Use Cases

Use Case 1: Load Railway Data

Use Case ID and Name: UC1 - Load Railway Data

Primary Actor: System Developer

Stakeholders and Interests:

- System Developer: Wants to ensure that railway data is loaded correctly and available for user queries.
- Client: Needs valid and complete data in memory to search for train connections.
- System: Must handle file I/O efficiently and report errors if data cannot be loaded.

Preconditions: The CSV file containing railway network data exists at a known location.

Success Guarantee (Postconditions):

- All records from the CSV file are successfully loaded into memory (e.g., stored in an ArrayList<Connection>).
- The data is ready for search and query operations.

Main Success Scenario (Basic Flow):

1. The actor, System Developer, starts the data loading process.
2. The system locates the CSV file from the specified directory.
3. The system opens and reads the file line by line.
4. For each record, the system parses fields: Route ID, Departure City, Arrival City, Departure Time, Arrival Time, Train Type, Days of Operation, First Class Rate, Second Class Rate

5. The system validates that each record follows the correct format and data types.
6. The system creates Connection objects for each record.
7. All objects are added to an in-memory collection, an ArrayList<Connection>
8. The system reports successful loading and number of records processed.

Extensions (Alternative Flows): If the file is not found the system aborts the loading process.

Special Requirements:

- The file parser must handle large data efficiently.
- The system should validate time formats (e.g., HH:MM) and numeric fields.

Technology and Data Variations List:

- File path may be configured via settings or loaded from a default path.
- CSV may use commas or semicolons as delimiters (configurable).

Open Issues: Should invalid records stop the loading process or just be skipped?

Use Case 2: Search for Connections

Use Case ID and Name: UC2 - Load Railway Data

Primary Actor: Client

Stakeholders and Interests:

- Client: Wants to find available train connections between specific cities, with details such as duration, ticket prices, and train type.
- System Developer: Wants to ensure that search functionality works properly and returns the exact results that the Client wants.
- System: Must perform searches quickly, calculate trip durations correctly, and handle both direct and indirect connections up to 2 stops.

Preconditions: Railway connections data has been successfully loaded into memory.

Success Guarantee (Postconditions):

- The system displays all matching connections (direct and indirect) according to the user set parameters.
- Each displayed connection includes all details and the calculated trip duration.

Main Success Scenario (Basic Flow):

1. The Client starts search by providing 0 or more search parameters (e.g., Departure City, Arrival City, Departure Time, Train Type, etc.).
2. The System searches the in-memory data for all direct connections that match the given criteria.
3. The System calculates the trip duration for each direct connection.
4. If no direct connection exists, the System searches for indirect connections with one stop (A → B → C).
5. If still no valid route exists, the System searches for two-stop connections (A → B → C → D) and calculates total travel time accordingly.
6. The System collects all matching direct and indirect routes into a result list.
7. The System displays the results to the Client, showing for each connection the: Departure City, Departure Time, Arrival City, Arrival Time, Train Type, Days of Operation, Ticket Rates, Total Trip Duration

Extensions (Alternative Flows):

- Invalid Search Parameters that the client enters make the system display the message “No connections found with the given parameters.”

Special Requirements:

- The search function should execute quickly, even for large datasets.
- Trip duration should be calculated accurately based on departure and arrival times taking into consideration the +1 day.

Technology and Data Variations List:

- Search may be case-insensitive (e.g., “Bilbao” = “bilBaO” or “BILbao”, etc.).

Open Issues: -

Use Case 3: Sort Search Results

Use Case ID and Name: UC3 - Sort Search Results

Primary Actor: Client

Stakeholders and Interests:

- Client: Wants to view the search results in a specific order (for example, by shortest trip duration or lowest ticket price, first class or second class) to make better travel decisions.
- System Developer: Wants the correct sorting functions for different criteria.

Preconditions: The client has already performed a successful search for connections so there are connections available for sorting.

Success Guarantee (Postconditions):

- Search results are re-ordered according to the selected sorting criterion (trip duration, first-class price, or second-class price).
- The sorted results are displayed to the client in the correctly set order.

Main Success Scenario (Basic Flow):

1. After viewing the search results in the display, the System asks the client if they want to sort the results.
2. If the Client selects to sort, they are presented with the option to sort by Duration, First class price, and Second class price.
3. The Client selects a sorting criterion.
4. The System receives the sorting request.
5. The System applies the selected sorting algorithm to the list of results.

6. The System rearranges the results in ascending order of the chosen criterion.
7. The System displays the updated list on the console.
8. The System confirms the applied sorting criterion (e.g., “Results sorted by Trip Duration”).

Extensions (Alternative Flows):

- Invalid Sorting input by Client makes the system displays an error message: “Invalid choice. Showing unsorted results:” and System

Special Requirements:

- Sorting should execute quickly for large result sets.

Technology and Data Variations List: -

Open Issues: -