System operations :

startApplication()

initializeSystem()

loadRailwayData()

searchConnections(departureCity, arrivalCity, date, passengers)

findRoutes()

calculateTripDuration(connections)

calculateTripPrice(connections)

validateChronology(connections)

displaySearchResults(tripList)

calculateTransferTime(connections)

Operation Contract

Operation Contracts :

Operation: startApplication()
Goal: Initialize the Django application and prepare the system for railway data loading
Cross References: Use Case: Load Railway Data
Preconditions:

- Django application is not running
- Railway system is not initialized
- System environment is properly configured

Postconditions:

- Django application starts successfully
- Django settings are loaded
- Application configuration is initialized
- System initialization process begins

Operation: initializeSystem()
Goal: Create the singleton StationNetworkManager instance and prepare the system for data loading

Cross References: Use Case: Load Railway Data

Preconditions:

- Django application is running
- StationNetworkManager is not initialized
- System configuration is valid

Postconditions:

- StationNetworkManager singleton instance is created
- System is ready for data loading
- Network manager is initialized with default settings

Operation: loadRailwayData()

Goal: Load railway network data from CSV file and build the graph structure for route searching

Cross References: Use Case: Load Railway Data

Preconditions:

- System is initialized
- CSV file exists at "./backend_django/data/eu_rail_network.csv"
- CSV file contains valid railway data with required columns
- File is readable and properly formatted

Postconditions:

- CSV data is parsed into Connection objects
- Station objects are created for each departure city
- Network graph structure is built
- Connections are organized by day of week and arrival city
- Test search is performed (Amsterdam to Eindhoven, Saturday)
- System is ready for searches

Operation: searchConnections(departureCity, arrivalCity, date, passengers)

Goal: Find all possible railway routes between two cities and calculate trip details

Cross References: Use Case: Search for Connection

Preconditions:

- Railway data is loaded
- departureCity and arrivalCity are valid City enums
- date is a valid date
- passengers is a positive integer
- Network graph is built

Postconditions:

- All possible routes between cities are found
- Routes are limited to maximum 3 connections
- Trip objects are created for valid routes

- Trip durations and prices are calculated
- Search results are prepared for display

Operation: findRoutes()
Goal: Execute DFS algorithm to find all valid paths between departure and arrival cities
Cross References: Use Case: Search for Connection
Preconditions:

- Search parameters are valid
- Network graph is built
- StationNetworkManager is initialized

Postconditions:

- DFS algorithm is executed
- All valid paths (direct and indirect) are found
- Routes are limited to maximum 3 connections
- Chronological validation is performed
- List of valid routes is returned

Operation: calculateTripDuration(connections)
Goal: Calculate the total travel duration for a trip consisting of multiple connections
Cross References: Use Case: Calculate Trip Duration
Preconditions:

- connections is a non-empty list of Connection objects
- All connections are chronologically valid
- Connection objects have valid departure and arrival times

Postconditions:

- Trip object is created
- total_travel_duration is calculated by summing connection durations
- departure_time is set from first connection
- arrival_time is set from last connection
- isDirect flag is determined based on number of connections
- Trip object is returned with calculated duration

Operation: calculateTripPrice(connections)
Goal: Calculate the total cost for first and second class tickets across all connections in a trip
Cross References: Use Case: Search for Connection
Preconditions:

- connections is a non-empty list of Connection objects
- All connections have valid price data
- Price values are positive numbers

Postconditions:

- total_first_class_price is calculated by summing first class prices
- total_second_class_price is calculated by summing second class prices
- Prices are summed across all connections in the trip
- Trip object is updated with calculated prices

Operation: validateChronology(connections)
Goal: Ensure that connections in a route are chronologically valid so passengers can make transfers
Cross References: Use Case: Search for Connection
Preconditions:

- connections is a list of Connection objects
- Connections are in sequence order
- Each connection has valid departure and arrival times

Postconditions:

- Each connection's arrival_time is validated against next connection's departure_time
- Only chronologically valid routes are kept
- Invalid routes are filtered out
- Validated connection list is returned

Operation: displaySearchResults(tripList)
Goal: Format and present search results to the user with all relevant trip information
Cross References: Use Case: Search for Connection
Preconditions:

- tripList is a list of Trip objects
- All trips have calculated durations and prices
- Trip objects contain valid connection data

Postconditions:

- Search results are formatted for display
- Trip information is presented to user
- Results include duration, price, and transfer information
- Formatted results are returned to client

Operation: calculateTransferTime(connections)**Goal:** Calculate the total waiting time between connections for multi-leg trips**Cross References:** Use Case: Calculate Trip Duration**Preconditions:**

- connections is a list with multiple Connection objects
- Connections are chronologically ordered
- Each connection has valid departure and arrival times

Postconditions:

- Transfer time between connections is calculated
- Wait times are summed for total transfer time
- Only positive wait times are included
- Total transfer time is returned

Each contract now includes a clear goal statement that explains the purpose and objective of the operation.
