Assignment 2: Time-Based Pricing Engine

1. Overview

This document describes the design and implementation of **Assignment 2**, which involves developing a **time-based pricing engine** using **Spring Boot**.
The application reads pricing data from a TSV file, stores it in memory, and exposes REST APIs to retrieve the correct price for a product based on a given time.

The solution is designed to demonstrate:

- File handling and parsing
- In-memory data storage
- Time-based business logic
- RESTful API development
- Test Driven Development (TDD)

2. Problem Description

The system consumes a **TSV (pipe-separated) file** containing pricing information for different products.
Each record defines a price that is valid only within a specific time range.

Key characteristics:

- A single SKU may appear multiple times
- Each SKU may have multiple pricing windows
- The applicable price depends on the request time

The application must determine the correct price for a given SKU at a given time or return an appropriate response if no price is applicable.

3. Functional Requirements

3.1 File Processing

- The application must read a TSV file from the application resources.
- The file must be parsed line by line.
- The header row must be ignored.
- Parsed data must be stored in memory for fast retrieval.

3.2 Price Retrieval

- An API must accept a SKU ID as a mandatory parameter.
- An optional time parameter may be provided.
- If a valid pricing window exists for the given time, the corresponding price must be returned.
- If no valid pricing window exists, the response must be NOT SET.

4. Application Architecture

The application follows a **layered architecture**:

- **Controller Layer**
  Handles incoming HTTP requests and outgoing responses.
- **Service Layer**
  Contains the business logic for loading pricing data and computing applicable prices.
- **Model Layer**
  Represents pricing data using domain objects.

No database layer is used, as the assignment explicitly requires in-memory storage.

5. Data Model and In-Memory Design

5.1 In-Memory Storage Strategy

Pricing data is stored using a map-based structure where:

- The **key** is the SKU ID
- The **value** is a list of price windows associated with that SKU

This structure enables:

- Constant-time lookup by SKU
- Efficient iteration over pricing windows
- Fast response times without persistence overhead

5.2 Price Window Representation

Each pricing interval is represented by a PriceWindow object containing:

- Start time
- End time
- Price

The LocalTime class from the Java Time API is used to ensure accurate time comparison.

6. REST API Specification

6.1 Load Pricing Data API

Purpose:
Loads pricing data from a TSV file into memory.

Endpoint:
POST /pricing/load

Query Parameter:

- filePath – Relative path of the TSV file inside the resources directory

Behavior:

- Reads the file using the application classloader
- Parses each row into pricing windows
- Stores the data in memory

Response:
A success message confirming that the file has been loaded.

6.2 Price Lookup API

Purpose:
Returns the applicable price for a given SKU and time.

Endpoint:
GET /pricing/price

Query Parameters:

- skuid (mandatory)
- time (optional)

Behavior:

- If the SKU does not exist, returns NOT SET
- If time is not provided, returns NOT SET
- If time falls within a valid pricing window, returns the corresponding price
- If multiple pricing windows overlap, the most recently defined window is applied

7. Business Logic Rules

- Pricing windows are evaluated with:
    - Start time inclusive

    - End time exclusive
- Time values are parsed using a flexible formatter to support both H:mm and HH:mm formats.
- Extra whitespace in TSV data is trimmed to ensure consistent SKU matching.
- Invalid inputs never cause application crashes and safely return NOT SET.

8. Error Handling and Edge Cases

The application safely handles:

- Missing or invalid SKU IDs
- Missing or malformed time values
- Whitespace inconsistencies in input files
- Requests outside all pricing windows

All parsing and validation logic is designed to prevent runtime exceptions.

9. Testing Strategy (TDD)

The application follows **Test Driven Development (TDD)** principles.

9.1 Service Layer Tests

- Verify correct loading of TSV data
- Confirm in-memory storage population
- Validate correct price selection for valid times
- Validate NOT SET responses for invalid cases

9.2 Controller Layer Tests

- Verify endpoint mappings
- Ensure controller invokes service layer methods
- Validate HTTP response status codes

JUnit 5, Mockito, and MockMvc are used for testing.

10. Assumptions and Limitations

- All data is stored in memory and is lost on application restart
- Pricing windows do not span across midnight
- The TSV file format is assumed to be structurally valid

- No authentication or authorization is implemented

11. Conclusion

This assignment demonstrates a clean and efficient implementation of a time-based pricing engine using Spring Boot.
By using in-memory storage, structured data modeling, and robust validation logic, the application delivers fast and predictable results.

The solution adheres to industry best practices and fulfills all functional and technical requirements of the assignment.