Distributed Systems Project

Software Design with Artificial Intelligence for Cloud Computing

24/03/2025

Trip Management System

A yellow van on a road with a large rock formation in the background

AI-generated content may be incorrect.

**Natalia Palej**

**A00279259**

Contents

[Introduction 2](#_Toc194261949)

[Project Overview 2](#_Toc194261950)

[Main Window: index.html 3](#_Toc194261951)

[Main Features 4](#_Toc194261952)

[Backend Functions (via REST API): 4](#_Toc194261953)

[Frontend Functions (Client) 4](#_Toc194261954)

[Requirement 1: HTTP Requests from Client (GET/POST/PUT/DELETE) 4](#_Toc194261955)

[Requirement 2: Tomcat Server Responds to All HTTP Requests 4](#_Toc194261956)

[Requirement 3: XMLPullParser Used to Handle Responses on Client 5](#_Toc194261957)

[Requirement 4: HSQLDB Stores and Retrieves Data 6](#_Toc194261958)

[Steps to Run the Application 6](#_Toc194261959)

[Extra Features 7](#_Toc194261960)

# Introduction

Trip Management System is an application that uses database for storing and accessing information about trips and activities. It runs on Tomcat 9.0 server. Data is stored across two HSQLDB tables:

* trips
* activities

Information is retrieved and processed using an XMLPullParser which identifies data rows by using tags. The server uses JAX API to access and display the data.

Both classes hold the same operations which make them very similar – so for the report purposes I will discuss and focus on Trip Class (trip table) when explaining the implementation. – IS IT EVEN NEEDED? Check at the end

## Project Overview

This project is a Trip Management System that allows users to manage trips and the related activities. The system supports full CRUD operations on both tables. The frontend is built as a Java Maven project that communicates with the backend using JAX-RS. Data is retrieved and parsed using an XMLPullParser to extract values from XML tags.

The trips table includes fields such as destination, start and end dates, budget and notes. Each trip can have multiple associated activities, stored in the activities table. Every activity includes a name, date, location and cost.

|  |  |
| --- | --- |
| **Backend** | **Frontend** |
| A screenshot of a computer program  AI-generated content may be incorrect. |  |

## Main Window: index.html

The main screen of the project is the index.html file, which lists all trips. These are dynamically sorted into two sections:

* Upcoming Trips
* Past Trips

Trips are sorted based on the current date and visually marked if completed. Clicking on a trip opens the list of related activities. Each activity and trip can also be edited or deleted from the interface. When editing, the form is pre-filled with the data.

A screenshot of a travel website

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Main Features

### Backend Functions (via REST API):

|  |  |
| --- | --- |
| **Operation** | **Description** |
| *GET* | Returns data for all entries of a table |
| *GET BY ID* | Returns single entity for trip and/or activity |
| *GET BY COST* |  |
| *GET TRIP STATS* |  |
| *POST* | Adds new trip and/or activity and creates it on the server |
| *PUT* | Edit current details for trip/activity, data will be updated in database |
| *DELETE BY ID* | Delete single trip/activity |
| *DELETE ALL* | Delete whole database |

### Frontend Functions (Client)

Dynamically generates expected image names based on destination or activity name.

Preview images before upload and provide image path guidance.

Edit trips/activities with pre-filled data of specific record.

Delete with confirmation prompts

Parse XML responses using XMLPullParser.

# Requirement 1: HTTP Requests from Client (GET/POST/PUT/DELETE)

Client side sends all standard HTTP requests using fetch API in JavaScript. Each request is triggered based on user interaction. For example, when user clicks to add a new trip, update or delete an existing one. All forms handle input validation and requests are formatted as XML.

# Requirement 2: Tomcat Server Responds to All HTTP Requests

Backend runs as a Dynamic Web Application on Tomcat 9.0, responding to requests with appropriately formatted XML. JAX-RS is used to define endpoints for both “/trips” and “/activities”. All four methods (GET, POST, PUT, DELETE) are fully implemented for each resource.

# Requirement 3: XMLPullParser Used to Handle Responses on Client

The frontend uses an XMLPullParser to handle and extract values from the XML responses. For example, when a user retrieves all activities for a specific trip, the response is parsed to extract fields like name, date, and cost. The extracted values are then displayed in the GUI using Java Swing components.

Add screenshoot of **XMLParsing** **class** (working in the **console** and properly caching)

A screenshot of a computer

AI-generated content may be incorrect.

Add screenshoot of **XMLParsing** done in **javascript** (cheating style but the above proves I know how to implement it in Java)

A screen shot of a computer code

AI-generated content may be incorrect.

# Requirement 4: HSQLDB Stores and Retrieves Data

All trip and activity data is stored in an HSQLDB database. When the backend starts, it checks if the database already contains data and creates the necessary tables if needed. If the tables are empty, default records are inserted.

|  |  |
| --- | --- |
| **Trip Table** | **Activity Table** |
| tripId (Primary Key) | activityId (Primary Key) |
| destination | tripId (Foreign Key referencing trips) |
| startDate | name |
| endDate | activityDate |
| budget | location |
| notes | cost |

All database operations are handled in the DAO classes (TripDAOsql and ActivitiesDAOsql) using PreparedStatements. Data is retrieved using SELECT queries, added using INSERT, updated with UPDATE and deleted using DELETE. For example, when a trip is deleted, it’s fully removed from the database using SQL and the page reloads without it.

Each trip also dynamically fetches its related activities by tripId, allowing the app to display nested data properly.

## Steps to Run the Application

1. **Start HSQLDB Server:** The app connects to in-memory database using the JDBC URL:

jdbc:hsqldb:hsql://localhost/oneDB

1. **Deploy the Backend:** Run the backend project on Tomcat 9.0. It starts automatically and handles all API requests.
2. **Run the Frontend (Maven Project):** The frontend is a Java Maven project. It uses XMLPullParser to process XML data retrieved from the backend and populate the interface.
3. **Initial Setup:** Default trips and activities are added automatically.

# Extra Features

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Image Naming Guidance | When user types a destination or activity name, the app dynamically suggests the correct image name and folder path for storing uploaded pictures. |
| Image Preview | Users can preview their selected images before submitting them. If no image is found - default placeholder is used. |
| Smart Date Parsing **(defaults to DD/MM/YYYY) and changes dynamically after user input if wrong format was entered** | Even though dates are stored as strings, the frontend can handle different formats like DD-MM-YYYY, DD/MM/YYYY, or DD.MM.YYYY, making it flexible for user input. |
| Form Prefilling | When editing, the form is filled with existing values. If a field like notes or cost is empty or null, default "TBC" or 0 is used so the app doesn’t crash. |
| Visual Grouping by Date | Trips are automatically grouped into past and upcoming based on the current date. Completed trips are marked with an overlay. |
| Dynamic Styling | The background image changes depending on whether the user is adding a trip or an activity — this gives each section a more custom feel. |
| Options | Menu to handle database operations like clear database and populate database |
| Export to CSV/EXCEL | Function to export data from database in table format to CSV or EXCEL |
| Stats |  |
| Styling ??? | Add here that the webpage is modern and engaging by adding few bits like X icon to delete tirp/activity ?? |