AltaGas TailCar-Trip Design Document

GitHub repo
https://github.com/GigCory/railcar-trip

Use Stories:

1. As a Rail Operations, I want to upload a CSV file containing railcar/equipment events, So that the system can process them into trips automatically.
2. As a System, I want to group events by equipment, order them by UTC, and create trips starting with W and ending with Z.So that trips are correctly calculated and stored in the database.
3. As a Rail Operations Manager, I want to see a grid of all processed trips with equipment, origin, destination, start/end times, and total hours, so that I can monitor equipment movements and operational performance.
4. As a Analyst/Planner,I want to select a specific trip from the trips grid,So that I can see all equipment events associated with that trip in chronological order.

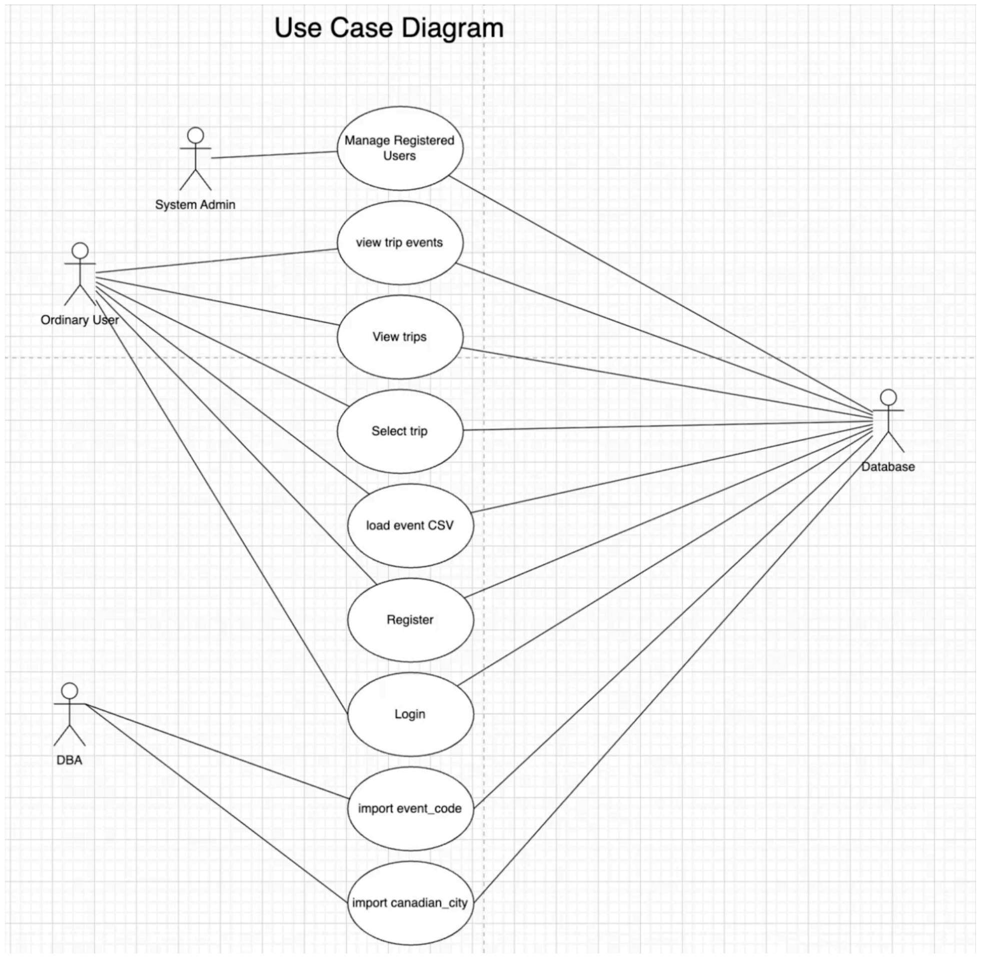
Function requirements:

1. Upload CSV files containing railcar/equipment events.Validate file format and required columns.
2. The system must process uploaded events by grouping them by equipment ID, ordering them by UTC time, creating trips from W (Released) to Z (Placed) events, calculating total trip hours, and persisting the trips along with their associated events in the database.
3. The system must display a grid of processed trips showing the equipment ID, origin city, destination city, start UTC time, end UTC time, and total trip hours for each trip.
4. The system must allow the user to select a trip from the grid and display all associated events in chronological order, including the event code, local and UTC time, and city.

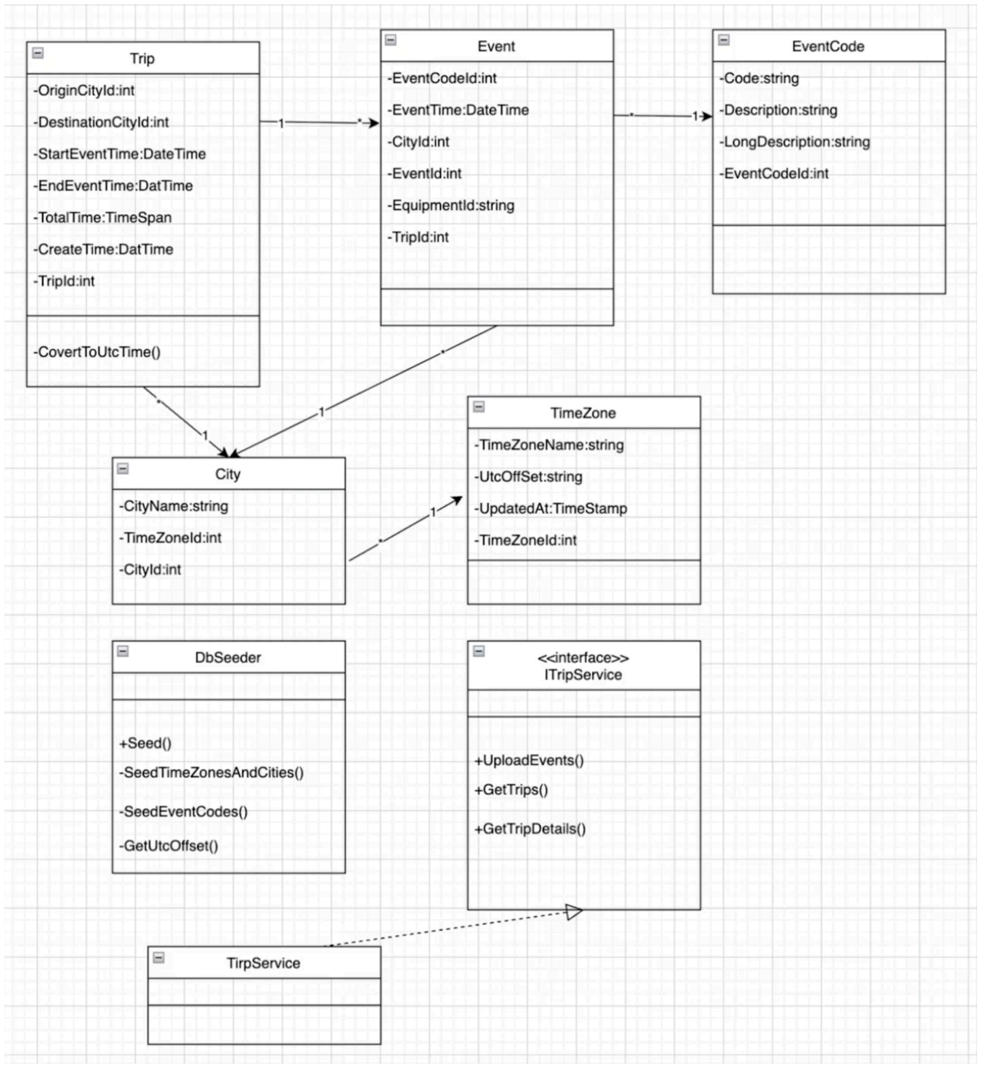
Non function requirements

1. The system should process CSV files with thousands of events efficiently.Grid loading and event drill-down should be responsive.
2. Data should be persisted consistently.System should handle invalid or partial input gracefully.
3. Conversion of local time to UTC must be precise for all supported cities.
4. Use EF Core for easier future updates.

User case Diagram



Class Diagram:



ER diagram:

Technical Architecture:

The “RailcarTrips” system is designed using a modern, cloud-native architecture that emphasizes scalability, maintainability, and clear separation of concerns.

Architecture Style

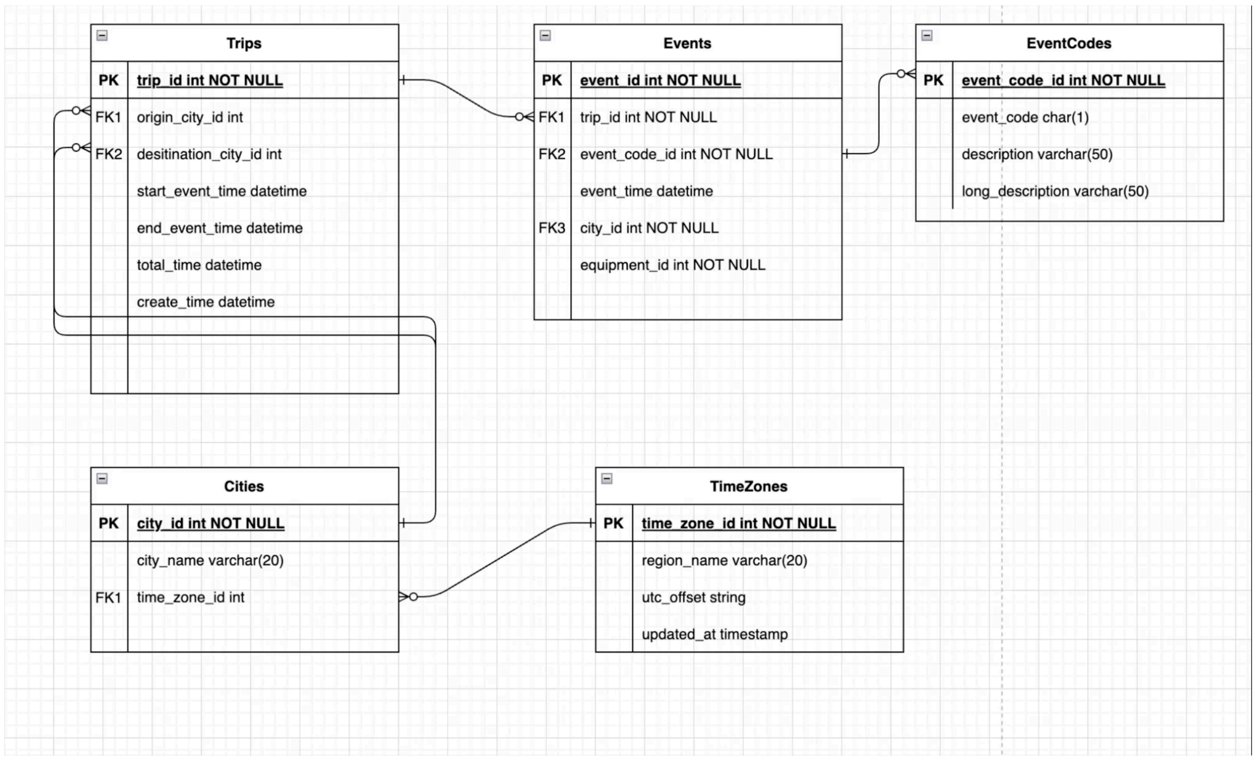
- The system follows a B/S (Browser/Server) architecture, where users interact with the application through a web browser and all business logic is handled on the server side.

Frontend

- The frontend is developed using Blazor WebAssembly, providing a rich, interactive user interface.
- It communicates with the backend through RESTful APIs.
- The UI is responsible for file uploads, displaying processed trips in a grid, and showing trip event details.

Backend

- The backend is implemented as a RESTful API using C# and ASP.NET Core.



- It handles CSV file processing, time zone conversion, trip generation logic, and validation.
- Entity Framework Core is used as the ORM to manage database access and persistence.

Database

- The system uses MySQL as the relational database.
- It stores cities, equipment events, trips, and related metadata.
- Trips act as parent entities with associated equipment events.

Authorization/Authentication (**ToDo**)

- JWT or bear token

Source Control

- All source code is managed in GitHub, enabling version control, collaboration, and CI/CD integration.

Cloud Platform(**ToDo**)

- The application is deployed on Microsoft Azure, leveraging cloud services for hosting, scalability, and reliability.

Containerization(**ToDo**)

- Docker is used to containerize the frontend, backend, and supporting services.
- This ensures consistent environments across development, testing, and production.

Infrastructure as Code(**ToDo**)

- Terraform is used to define and provision Azure infrastructure.
- This includes resources such as application services, databases, networking, and container registries.

Project manger:(ToDo)

- Methodology: Agile
- Framework: Scrum
- **Sprint Management:** Jira-based sprint planning and tracking

Testing:

Unit Testing: The NUnit framework is used to implement unit tests for backend business logic and services, while bUnitis used to test Blazor UI components at the component level.

Integration Testing: Integration tests are performed using mocked dependencies with Moq, and RESTful APIs are validated using Postman to ensure correct communication between frontend and backend components.

Product (End-to-End) Testing(ToDo): Selenium is used to automate end-to-end tests of the application, simulating real user workflows such as file uploads, viewing trips in the grid, and selecting trips to display their events, ensuring that the system functions correctly across the full stack and in multiple browsers.

Unit Test Cases – Backend

1. CSV Parsing
   - Verify that the CSV file is correctly parsed into Event objects.
2. Trip Creation Logic
   - Ensure trips start with a W (Released) event and end with a Z (Placed) event.
   - Verify correct calculation of TotalTripTime.
3. Event Ordering
   - Ensure events are correctly ordered by UTC time for each equipment.
4. Data Persistence
   - Verify trips and associated events are correctly saved to the database.
5. Time Zone Conversion
   - Confirm local event times are correctly converted to UTC based on city.
6. Validation
   - Detect duplicate events for the same equipment, city, and time.
   - Detect unknown cities or invalid event codes.

Unit Test Cases – Frontend (bUnit)

1. Trips Grid Rendering
   - Verify that the trips grid renders the correct number of rows when given mock trip data.
   - Ensure column headers (Equipment ID, Origin, Destination, Start, End, Hours) display correctly.
2. Trip Selection
   - Test that selecting a trip triggers the display of its associated events.
3. Empty States
   - Ensure proper UI messages appear when there are no trips or no events.
4. File Upload Component
   - Validate that uploading a file triggers the appropriate backend API call.
5. UI Interactions
   - Verify buttons, links, and other interactive elements behave correctly.