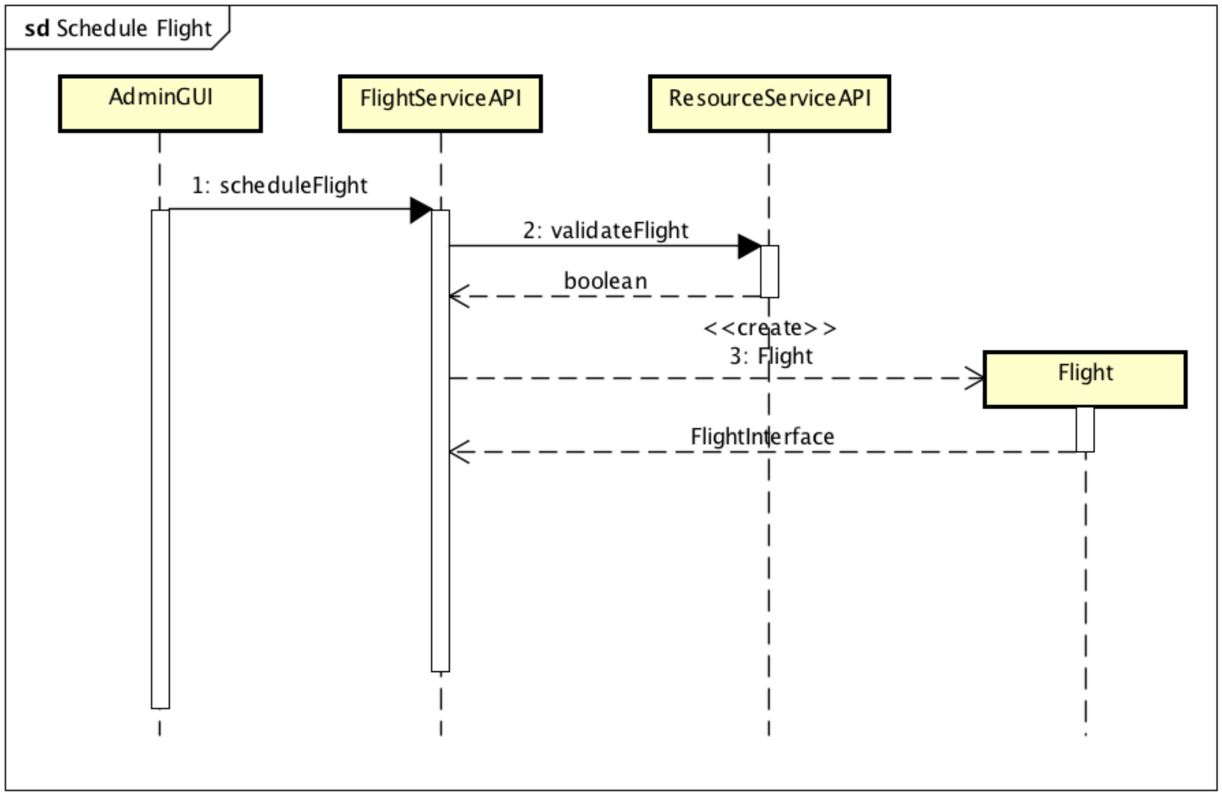
CustomerServiceException
Exception class for the Customer Service module.

Properties

Property Name	Type	Description
error	String	The error raised.
action	String	The action that raised the error.

V. Sequence Diagrams

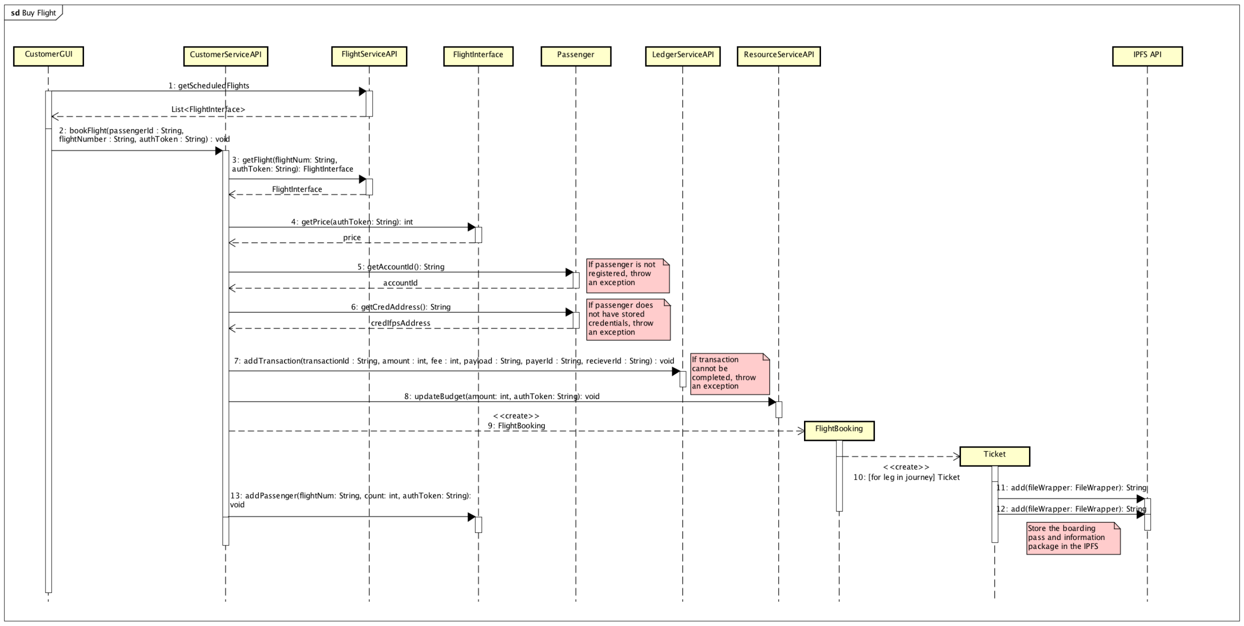
Schedule Flight



Sequence diagram: the sequence of events when an Admin schedules a new flight through the Admin GUI

Flights are scheduled by administrators through the Admin GUI. Admins can input the desired details of the flight, which is then passed to the Flight Service API on the backend. The Flight Service uses the Resource Service API to validate that the flight specifications are valid (i.e. there is sufficient fuel, the space craft exists). If the flight is valid, a new Flisght object is created and a Flight Interface is added as an association of the Flight Service API implementation.

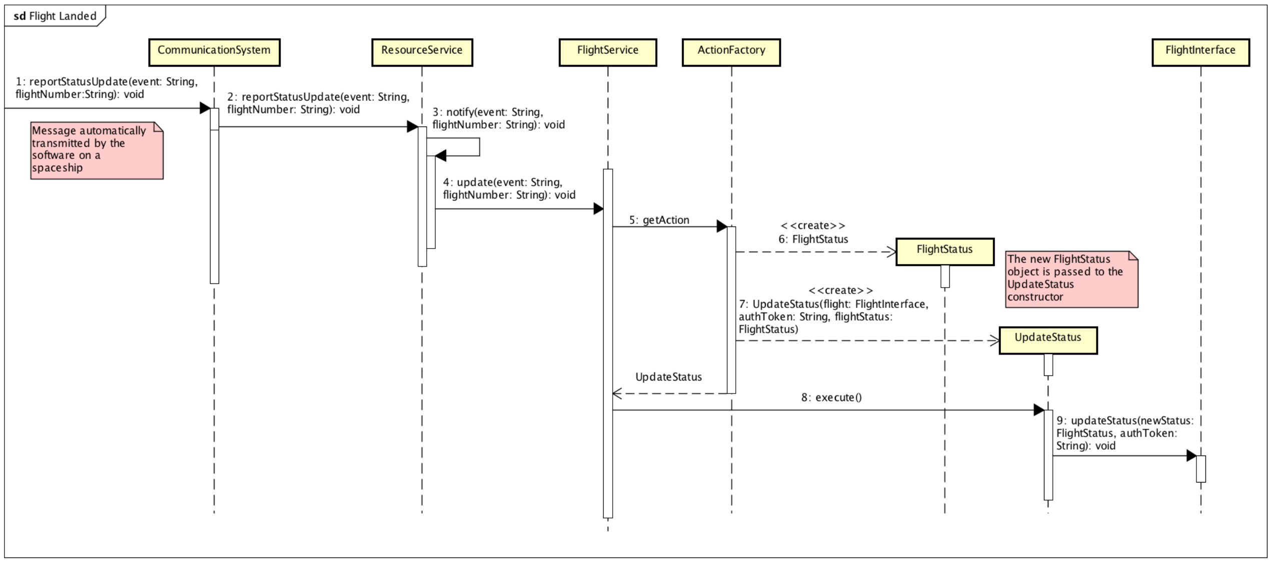
Buy Flight


sd Buy Flight
CustomerGUI
CustomerServiceAPI
FlightServiceAPI
FlightInterface
Passenger
LedgerServiceAPI
ResourceServiceAPI
IPFS API
1: getScheduledFlights
List<FlightInterface>
2: bookFlight(passengerId : String, flightNumber : String, authToken : String) : void
3: getFlight(flightNum: String, authToken: String): FlightInterface
FlightInterface
4: getPrice(authToken: String): int
price
5: getAccountId(): String
accountId
If passenger is not registered, throw an exception
6: getCredAddress(): String
credIfpsAddress
If passenger does not have stored credentials, throw an exception
7: addTransaction(transactionId : String, amount : int, fee : int, payload : String, payerId : String, recieverId : String) : void
If transaction cannot be completed, throw an exception
8: updateBudget(amount: int, authToken: String): void
<<create>>
9: FlightBooking
FlightBooking
<<create>>
10: [for leg in journey] Ticket
Ticket
11: add(fileWrapper: FileWrapper): String
12: add(fileWrapper: FileWrapper): String
Store the boarding pass and information package in the IPFS
13: addPassenger(flightNum: String, count: int, authToken: String): void

Sequence diagram: sequence of events when a customer books a flight through the Customer GUI

- Customers may book travel through the Customer GUI. Customers must be logged in and known to the Customer Service module in order to access the GUI. Customers browse flights, which are provided to the frontend by a call to the Flight Service API.
- When a Customer selects a flight, the backend calls the Customer Service API and attempts to book the flight. First, a transaction is created for the purchase of the flight.
- If the transaction is successful, a record of the flight booking is created. The Flight Booking object itself creates tickets for the travel (i.e. if there are multiple stops or if the booking is round-trip). The tickets are stored in the IPFS.
- The Flight Booking object has an association with the Ticket Interfaces for each ticket. The Customer Service implementation has an association with the Booking Interface.
- If the booking is successful, the Customer GUI increments the number of passengers on the selected flight by calling the Flight Serivce API.

Flight Landed

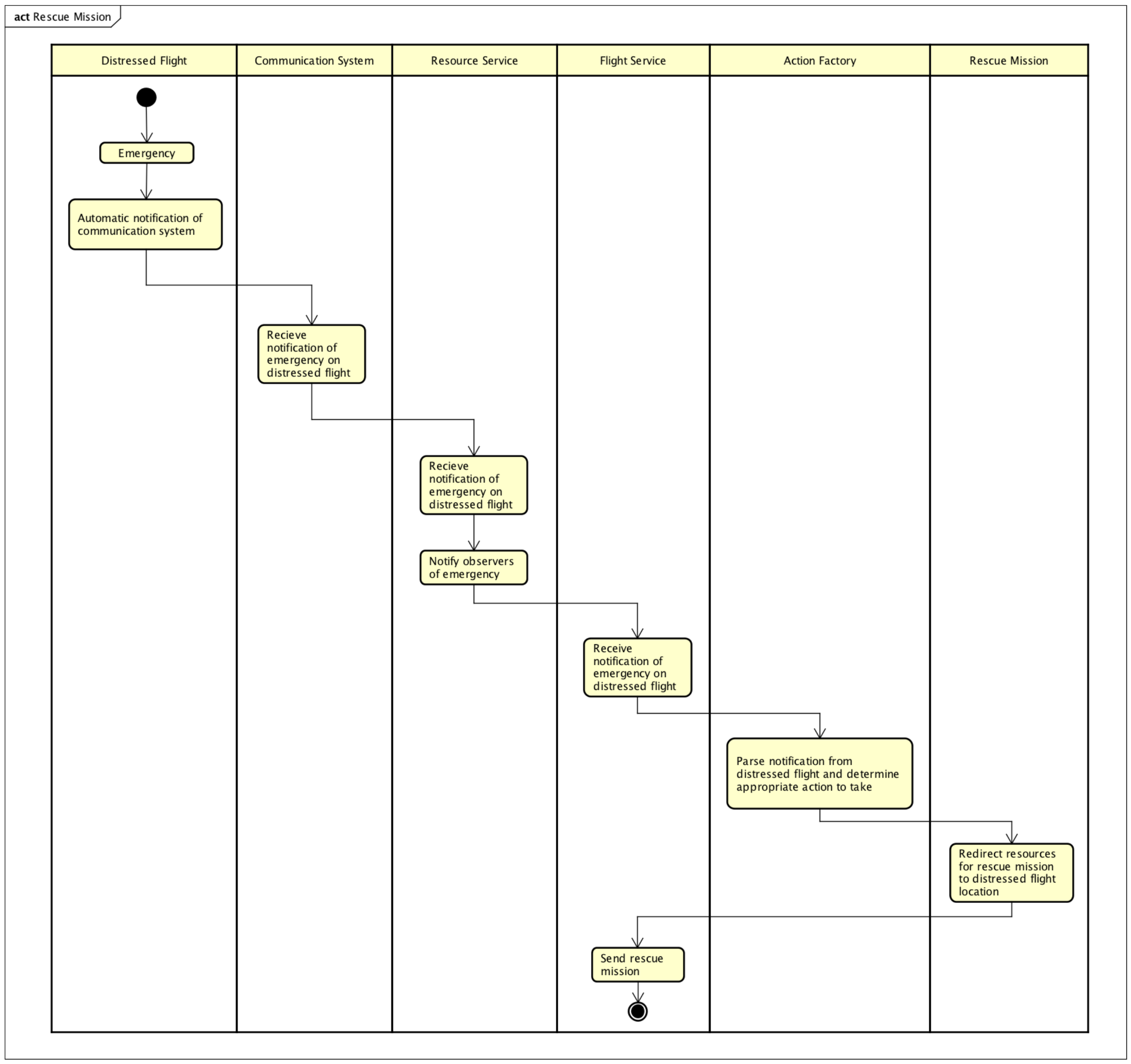

sd Flight Landed
CommunicationSystem
ResourceService
FlightService
ActionFactory
FlightInterface
1: reportStatusUpdate(event: String, flightNumber:String): void
Message automatically transmitted by the software on a spaceship
2: reportStatusUpdate(event: String, flightNumber: String): void
3: notify(event: String, flightNumber: String): void
4: update(event: String, flightNumber: String): void
5: getAction
<<create>>
6: FlightStatus
FlightStatus
The new FlightStatus object is passed to the UpdateStatus constructor
<<create>>
7: UpdateStatus(flight: FlightInterface, authToken: String, flightStatus: FlightStatus)
UpdateStatus
UpdateStatus
8: execute()
9: updateStatus(newStatus: FlightStatus, authToken: String): void

Sequence diagram: a sequence of events when a flight arrives at its destination

- When a flight lands, a message is sent from the software on the spaceship to the ISTS computer/communication system. Upon receiving the notification, the Communication System reports an event to the Resource Service.
- The Resource Service, a Subject, updates its Observers of the event.
- The Flight Service has registered interest in the Resource Service and therefore receives the notification that the flight has landed.
- The notification is passed off to the Action Factory to be parsed.
- The Action Factory parses the notification content and determines that a flight has an updated status. The Action Factory creates a new Flight Status and passes it to a new Update Status object, along with the interface for the flight in question. The Update Status object is returned to the Flight Service.
- The Flight Service executes the Update Status command.
- The Update Status command uses the update status command on the Flight Interface to provide a new status to the flight in question.

VI. Activity Diagram

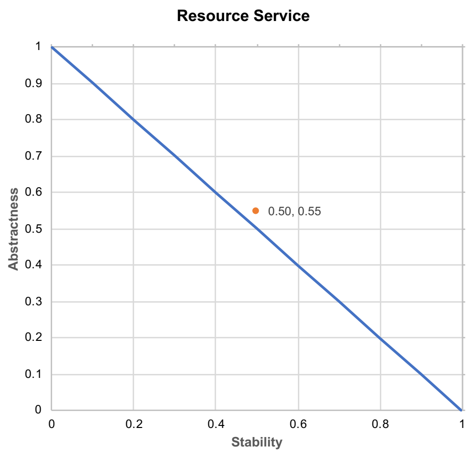
Rescue Mission


act Rescue Mission
Distressed Flight
Communication System
Resource Service
Flight Service
Action Factory
Rescue Mission
Emergency
Automatic notification of communication system
Recieve notification of emergency on distressed flight
Recieve notification of emergency on distressed flight
Notify observers of emergency
Receive notification of emergency on distressed flight
Parse notification from distressed flight and determine appropriate action to take
Redirect resources for rescue mission to distressed flight location
Send rescue mission

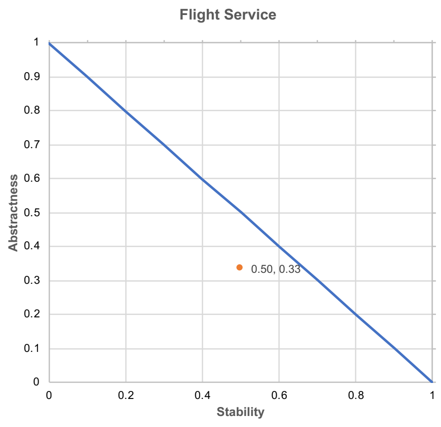
Rescue mission activity diagram: the emergency sequence is triggered by a distressed flight's software and travels through the resource service and the flight service, which deploys a rescue mission.

VII. Instability and Abstractness Metrics

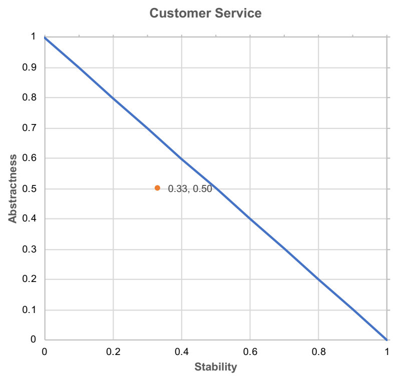
The instability of a module is defined as the ratio of dependencies to dependent modules plus dependencies. The abstractness of a module is the ratio of abstract classes to concrete classes. These two metrics can be plotted as coordinates and can be used to evaluate a module. See below for instability vs. abstractness charts for the Resource Service, Flight Service, and Customer Service.



Resource Service chart: stability vs. abstractness



Flight service chart: stability vs. abstractness



Customer service chart: stability vs. abstractness

The Resource Service module's metrics are very near the main line. If there was one more concrete class defined in the design, the metrics would both be 0.5. This means that the module has a reasonable ratio of abstract to concrete classes and a reasonable ratio of dependencies vs. dependent modules.

The Flight Service module’s metrics suggest that it is not abstract enough given its stability, though the metrics are still reasonable. This makes intuitive sense given that the Flight Service has an equal number of dependent vs. dependent-on modules (3 each) and therefore should be fairly abstract. The Resource Service, a similarly stable (2 dependent and 2 dependent-on modules) module, has a higher abstractness metric.

The Customer Service module’s metrics similarly suggest that it is not abstract enough given its instability. The Customer Service module is near the top-level of the component diagram – it follows that it should be more abstract to compensate.

VIII. Exceptions, Testing, and Risk

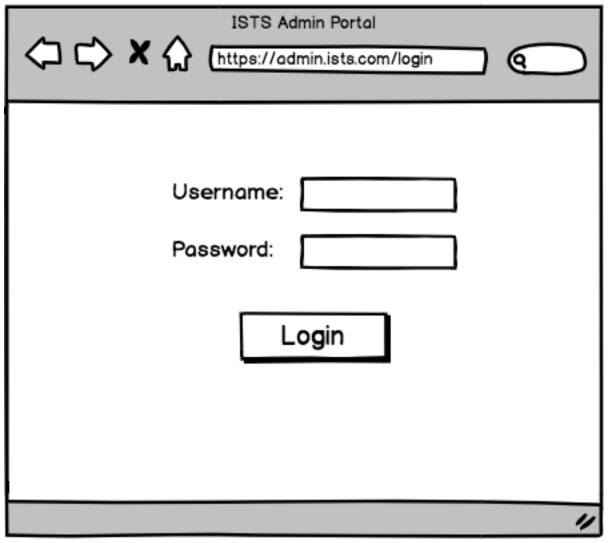
Each module described above has its own exception class: ResourceServiceException, FlightServiceException, and CustomerServiceException. All three exception classes have two properties: error and action. The error records the exception raised, and the action property details the action the user attempted that raised the error. Exceptions are thrown whenever a user inputs something correct to either the Admin or Customer GUI. See class dictionaries for specific circumstances that raise errors for each module.

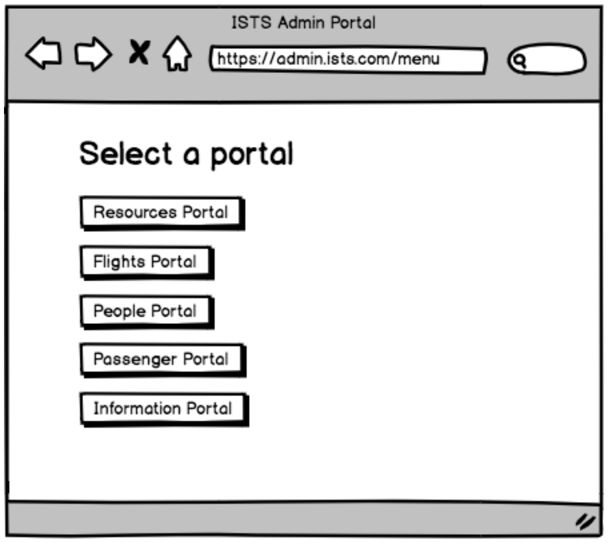
All three modules have an API that could be tested using a Command Processor. It is recommended to develop a programmatic method for testing these modules’ APIs before integrating with the UI. Each module should be deployed as a microservice with replicas to ensure redundancy and a separation of the modules. The modules can communicate with each other’s APIs over a network and should be implemented in such a way as to be able to function as expected when run as microservices.

There are a few areas of risk to be aware of. First, the resources are currently not stored in any permanent file storage. It would be less risky to save records of the resources in the IPFS or a database on earth. Second, the communication and computer systems are singular points of failure within the system. If either of these systems fail, or the network connection fails, the spaceships are without communication with any service and cannot receive help if necessary. This is a significant risk to the safety of the ISTS.

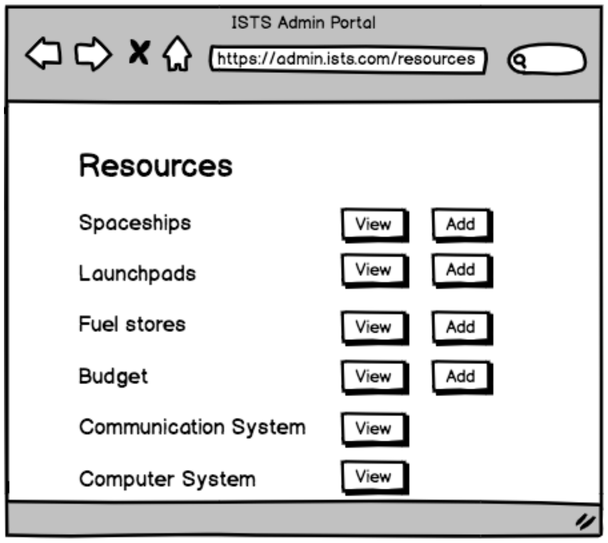
VIV. GUIs

Administrator GUI

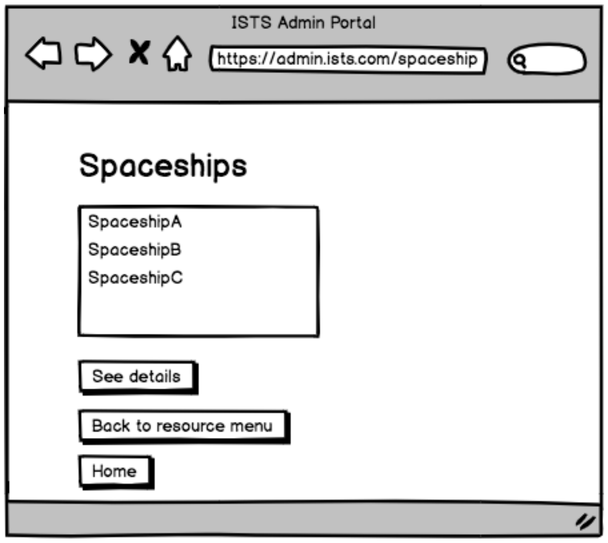


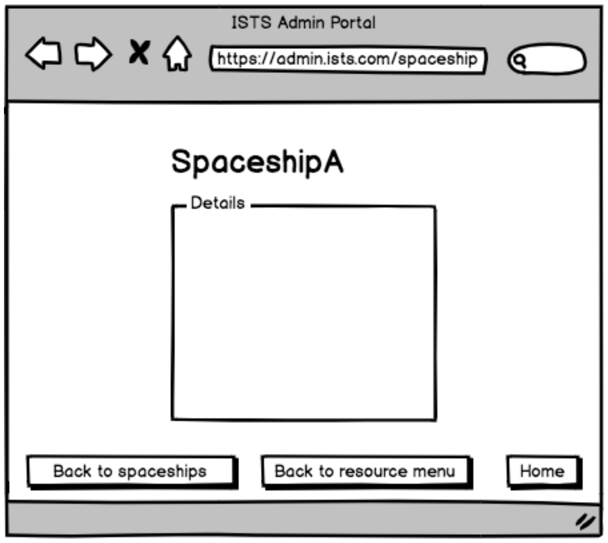


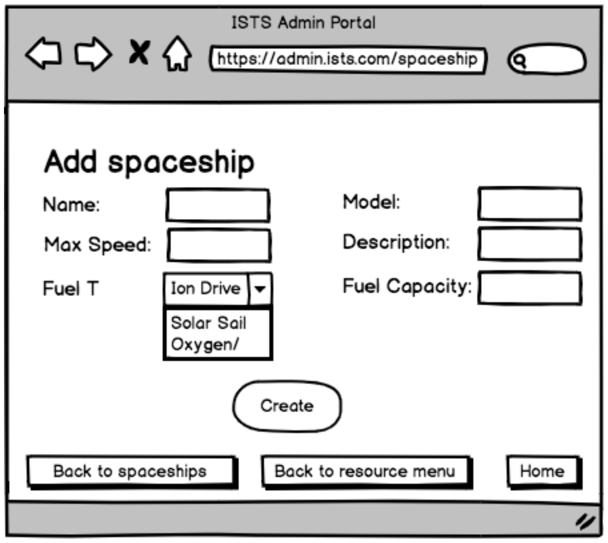
Resources Flow



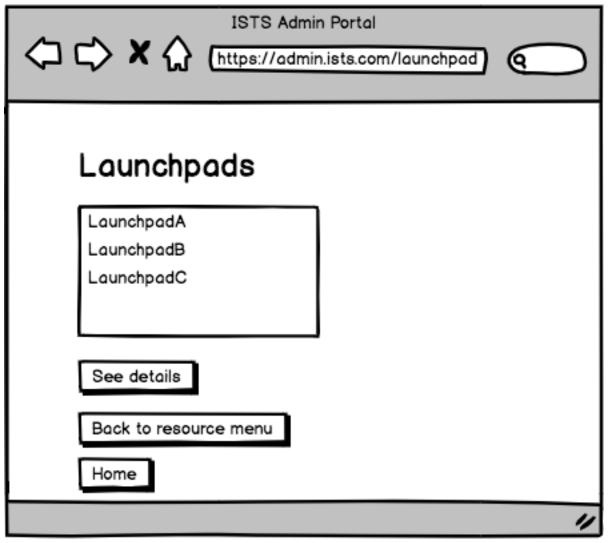
Spaceships

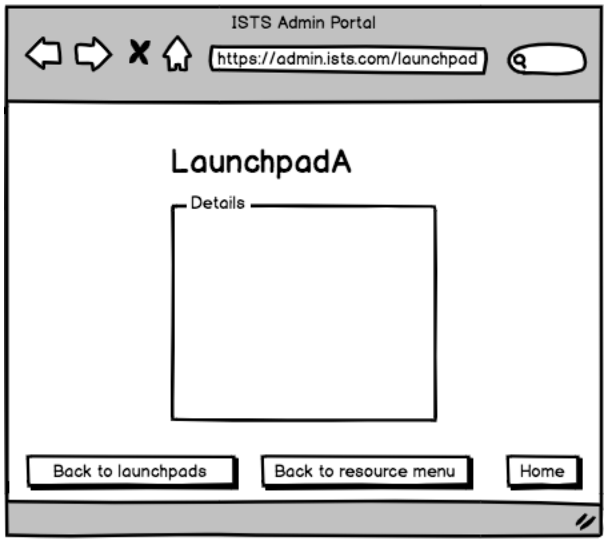

ISTS Admin Portal
https://admin.ists.com/spaceship
Spaceships
SpaceshipA
SpaceshipB
SpaceshipC
See details
Back to resource menu
Home

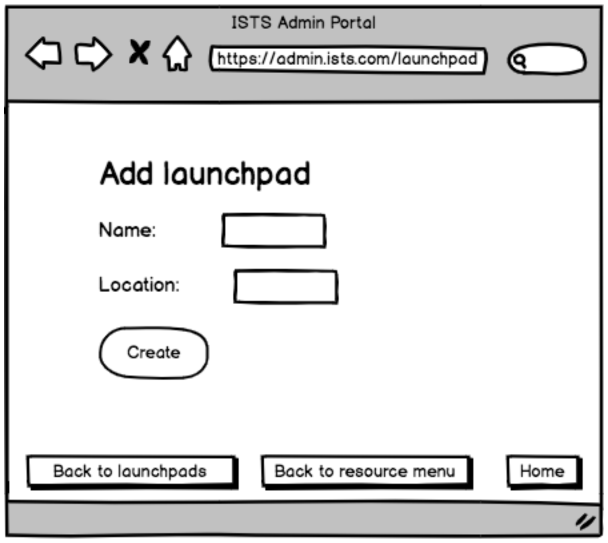

ISTS Admin Portal
https://admin.ists.com/spaceship
SpaceshipA
Details
Back to spaceships
Back to resource menu
Home


ISTS Admin Portal
https://admin.ists.com/spaceship
Add spaceship
Name:
Model:
Max Speed:
Description:
Fuel T
Ion Drive
Solar Sail
Oxygen/
Fuel Capacity:
Create
Back to spaceships
Back to resource menu
Home

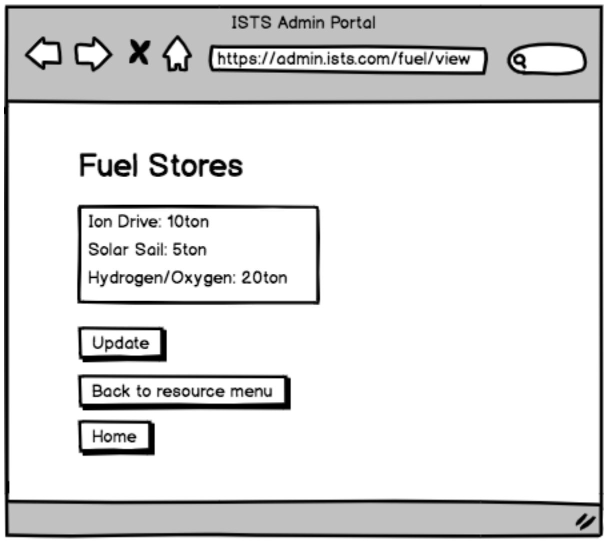
Launchpads

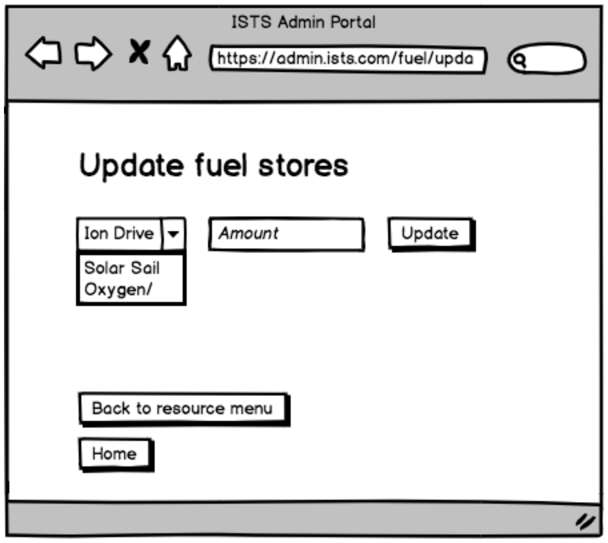

ISTS Admin Portal
https://admin.ists.com/launchpad
Launchpads
LaunchpadA
LaunchpadB
LaunchpadC
See details
Back to resource menu
Home


ISTS Admin Portal
https://admin.ists.com/launchpad
LaunchpadA
Details
Back to launchpads
Back to resource menu
Home

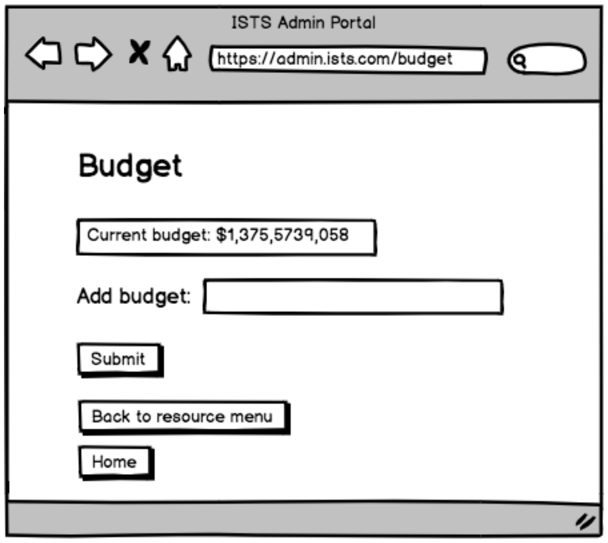

ISTS Admin Portal
https://admin.ists.com/launchpad
Add launchpad
Name:
Location:
Create
Back to launchpads
Back to resource menu
Home

Fuel stores

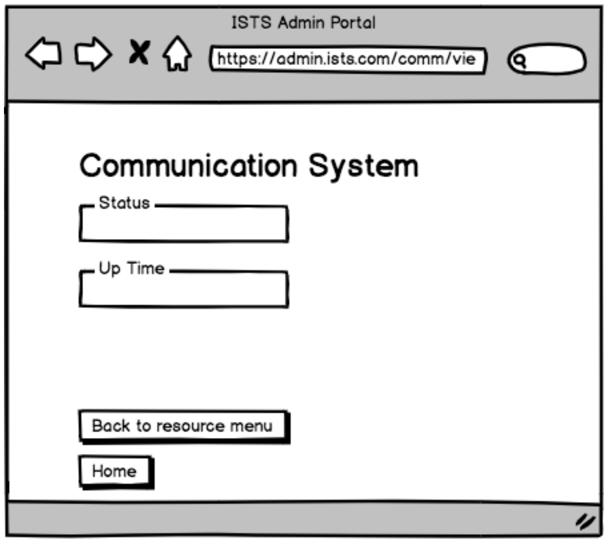

ISTS Admin Portal
https://admin.ists.com/fuel/view
Fuel Stores
Ion Drive: 10ton
Solar Sail: 5ton
Hydrogen/Oxygen: 20ton
Update
Back to resource menu
Home



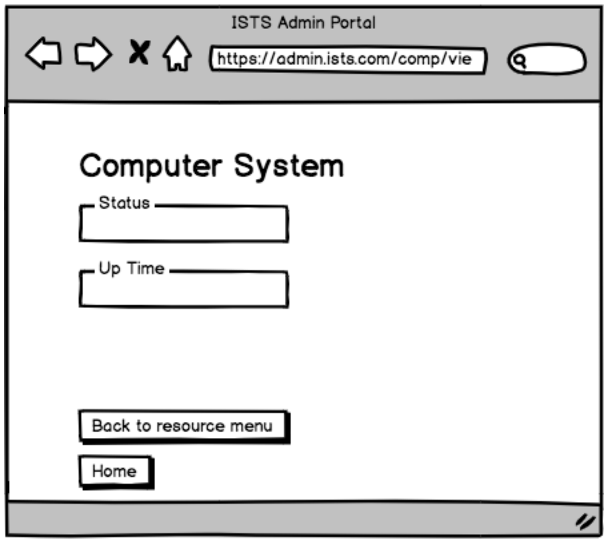
Budget



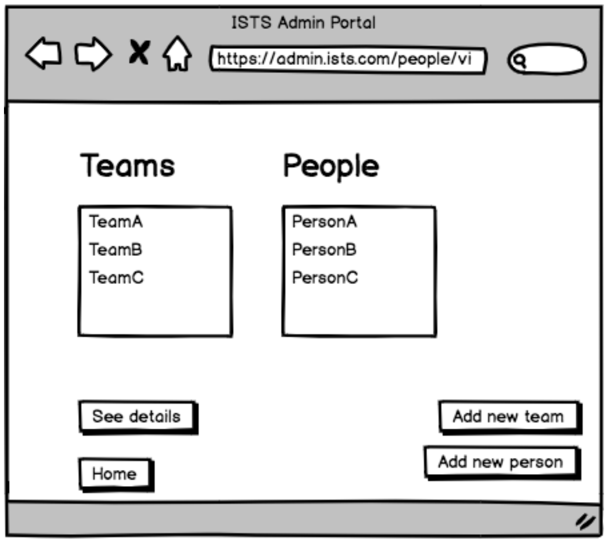
Communication System

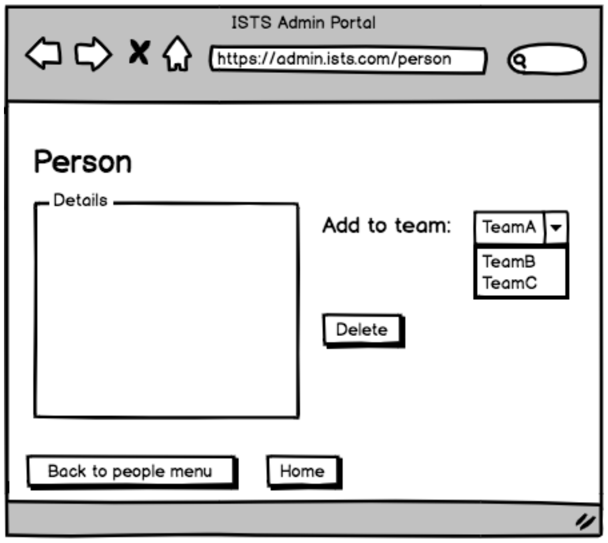


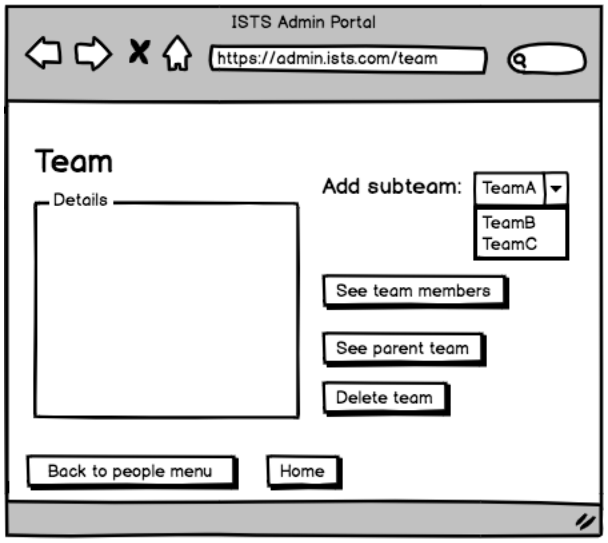
Computer System

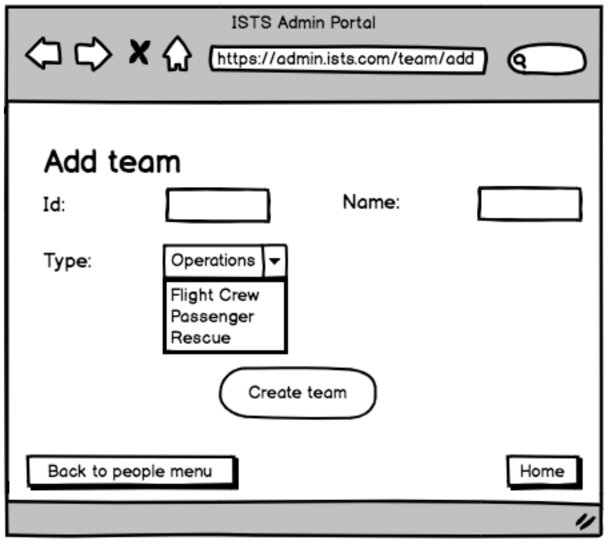


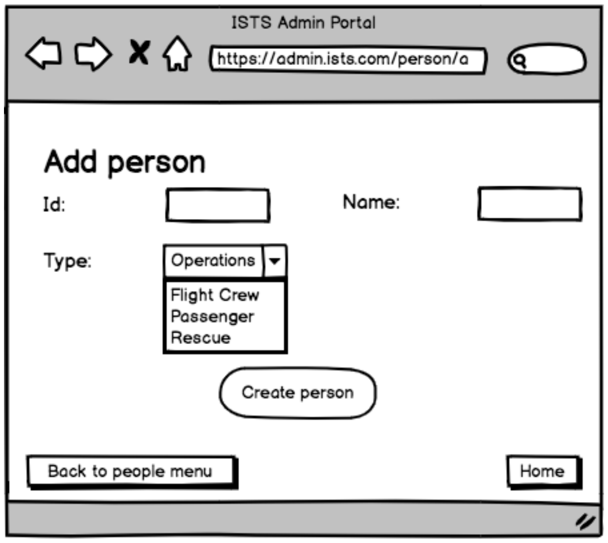
People Portal


ISTS Admin Portal
https://admin.ists.com/people/vi
Teams
People
TeamA
TeamB
TeamC
PersonA
PersonB
PersonC
See details
Add new team
Home
Add new person

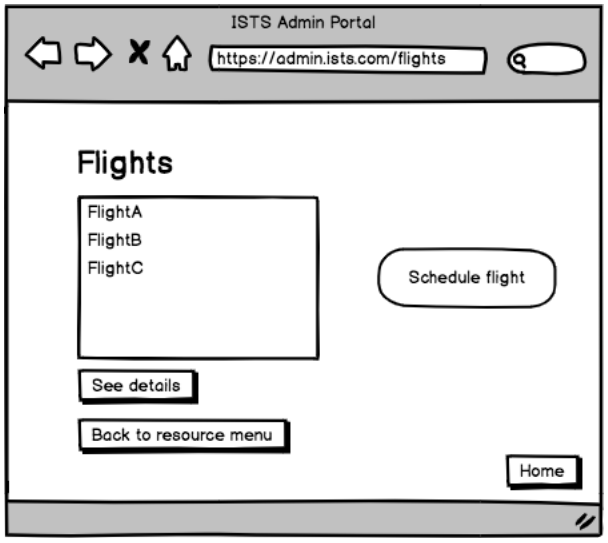

ISTS Admin Portal
https://admin.ists.com/person
Person
Details
Add to team:
TeamA
TeamB
TeamC
Delete
Back to people menu
Home

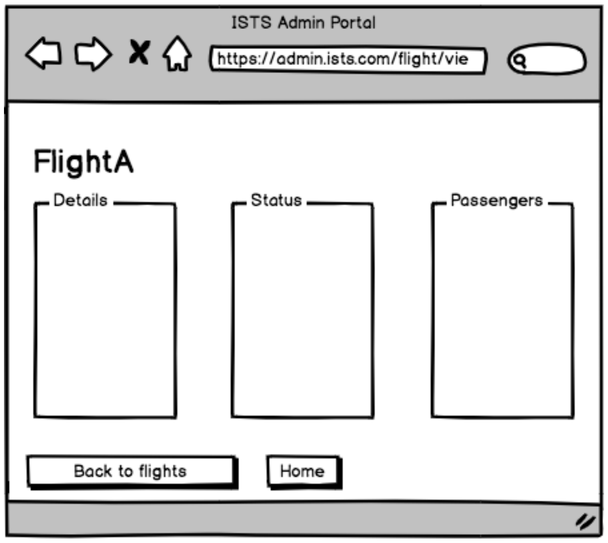

ISTS Admin Portal
https://admin.ists.com/team
Team
Details
Add subteam:
TeamA
TeamB
TeamC
See team members
See parent team
Delete team
Back to people menu
Home

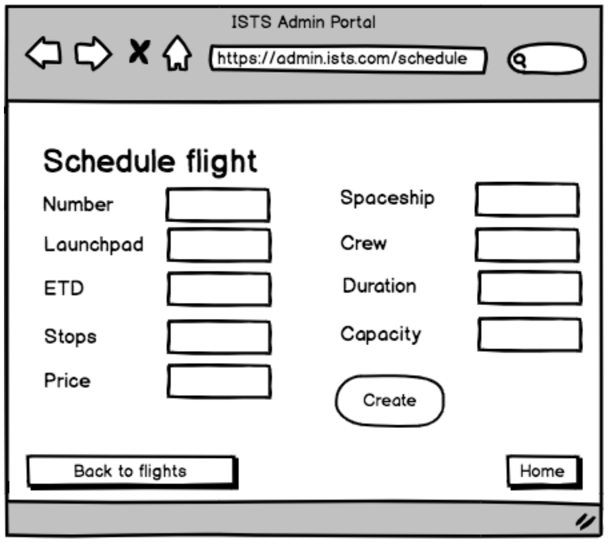




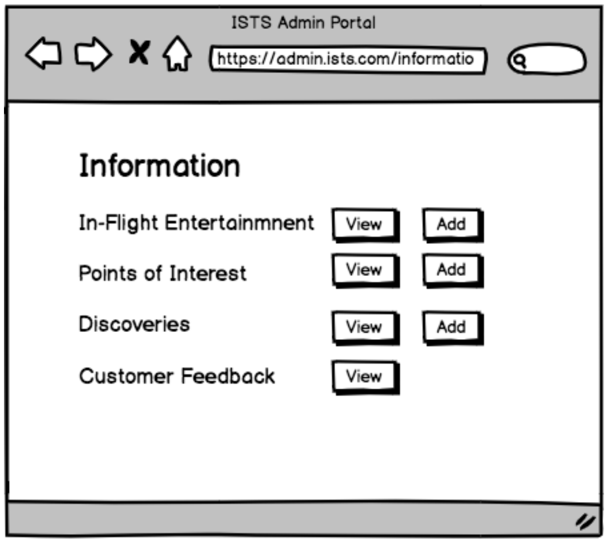
Flight Portal

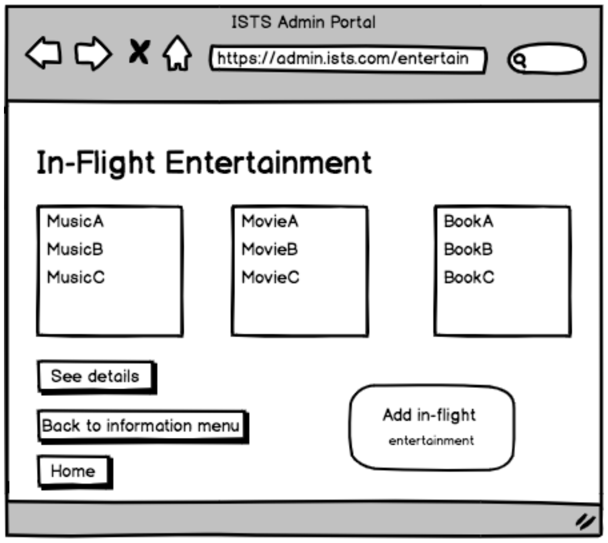

ISTS Admin Portal
https://admin.ists.com/flights
Flights
FlightA
FlightB
FlightC
Schedule flight
See details
Back to resource menu
Home

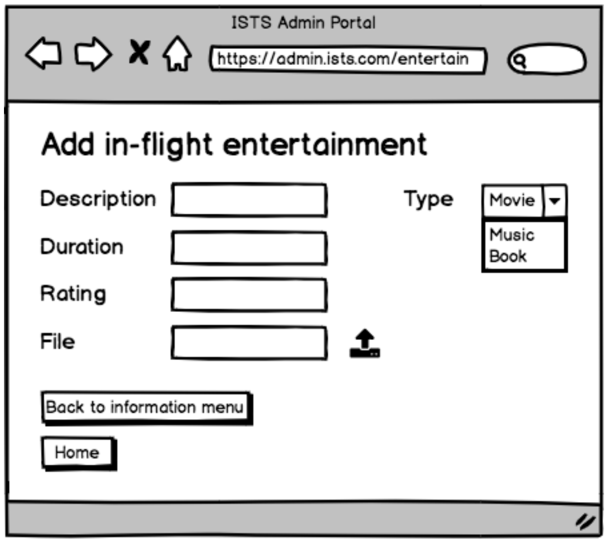

ISTS Admin Portal
https://admin.ists.com/flight/vie
FlightA
Details
Status
Passengers
Back to flights
Home

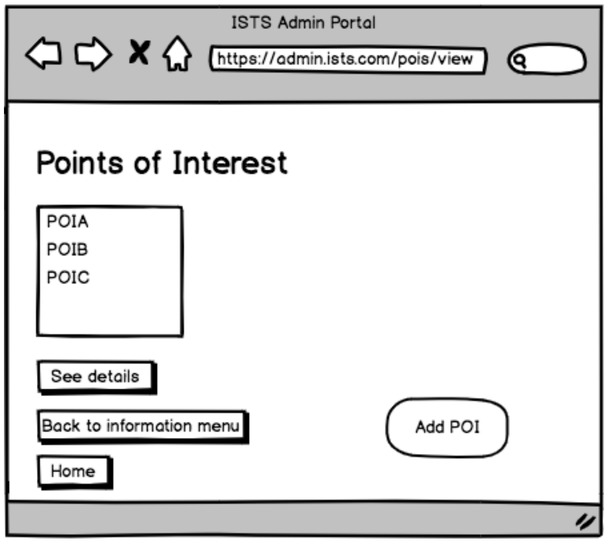

ISTS Admin Portal
https://admin.ists.com/schedule
Schedule flight
Number
Launchpad
ETD
Stops
Price
Spaceship
Crew
Duration
Capacity
Create
Back to flights
Home

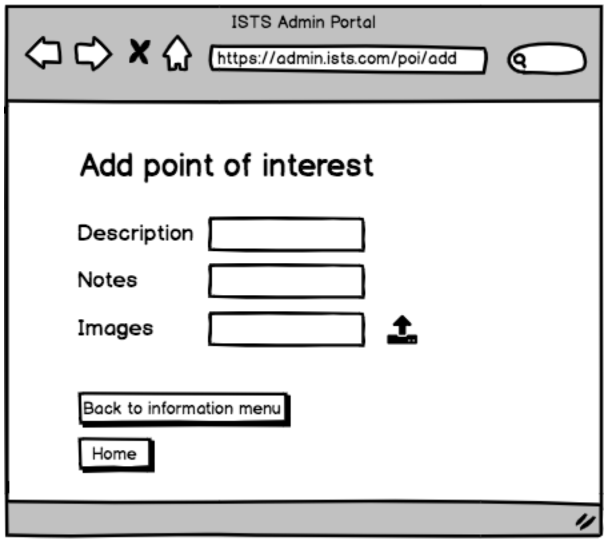
Information Portal

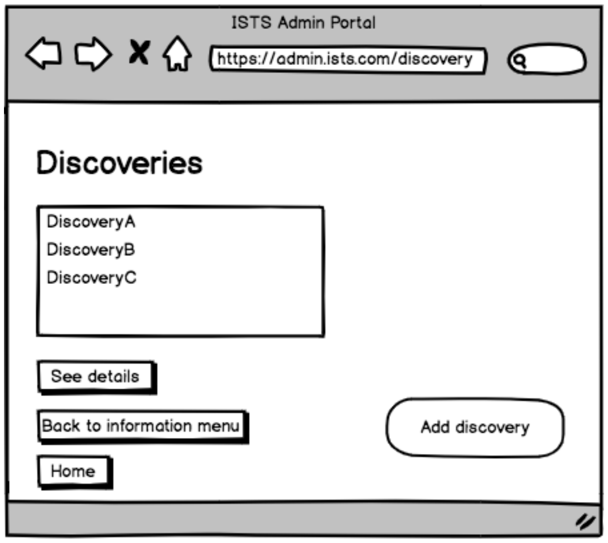


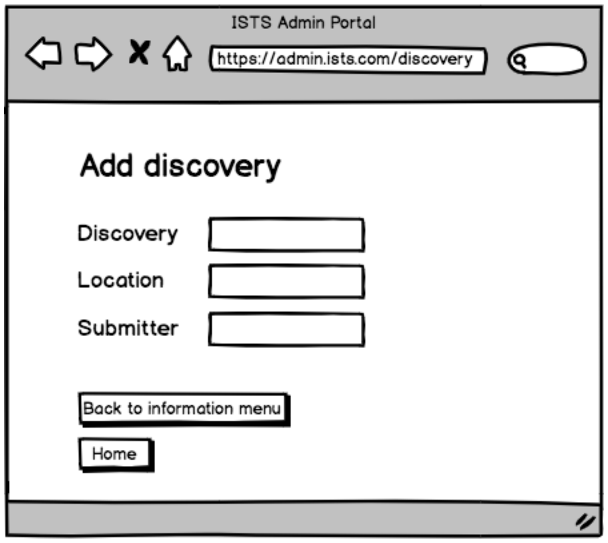


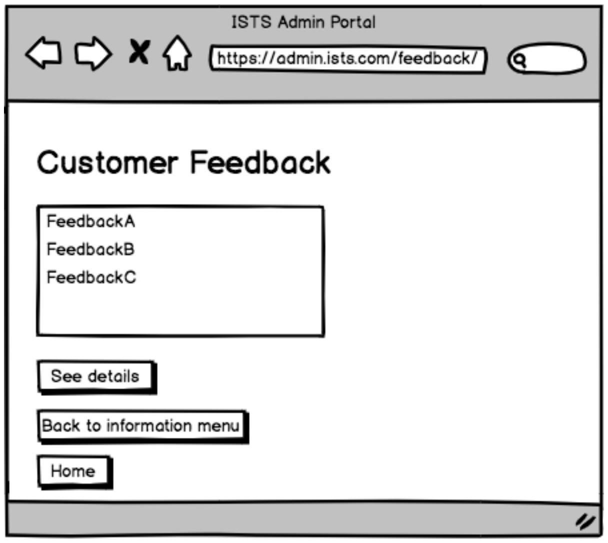

ISTS Admin Portal
https://admin.ists.com/entertain
Add in-flight entertainment
Description
Duration
Rating
File
Type
Movie
Music
Book
Back to information menu
Home


ISTS Admin Portal
https://admin.ists.com/pois/view
Points of Interest
POIA
POIB
POIC
See details
Back to information menu
Home
Add POI

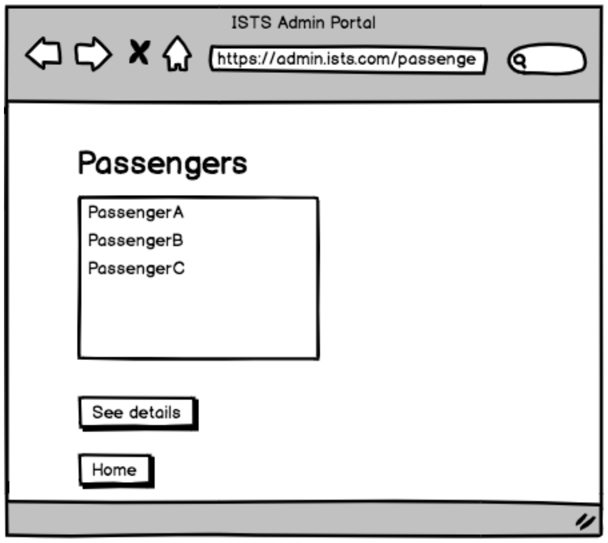

ISTS Admin Portal
https://admin.ists.com/poi/add
Add point of interest
Description
Notes
Images
Back to information menu
Home

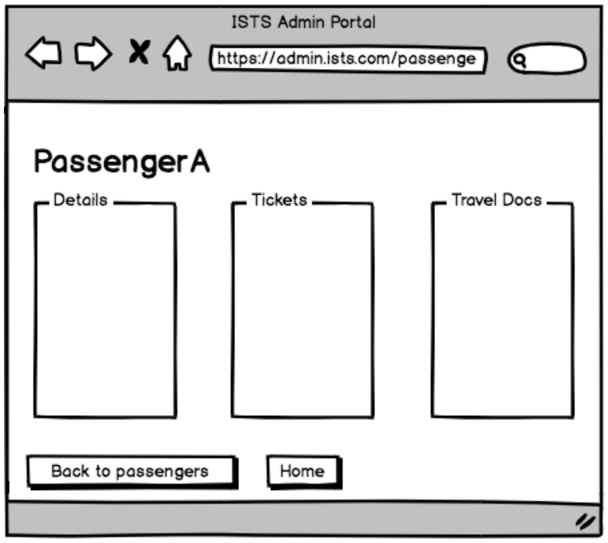

ISTS Admin Portal
https://admin.ists.com/discovery
Discoveries
DiscoveryA
DiscoveryB
DiscoveryC
See details
Back to information menu
Add discovery
Home


ISTS Admin Portal
https://admin.ists.com/discovery
Add discovery
Discovery
Location
Submitter
Back to information menu
Home


ISTS Admin Portal
https://admin.ists.com/feedback/
Customer Feedback
FeedbackA
FeedbackB
FeedbackC
See details
Back to information menu
Home

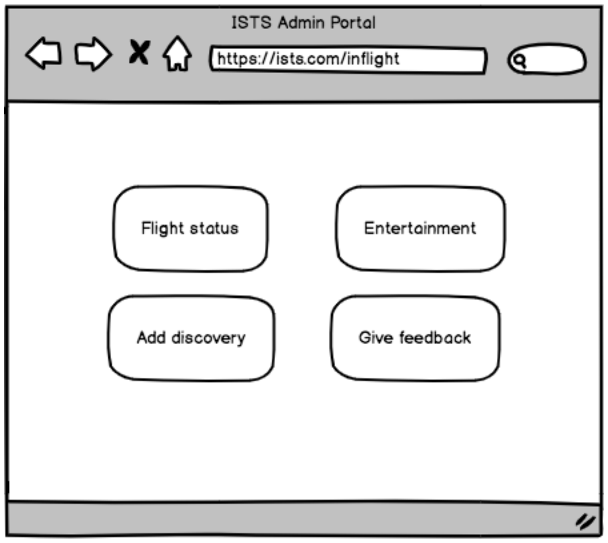
Passenger Portal

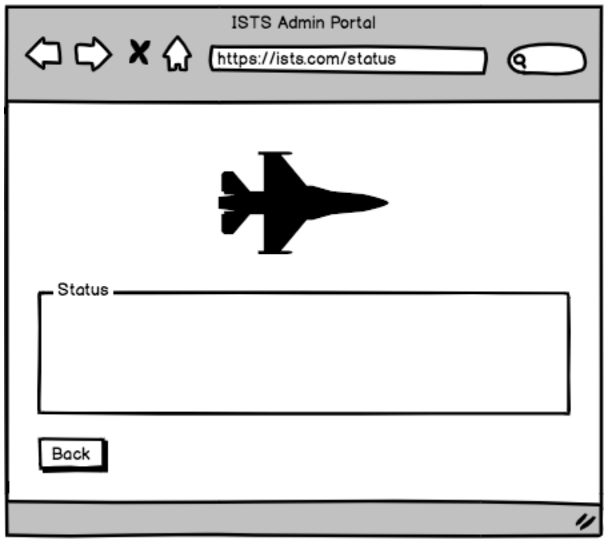


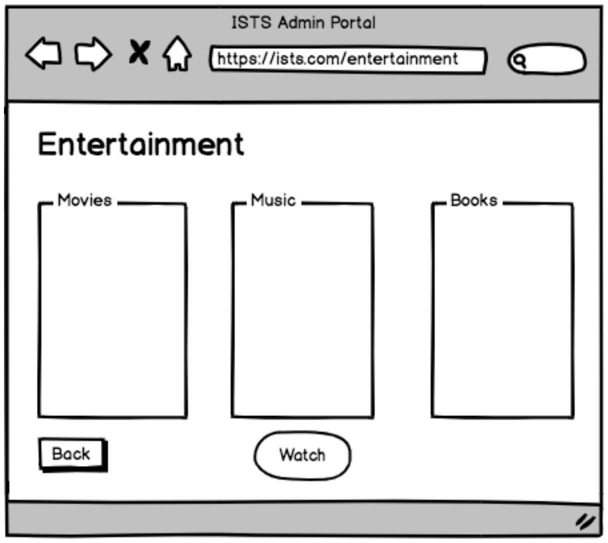


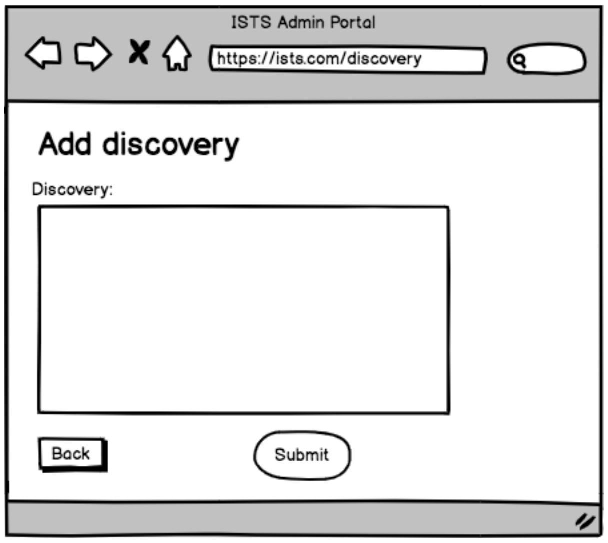
Customer GUI

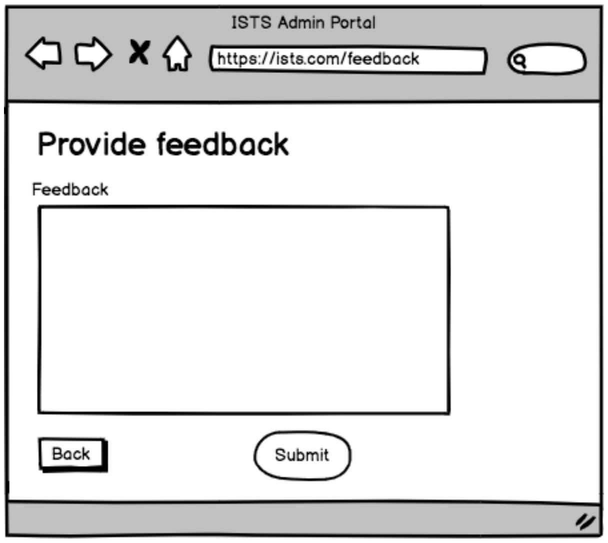
In-flight view


ISTS Admin Portal
https://ists.com/inflight
Flight status
Entertainment
Add discovery
Give feedback

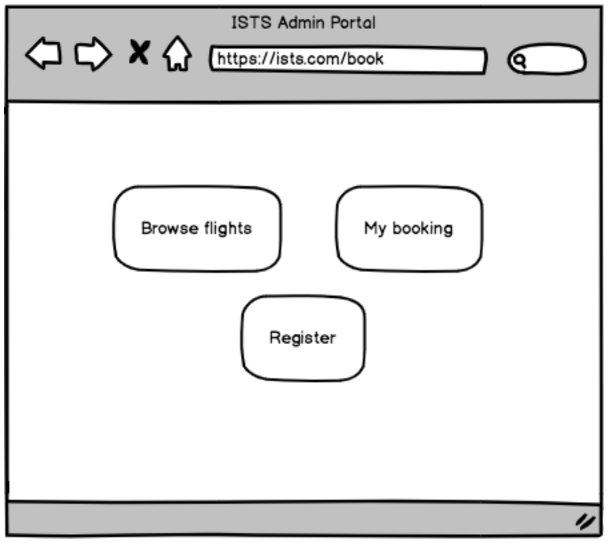

ISTS Admin Portal
https://ists.com/status
Status
Back

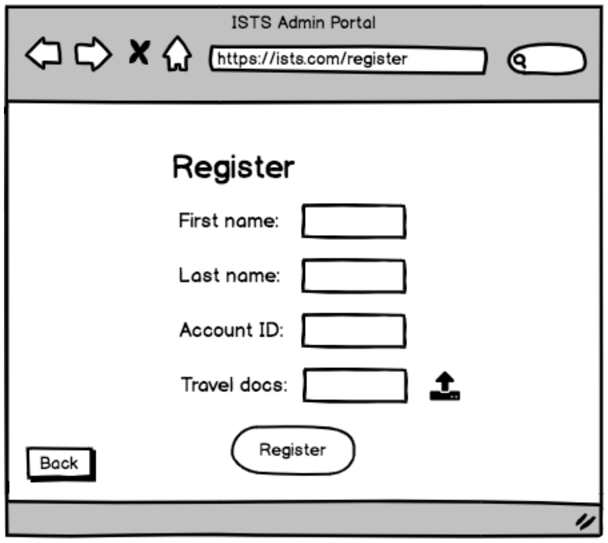

ISTS Admin Portal
https://ists.com/entertainment
Entertainment
Movies
Music
Books
Back
Watch

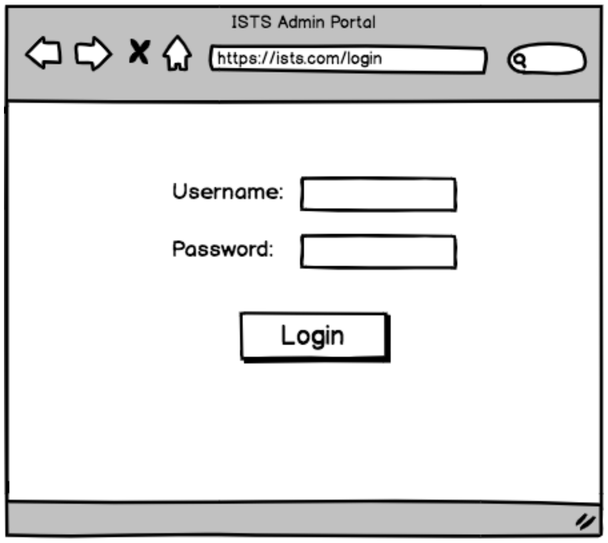


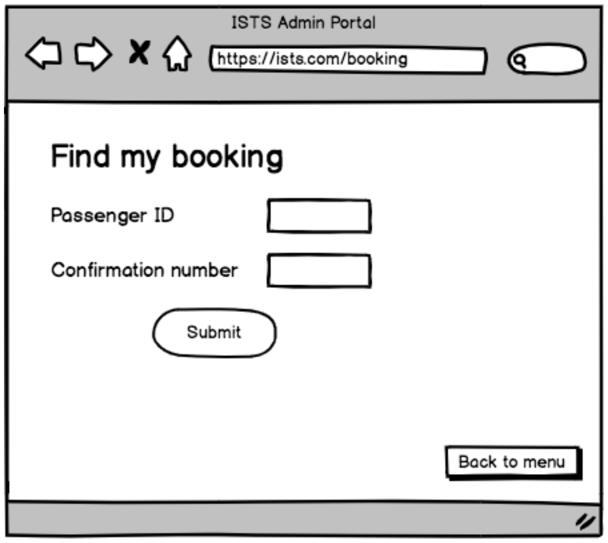


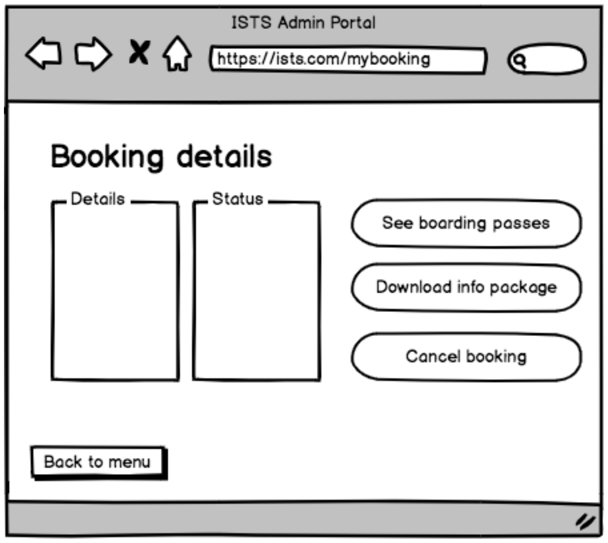
Regular booking view

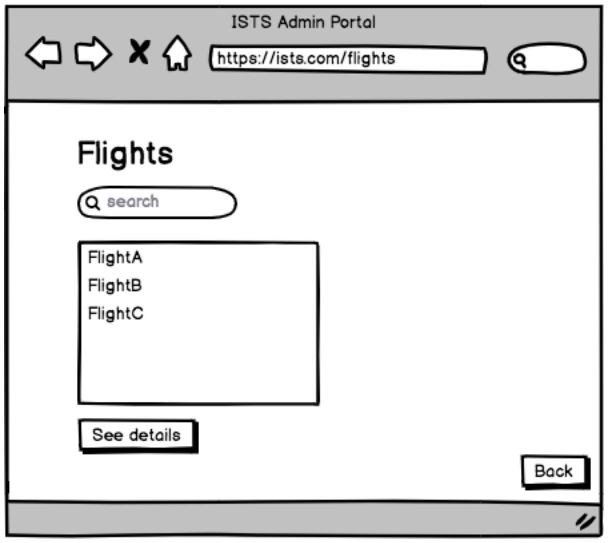

ISTS Admin Portal
https://ists.com/book
Browse flights
My booking
Register

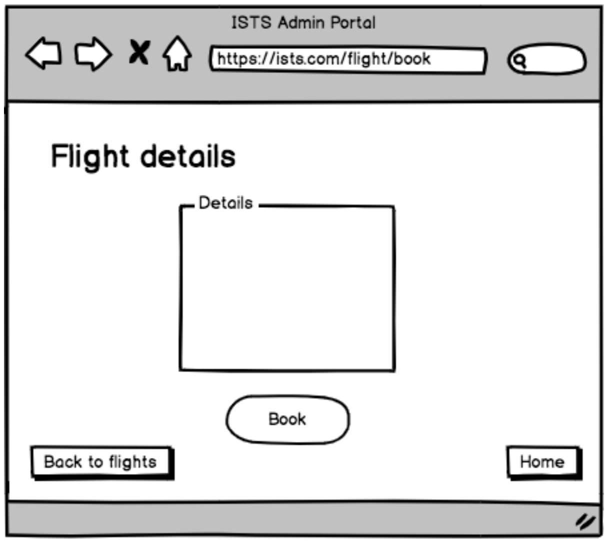

ISTS Admin Portal
https://ists.com/register
Register
First name:
Last name:
Account ID:
Travel docs:
Back
Register


ISTS Admin Portal
https://ists.com/login
Username:
Password:
Login


ISTS Admin Portal
https://ists.com/booking
Find my booking
Passenger ID
Confirmation number
Submit
Back to menu


ISTS Admin Portal
https://ists.com/mybooking
Booking details
Details
Status
See boarding passes
Download info package
Cancel booking
Back to menu


ISTS Admin Portal
https://ists.com/flights
Flights
search
FlightA
FlightB
FlightC
See details
Back


ISTS Admin Portal
https://ists.com/flight/book
Flight details
Details
Book
Back to flights
Home