Galactic Command System

User Requirements

The system is a space mission management application that enables users to manage crew members, spaceships, missions, and planets.

Crew Management

The system allows the creation and assignment of crew members, who are divided into two roles: engineers and navigators.

Crew members are represented by an abstract class, CrewMember, and specialized via inheritance into Engineer and Navigator. Each crew member has basic attributes such as first name, last name, and a credit balance used to track their value.

A crew member can either be assigned to a spaceship or stationed on a planet, but not both at once. This exclusivity is enforced through logic in the assignToSpaceShip() and assignToPlanet() methods, which throw an exception if a conflicting assignment is made. This ensures a clear one-to-one relationship between a crew member and their current location.

All crew assignments to spaceships are tracked over time using the CrewHistory class. Each record in CrewHistory stores the crew member, the spaceship, and the date of assignment, allowing a complete history of all assignments to be maintained for both the spaceship and the crewmate.

SpaceShip Management

Each spaceship has a required unique name, a list of currently assigned crew members, and a log of past assignments (crewHistoryList).

The class includes utility methods such as crewCount() to return the current number of onboard crew and needsRepairs() to indicate whether the ship requires maintenance. These methods help manage fleet status and operational readiness.

Planet Management

Planets can be added to the system with attributes such as a unique name, a position in 3D space, and an optional atmosphere type. They also contain a list of available resources (e.g. minerals), and one of these must be marked as the primary resource. The primary resource must be a subset of the planet's resources.

The number of planets is tracked by a static class-level attribute, PlanetCount, which is automatically updated when planets are added or removed.

Planets can host crew members and serve as destinations for missions.

Mission Management

Each mission has a name, a funding amount (with a minimum enforced at 100), and a type that determines its behavior at runtime.

the mission's behavior is modeled through dynamic inheritance using class flattening. A mission can be instantiated as either an ActiveMission or an InactiveMission, both of which extend the abstract base class Mission. This allows for polymorphic behavior depending on the mission type, for example, only active missions allow modifications on the objectives.

Missions also maintain a list of objectives (Objective entities), objectives are tightly bound to their parent mission and cannot exist independently.

Missions are linked to one or more spaceships through MissionAssignment entities, telling which spaceships are working on a mission.

The system enforces a rule that funding increases can't exceed 50% of the current amount.

Objective Management

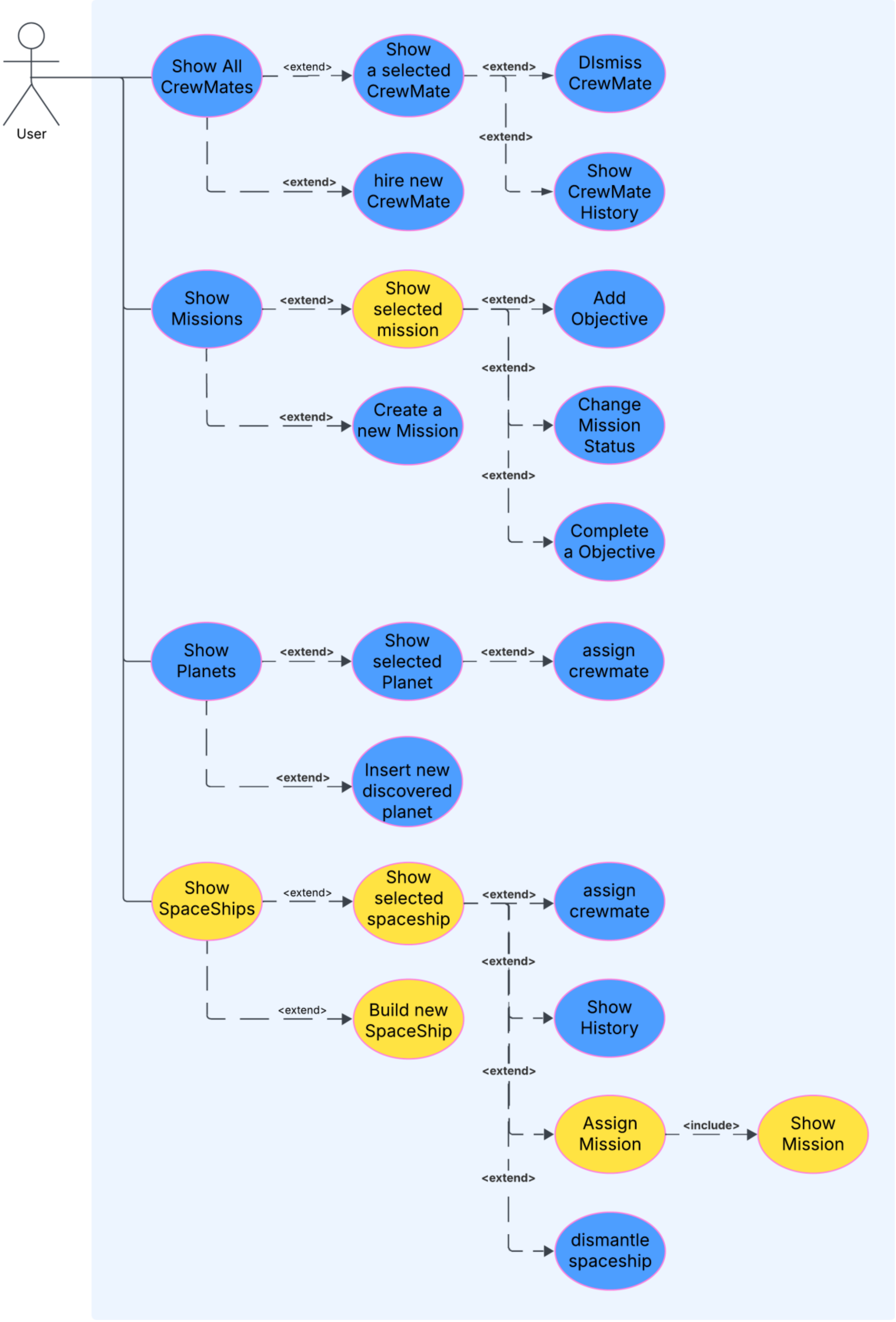
Objectives define the specific goals of a mission and are categorized as exploration or building tasks. An objective can have one or both types.

Each objective must define the necessary data depending on its type: exploration objectives require a valid location, while building objectives require both a structure and an associated planet.

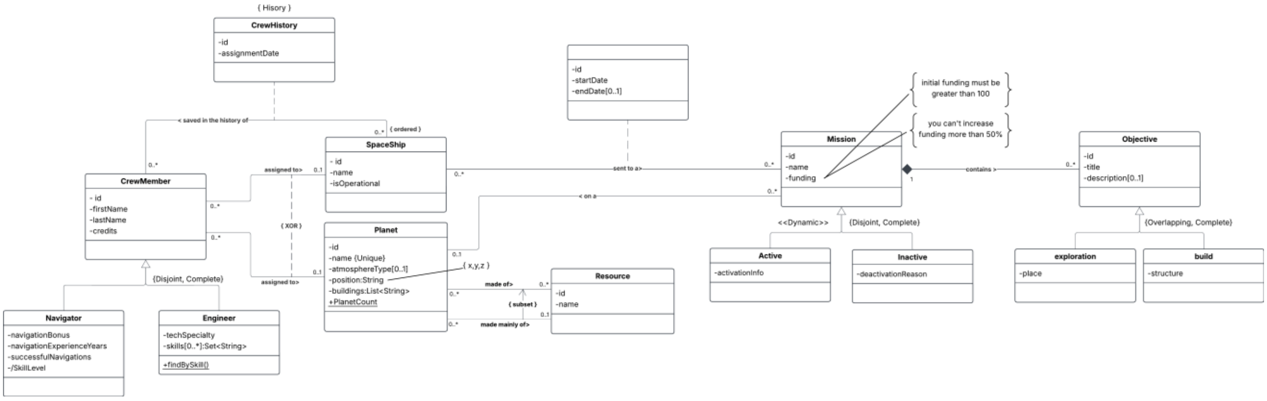
The system uses lifecycle callbacks (@PrePersist, @PreUpdate) to validate objectives before saving them. Objectives cannot be deleted from inactive missions, as enforced by @PreRemove.

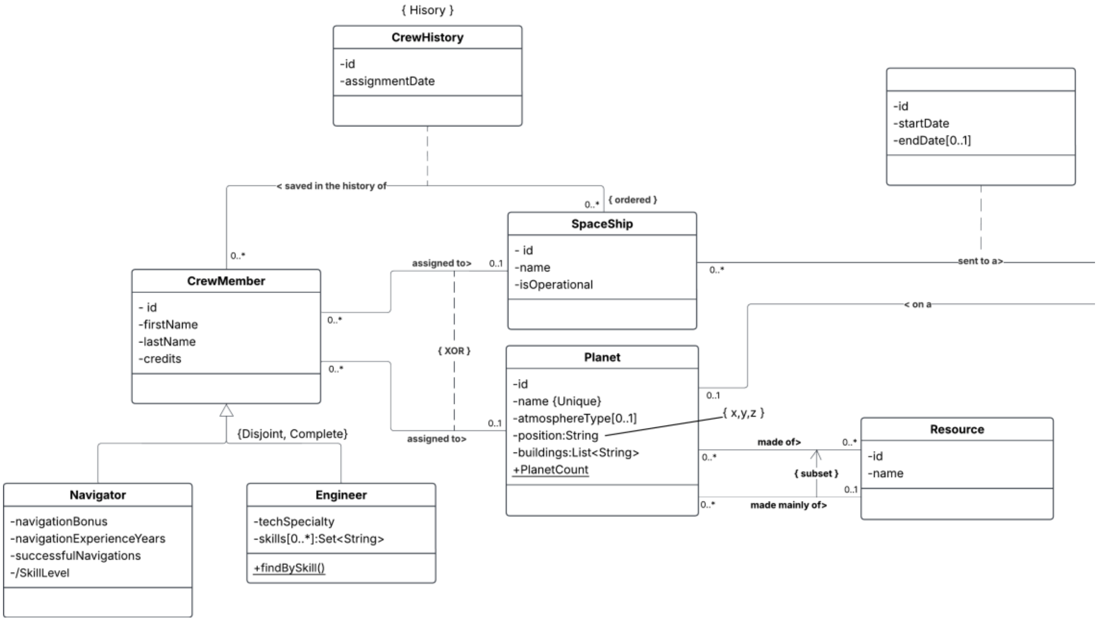
Every objective is part of a mission and is automatically removed upon completion. This ensures that only relevant, active tasks remain linked to ongoing missions.

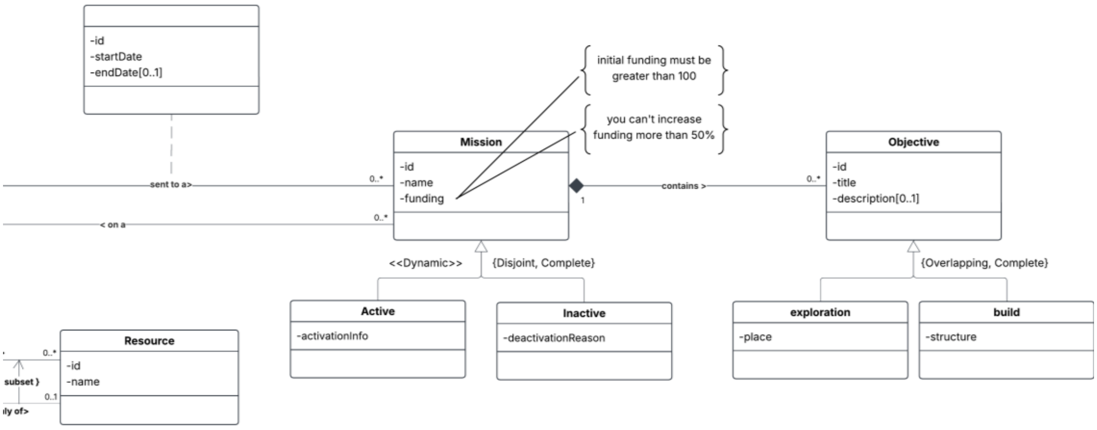
Use Case Diagram



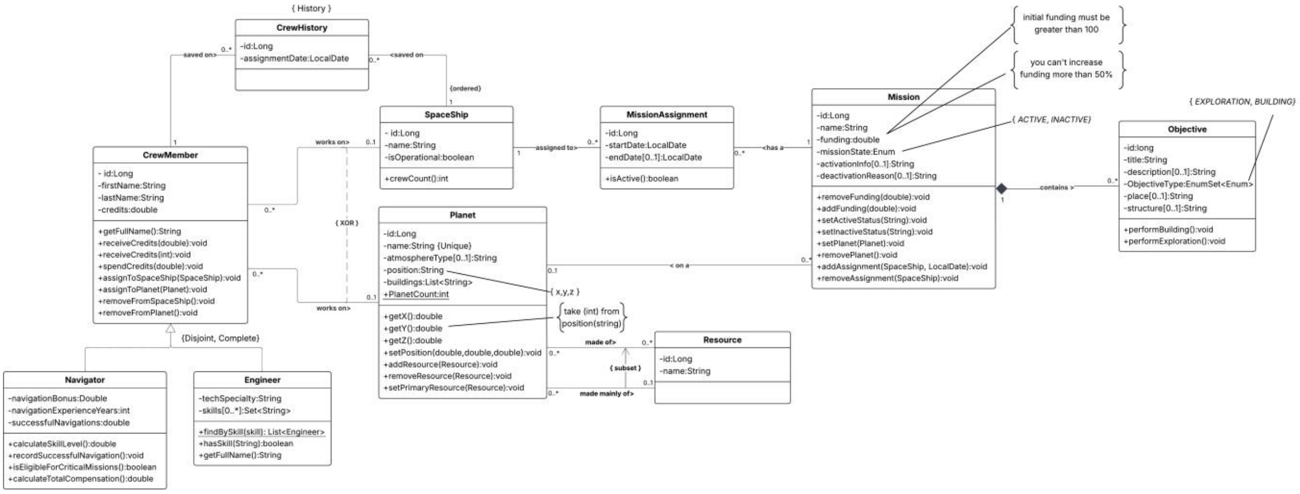
The class diagram – analytical

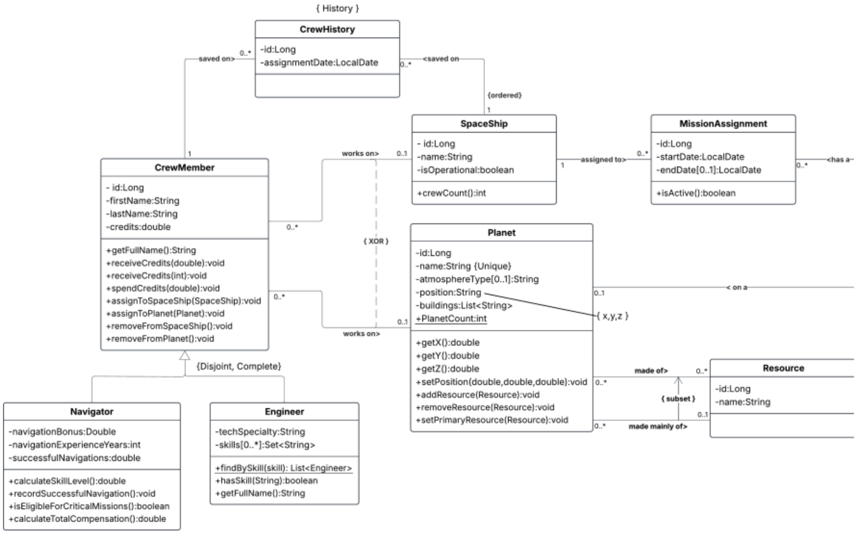


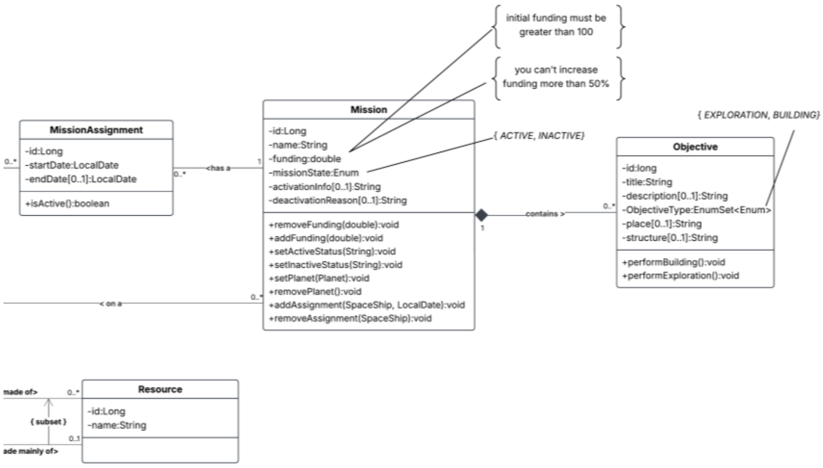




The class diagram – design


{ History }
CrewHistory
-id:Long
-assignmentDate:LocalDate
saved on>
0..*
<saved on
{ordered}
1
CrewMember
- id:Long
-firstName:String
-lastName:String
-credits:double
+getFullName():String
+receiveCredits(double):void
+receiveCredits(int):void
+spendCredits(double):void
+assignToSpaceShip(SpaceShip):void
+assignToPlanet(Planet):void
+removeFromSpaceShip():void
+removeFromPlanet():void
works on>
0..1
0..*
{ XOR }
works on>
{Disjoint, Complete}
Navigator
-navigationBonus:Double
-navigationExperienceYears:int
-successfulNavigations:double
+calculateSkillLevel():double
+recordSuccessfulNavigation():void
+isEligibleForCriticalMissions():boolean
+calculateTotalCompensation():double
Engineer
-techSpecialty:String
-skills[0..*]:Set<String>
+findBySkill(skill): List<Engineer>
+hasSkill(String):boolean
+getFullName():String
SpaceShip
- id:Long
-name:String
-isOperational:boolean
+crewCount():int
assigned to>
MissionAssignment
-id:Long
-startDate:LocalDate
-endDate[0..1]:LocalDate
+isActive():boolean
<has a
Planet
-id:Long
-name:String {Unique}
-atmosphereType[0..1]:String
-position:String
-buildings:List<String>
+PlanetCount:int
+getX():double
+getY():double
+getZ():double
+setPosition(double,double,double):void
+addResource(Resource):void
+removeResource(Resource):void
+setPrimaryResource(Resource):void
{ x,y,z }
take (int) from position(string)
< on a
made of>
{ subset }
made mainly of>
Resource
-id:Long
-name:String
initial funding must be greater than 100
you can't increase funding more than 50%
{ ACTIVE, INACTIVE}
Mission
-id:Long
-name:String
-funding:double
-missionState:Enum
-activationInfo[0..1]:String
-deactivationReason[0..1]:String
+removeFunding(double):void
+addFunding(double):void
+setActiveStatus(String):void
+setInactiveStatus(String):void
+setPlanet(Planet):void
+removePlanet():void
+addAssignment(SpaceShip, LocalDate):void
+removeAssignment(SpaceShip):void
contains >
{ EXPLORATION, BUILDING}
Objective
-id:long
-title:String
-description[0..1]:String
-ObjectiveType:EnumSet<Enum>
-place[0..1]:String
-structure[0..1]:String
+performBuilding():void
+performExploration():void





Classes and Attributes

- **Class Extent**:
  All class instances are stored in the database using JPA entity annotations (e.g., `@Entity`), which ensures persistence of every object created

- **Complex Attribute**:
  The position attribute in the Planet class is stored as a String in the format "x,y,z", which encodes three coordinate values. Helper methods getX(), getY(), getZ(), and setPosition(x, y, z) handle parsing and formatting, encapsulating this complexity.

- **Mandatory Attribute**:
  The name field in Mission is annotated with @NotBlank, meaning it's required for creating the object.

- **Optional Attribute**:
  The endDate attribute in MissionAssignment is nullable, allowing it to be left unspecified.

- **Multi-valued Attribute**:
  In the Engineer class, the skills attribute is a Set<String>, allowing multiple values.

- **Class Attribute**:
  PlanetCount is a static field in the Planet class. It's updated when an object Planet is created or deleted to reflect the current number of existing Planet instances.

- **Derived Attribute**:
  The calculateSkillLevel() method in Navigator returns a computed value based on the Navigator experience and number of successful navigations. This value is not stored in the database but is derived on-the-fly.

Methods

- **Class Method**:
  findEngineerWithSkill(String skill) is a example of a static or repository-level method that retrieves all engineers possessing a particular skill.

- **Method Overriding**:
  The `getFullName()` method is overridden in `Engineer` to include the role in the output.

- **Method Overloading**:
  In the Planet class, the setPosition() method is overloaded: one version accepts a string (like "10,20,30"), and another accepts three double values to build the string dynamically.

Associations

- **Basic Association**:
  A Mission is associated with a Planet via a @ManyToOne relationship, allowing multiple missions per planet.

- **Association with Attribute**:
  MissionAssignment represents a many-to-many relationship between Mission and SpaceShip, and adds an extra attribute (startDate, and optionally endDate) to enrich the association.

- **Composition**:
  Objective cannot exist without its parent Mission. This is enforced with orphanRemoval = true.

Inheritance

- **Abstract Class**:
  CrewMember is declared as abstract and is extended by both Engineer and Navigator, providing a shared set of fields and methods to all crew types.
  The Engineer class includes a techSpecialty, a set of skills, several methods to find specific engineers based on their skills, and a custom name.
  In contrast, the Navigator class features a navigationBonus, navigationExperienceYears, successfulNavigations, and related methods for handling navigation-specific logic.

- **Overlapping Inheritance**:
  An Objective can represent different roles such as *exploration*, *building*, or both at the same time. Rather than creating separate subclasses for each type, this is handled within a single class by flattening the hierarchy.
  Each type of objective requires specific parameters: for example, exploration objectives must define a valid place, while building objectives must specify a structure. These attributes are validated automatically to ensure that all necessary data is present, depending on the purpose of the objective.

- **Dynamic Inheritance**:
  A Mission can dynamically change its behavior at runtime by switching between INACTIVE and ACTIVE states. This state-dependent behavior is handled without separate subclasses, again achieved by flattening the class hierarchy.
  Only active missions can complete tasks.

- **Multi-Aspect Inheritance**:
  The Mission class uses an inheritance and also includes a composition relationship with Objective. At the same time, Objective itself embodies aspects of inheritance.

Constraints

- **Dynamic Constraint**:
  In Mission, addFunding() limits the increase to 50% of current funding.

- **Static Constraint**:
  The funding attribute in Mission uses @Min(100) to enforce a compile-time rule that funding must be at least 100.

- **Unique Constraint**:
  The name attribute of Planet is annotated with @Column(unique = true), ensuring that no two planets can have the same name in the database.

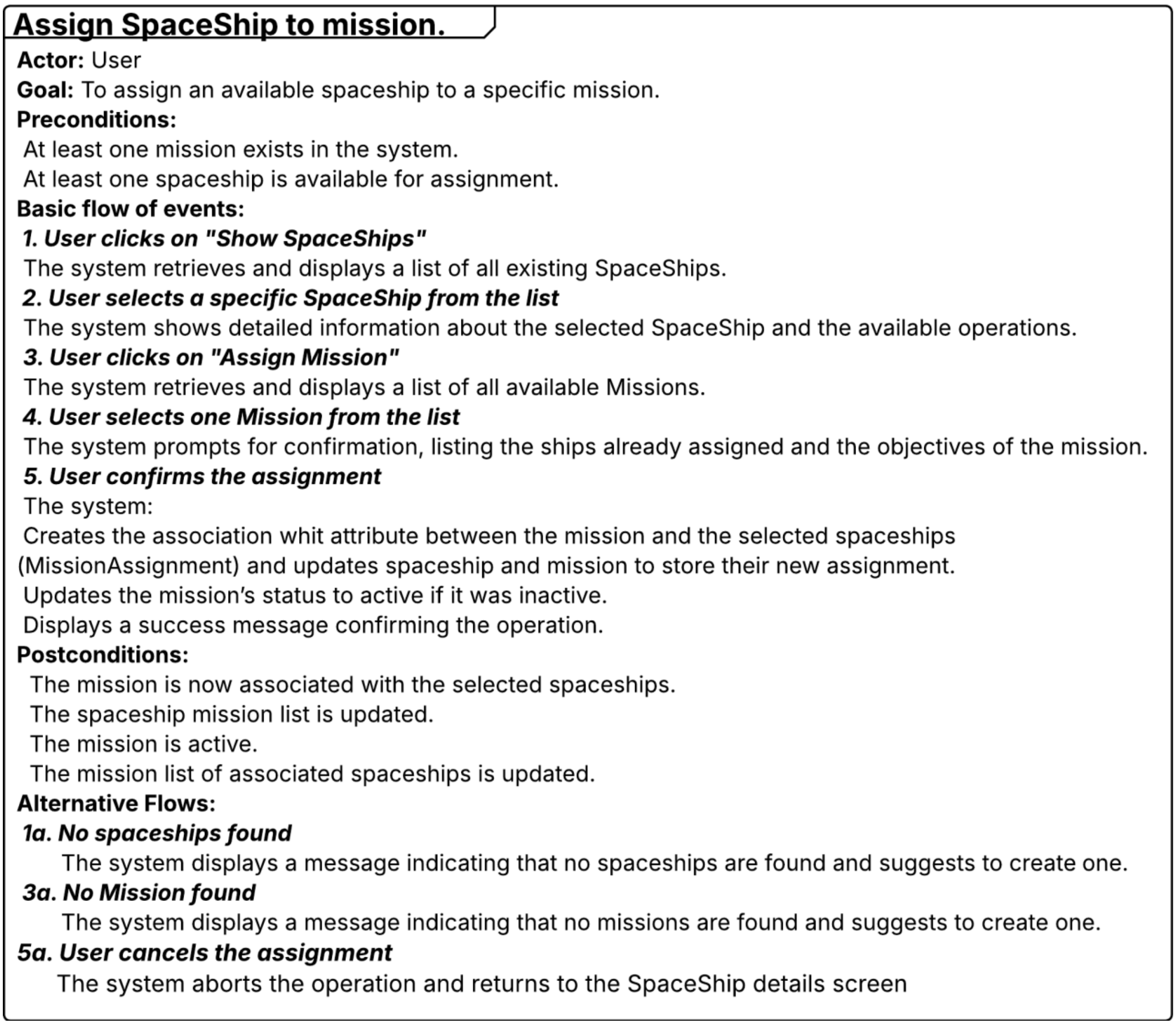
- **Subset Constraint**:
  In the Planet class, the setPrimaryResource(Resource resource) method throws an exception if the provided resource is not in the planet's general resources set enforcing the rule that the primary resource must be part of the full list.

- **Ordered Association**:
  SpaceShip contains a list of CrewHistory objects, which are ordered by assignmentDate. This ensures chronological history tracking.

- **Bag/History**:
  The CrewHistory class maintains a complete log of a CrewMember's past assignments to spaceships. This is modeled as a historical "bag" that is separate from current assignments.

- **XOR Constraint**:
  In CrewMember, a crew member can either be assigned to a Planet or a SpaceShip, but not both at the same time. This is enforced in the assignToSpaceShip() and assignToPlanet() methods through checks that throw exceptions if a conflicting assignment already exists

Use Case Scenario: Assign Spaceship to a Mission



Assign SpaceShip to mission.

Actor: User

Goal: To assign an available spaceship to a specific mission.

Preconditions:

At least one mission exists in the system.

At least one spaceship is available for assignment.

Basic flow of events:

1. User clicks on "Show SpaceShips"

The system retrieves and displays a list of all existing SpaceShips.

2. User selects a specific SpaceShip from the list

The system shows detailed information about the selected SpaceShip and the available operations.

3. User clicks on "Assign Mission"

The system retrieves and displays a list of all available Missions.

4. User selects one Mission from the list

The system prompts for confirmation, listing the ships already assigned and the objectives of the mission.

5. User confirms the assignment

The system:

Creates the association whit attribute between the mission and the selected spaceships (MissionAssignment) and updates spaceship and mission to store their new assignment.

Updates the mission's status to active if it was inactive.

Displays a success message confirming the operation.

Postconditions:

The mission is now associated with the selected spaceships.

The spaceship mission list is updated.

The mission is active.

The mission list of associated spaceships is updated.

Alternative Flows:

1a. No spaceships found

The system displays a message indicating that no spaceships are found and suggests to create one.

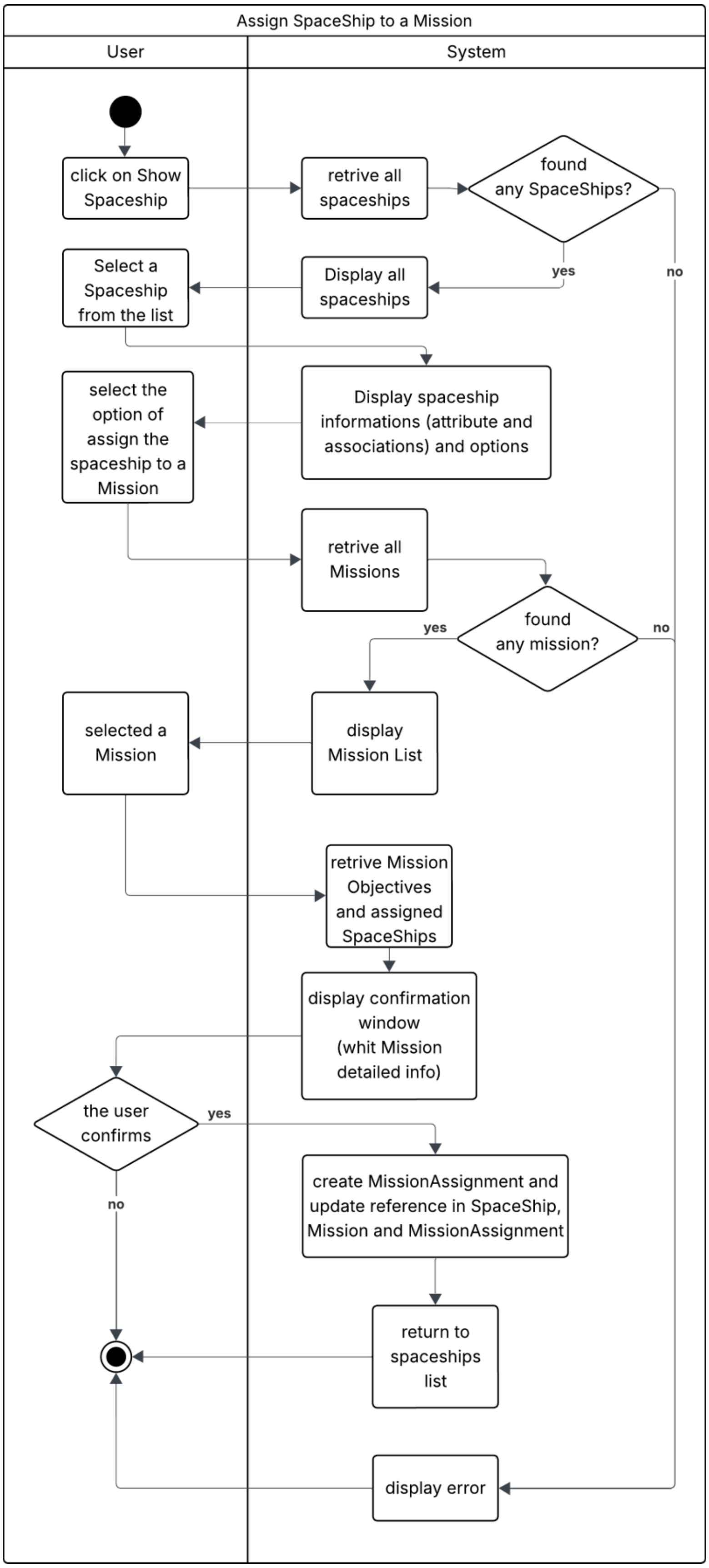
3a. No Mission found

The system displays a message indicating that no missions are found and suggests to create one.

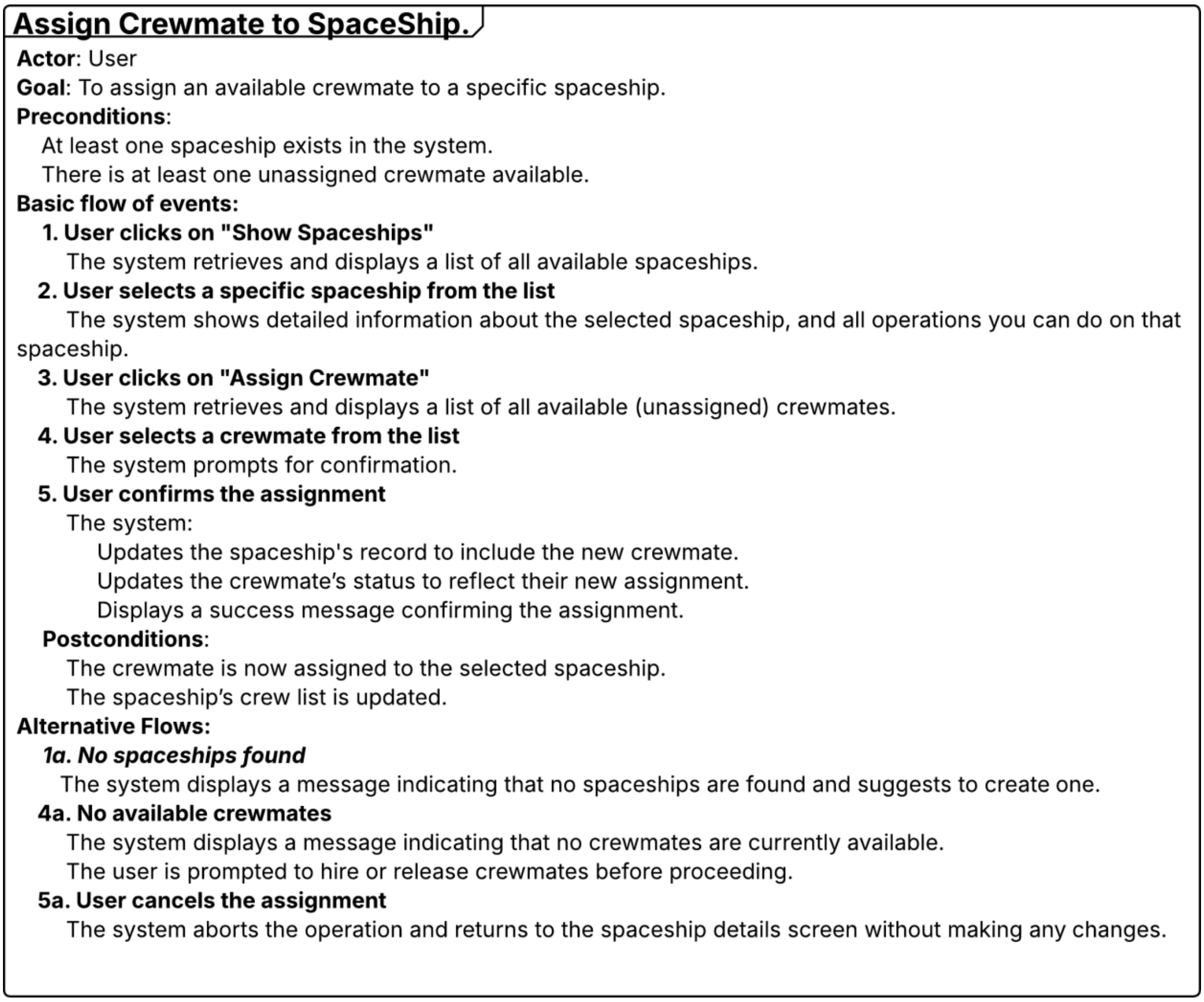
5a. User cancels the assignment

The system aborts the operation and returns to the SpaceShip details screen

Activity Diagram: Assign Spaceship to a Mission


Assign SpaceShip to a Mission
User
System
click on Show Spaceship
retrive all spaceships
found any SpaceShips?
yes
no
Select a Spaceship from the list
Display all spaceships
select the option of assign the spaceship to a Mission
Display spaceship informations (attribute and associations) and options
retrive all Missions
found any mission?
yes
no
selected a Mission
display Mission List
retrive Mission Objectives and assigned SpaceShips
display confirmation window (whit Mission detailed info)
the user confirms
yes
no
create MissionAssignment and update reference in SpaceShip, Mission and MissionAssignment
return to spaceships list
display error

Use Case Scenario: Assign Crewmate to Spaceship



Assign Crewmate to SpaceShip.

Actor: User

Goal: To assign an available crewmate to a specific spaceship.

Preconditions:

At least one spaceship exists in the system.
There is at least one unassigned crewmate available.

Basic flow of events:

1. User clicks on "Show Spaceships"

The system retrieves and displays a list of all available spaceships.

2. User selects a specific spaceship from the list

The system shows detailed information about the selected spaceship, and all operations you can do on that spaceship.

3. User clicks on "Assign Crewmate"

The system retrieves and displays a list of all available (unassigned) crewmates.

4. User selects a crewmate from the list

The system prompts for confirmation.

5. User confirms the assignment

The system:

Updates the spaceship's record to include the new crewmate.
Updates the crewmate’s status to reflect their new assignment.
Displays a success message confirming the assignment.

Postconditions:

The crewmate is now assigned to the selected spaceship.
The spaceship’s crew list is updated.

Alternative Flows:

1a. No spaceships found

The system displays a message indicating that no spaceships are found and suggests to create one.

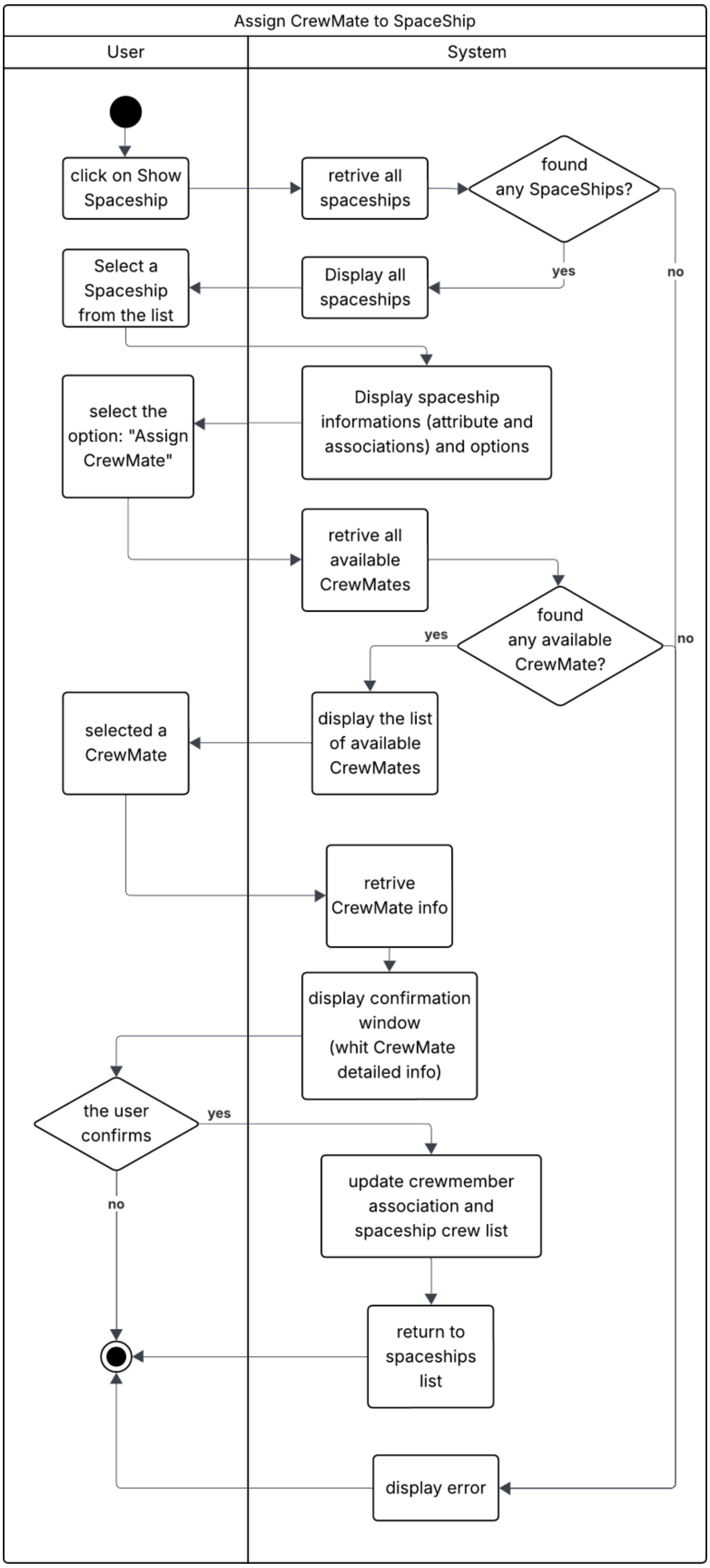
4a. No available crewmates

The system displays a message indicating that no crewmates are currently available.
The user is prompted to hire or release crewmates before proceeding.

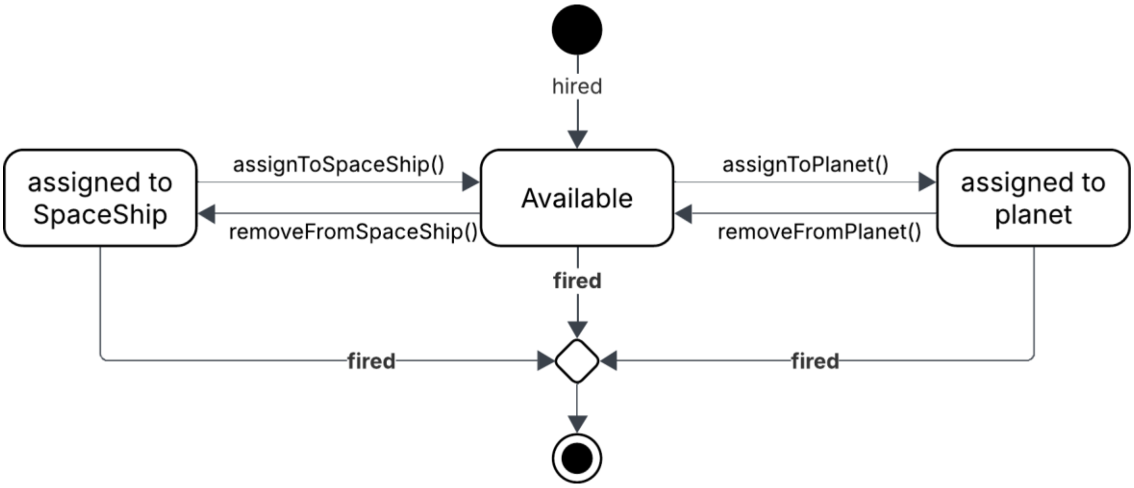
5a. User cancels the assignment

The system aborts the operation and returns to the spaceship details screen without making any changes.

Activity Diagram: Assign Crewmate to Spaceship


Assign CrewMate to SpaceShip
User
System
click on Show Spaceship
retrive all spaceships
found any SpaceShips?
yes
no
Select a Spaceship from the list
Display all spaceships
Display spaceship informations (attribute and associations) and options
select the option: "Assign CrewMate"
retrive all available CrewMates
found any available CrewMate?
yes
no
selected a CrewMate
display the list of available CrewMates
retrive CrewMate info
display confirmation window (whit CrewMate detailed info)
the user confirms
yes
no
update crewmember association and spaceship crew list
return to spaceships list
display error

State Diagram - Crewmate



Purpose

This state diagram models the lifecycle of a Crewmate object within the system. It defines the valid states a crewmate can occupy and the events that trigger transitions between those states.

Valid States

- **Available**

  This is the initial active state after a crewmate has been created (i.e., hired).

  In this state, the crewmate is idle and ready to be assigned to a ship or planet.

  Incoming Transitions: Upon creation, or when removed from a ship or planet.

  Outgoing Transitions: Assignment to a spaceship, assignment to a planet, or dismissal.

- **AssignedToShip**
  The crewmate is currently assigned to a spaceship. While in this state, they cannot be assigned elsewhere.

- **AssignedToPlanet**
  The crewmate is currently stationed on a planet or colony. They are not available for other tasks.

- **Fired**
  The crewmate has been permanently removed from duty. This is a terminal state with no further transitions allowed.

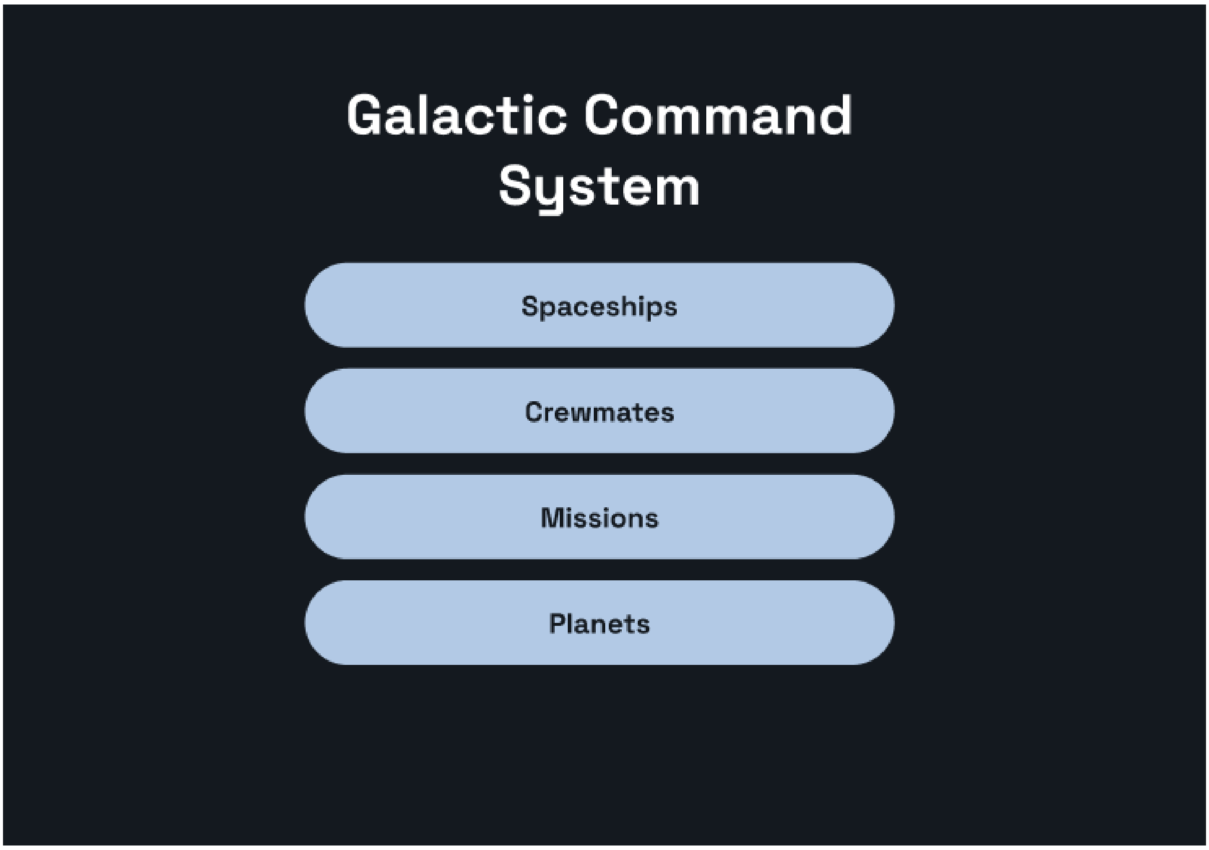
Constraints

- A crewmate can only be assigned if they are in the "Available" state.
- A crewmate cannot be assigned to both a ship and a planet at the same time.
- All assignments and removals must update both the crewmate and the associated entities (ship or planet).
- The "Fired" state is terminal; no further transitions are allowed once reached.

Transitions

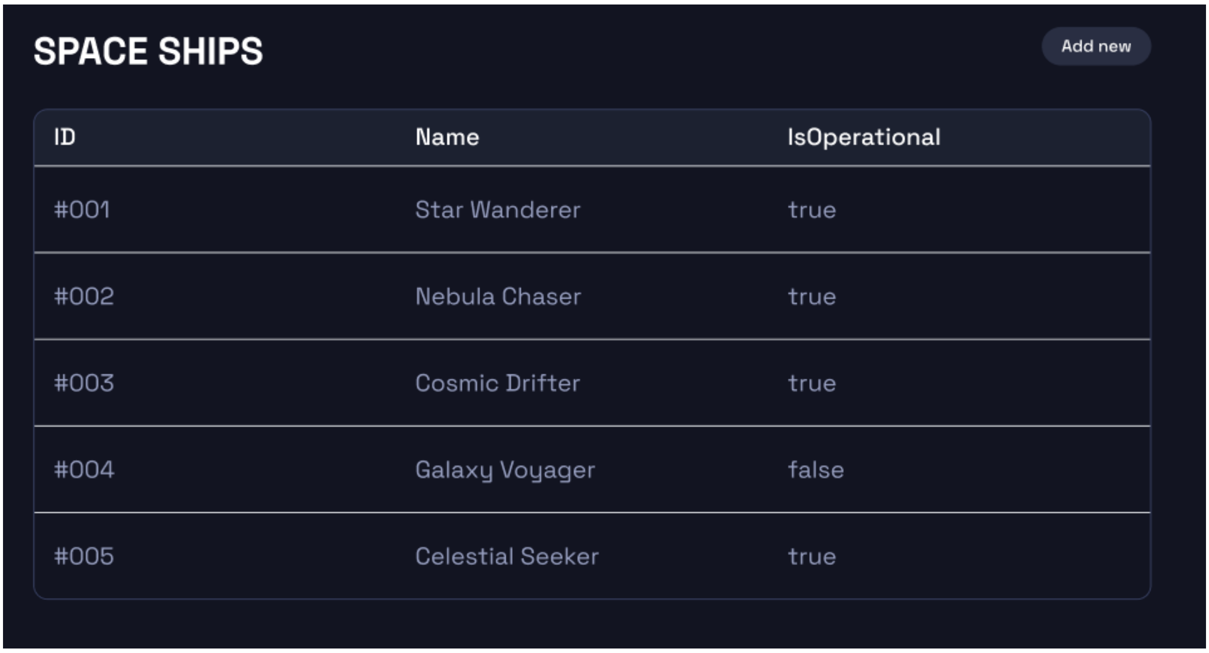
- **assignToSpaceShip(ship)**
  - Check if the CrewMate is available
  - Sets the spaceship reference
  - Adds the crewmate to the ship’s crew
  - State becomes AssignedToShip
- **assignToPlanet(planet)**
  - Check if the CrewMate is available
  - Sets the planet reference
  - Adds the crewmate to the planet's workers
  - State becomes AssignedToPlanet
- **removeFromSpaceShip()**
  - Removes from the ship’s crew
  - Clears spaceship reference
  - State becomes Available
- **removeFromPlanet()**
  - Removes from the planet’s workers
  - Clears planet reference
  - State becomes Available
- **fire**
  - Removes from ship or planet
  - Clears all assignments
  - Delete the object

GUI design



This is the home page of our application.
From here, users can navigate to various sections of the system:

- Clicking on **"Spaceships"** will take you to a list of all spaceships.
- Clicking on **"Crewmates"** opens a list of all crew members currently in the system.
- Clicking on **"Missions"** shows an overview of all missions, including their statuses and details.
- Clicking on **"Planets"** displays a list of all known planets.



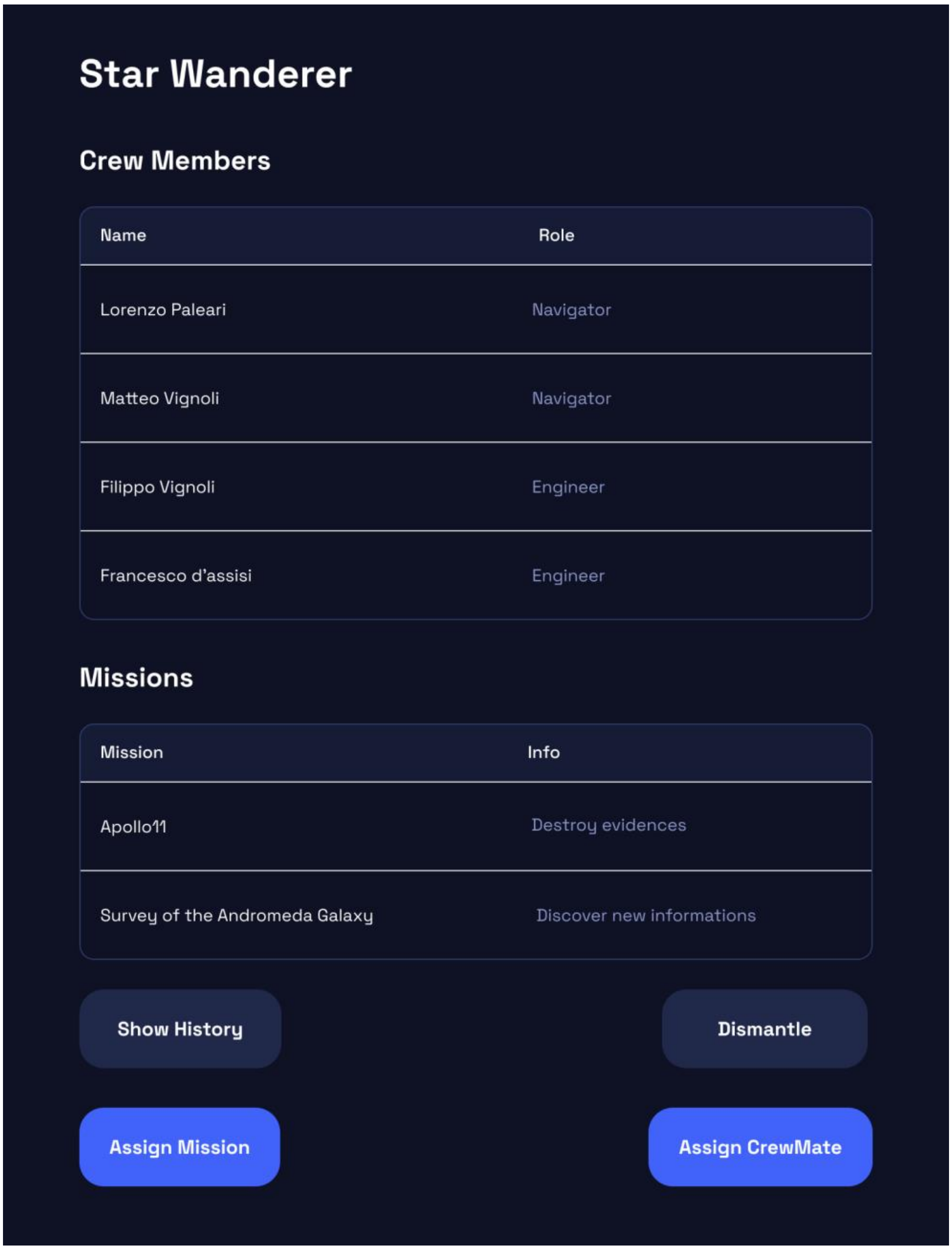
ID	Name	IsOperational
#001	Star Wanderer	true
#002	Nebula Chaser	true
#003	Cosmic Drifter	true
#004	Galaxy Voyager	false
#005	Celestial Seeker	true

After selecting the **"SpaceShip"** section, you are directed to a page displaying all created SpaceShips.
Each spaceShip has:

- **ID**
- **Name**
- **Operational status**

From this page, you have two main options:

- Click **"Add New"** to be redirected to a form for creating a new spaceship.
- Click on a SpaceShip to view its detailed profile and perform various actions related to the vessel.



Star Wanderer

Crew Members

Name	Role
Lorenzo Paleari	Navigator
Matteo Vignoli	Navigator
Filippo Vignoli	Engineer
Francesco d'assisi	Engineer

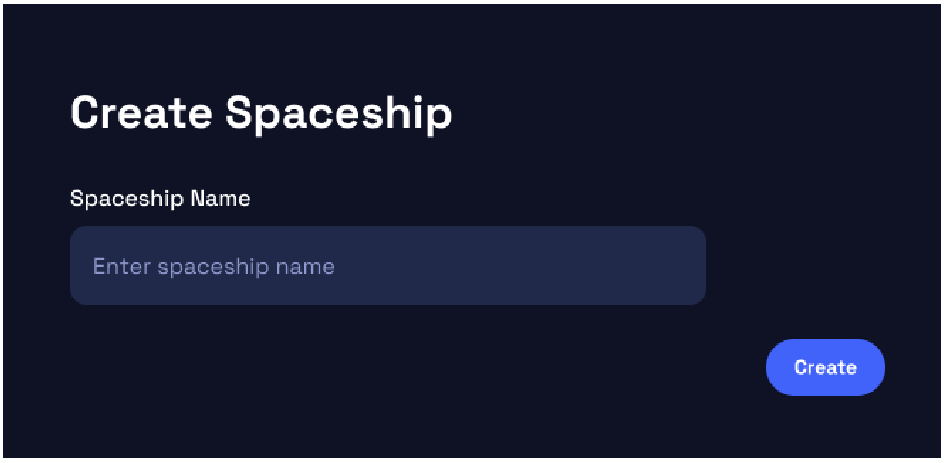
Missions

Mission	Info
Apollo11	Destroy evidences
Survey of the Andromeda Galaxy	Discover new informations

When a specific spaceship is **selected** from the list, its profile page is displayed. This page provides detailed information about the spaceship, including the list of assigned crew members and the missions it is currently involved in.

From this page, you can:

- **Click "Show History"** to view a timeline of all previous crew members assigned to the spaceship.
- **Click "Assign Mission"** to open a page where you can assign the spaceship to a new mission.
- **Click "Assign Crewmate"** to add a new crew member to the ship.
- **Click "Dismantle"** to remove the spaceship and delete all its associations.



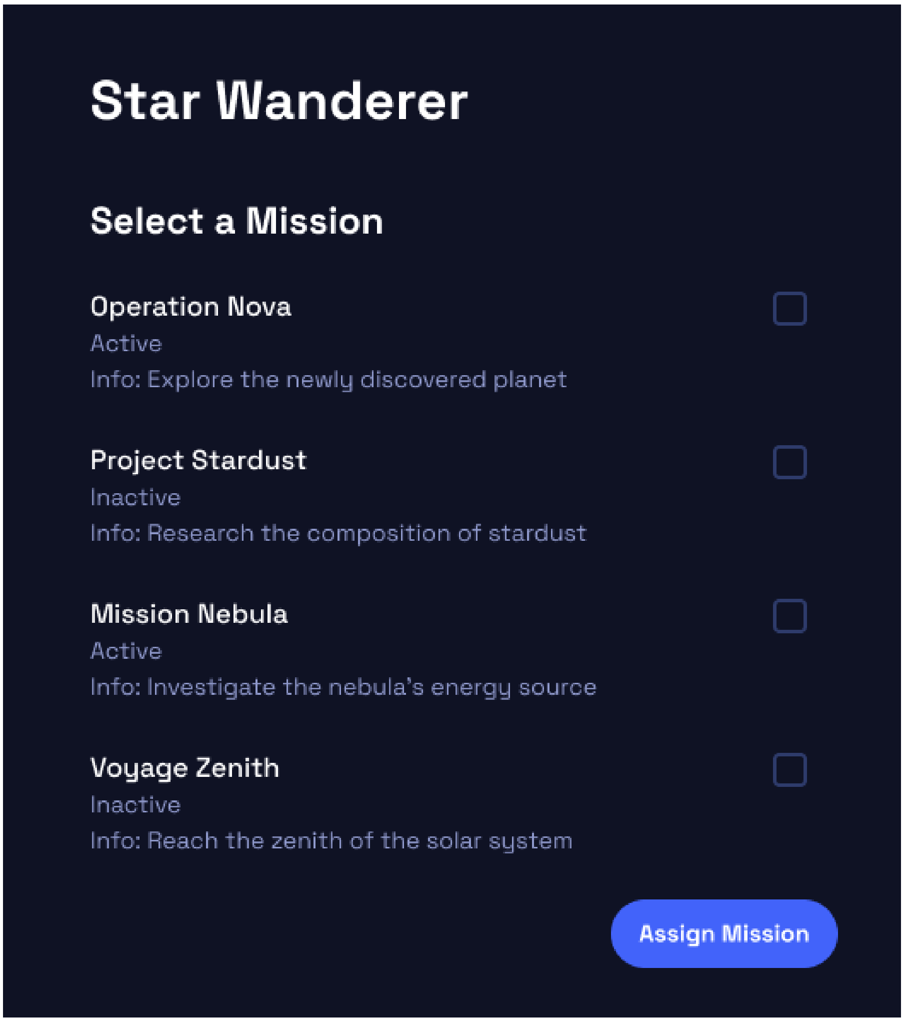
Create Spaceship

Spaceship Name

Enter spaceship name

Create

After clicking on "**Add New**," you will be redirected to this page. Here, you can enter the name of the spaceship and then click "**Create**" to create the new object.



Star Wanderer

Select a Mission

Operation Nova □
Active
Info: Explore the newly discovered planet

Project Stardust □
Inactive
Info: Research the composition of stardust

Mission Nebula □
Active
Info: Investigate the nebula's energy source

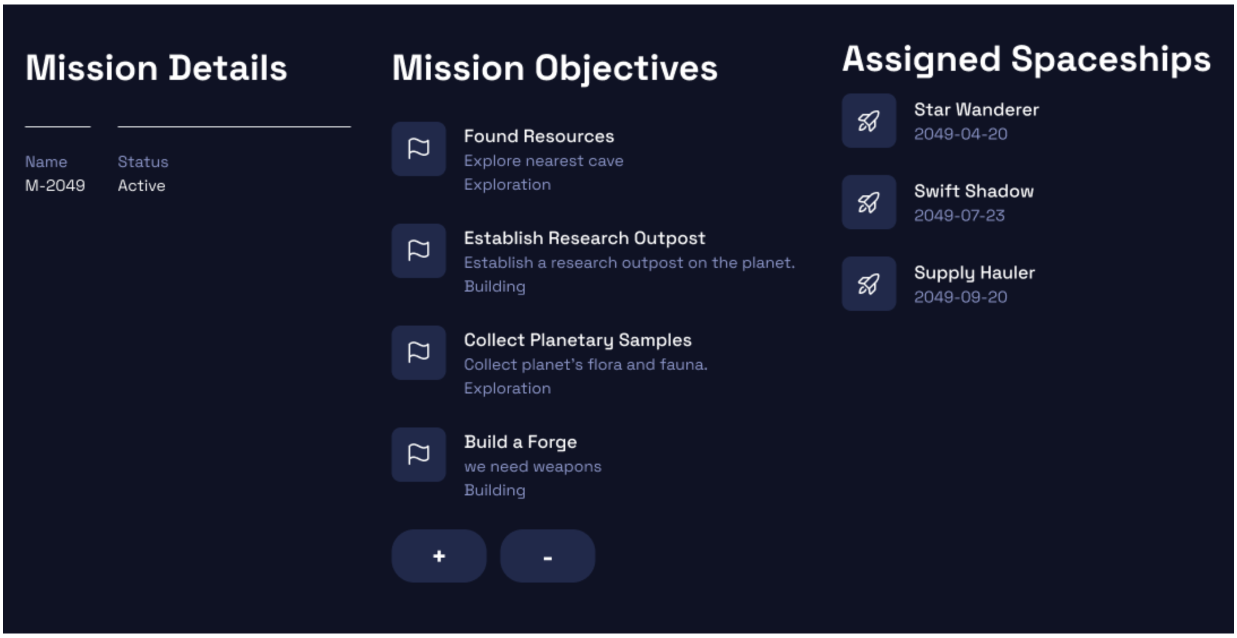
Voyage Zenith □
Inactive
Info: Reach the zenith of the solar system

Assign Mission

This page is accessed by clicking "Assign Mission" on the page displaying a specific spaceship.

Here, all available missions that the spaceship can join are listed.

To assign a mission to the spaceship, simply select a mission and click "Assign Mission". This action will create a Mission Assignment object, update the related associations accordingly, and automatically redirect you to the page displaying the mission details.



Mission Details

Name	Status
M-2049	Active

Mission Objectives

Found Resources
Explore nearest cave
Exploration

Establish Research Outpost
Establish a research outpost on the planet.
Building

Collect Planetary Samples
Collect planet's flora and fauna.
Exploration

Build a Forge
we need weapons
Building

Assigned Spaceships

Star Wanderer
2049-04-20

Swift Shadow
2049-07-23

Supply Hauler
2049-09-20

This page displays the details of a specific mission, including its name, status, all associated objectives (along with their descriptions and types), and the spaceships assigned to the mission.

At the bottom of the objectives list, there are two buttons that allow you to add new objectives or remove those that have already been completed.