Milestone 1 Evaluation Documentation

Introduction

This document outlines the evaluation process for Milestone 1. The milestone focuses on assessing the foundational understanding of Node.js concepts taught in the syllabus. The evaluation will be based on students' ability to conceptualize, design, and implement functional backend APIs for real-world mini-projects.

Evaluation Goals

- **Technical Proficiency**: Validate the students’ understanding of Node.js fundamentals, asynchronous programming, file handling, and data structures.
- **Problem-Solving Skills**: Assess their ability to implement real-world backend functionalities using efficient logic and clear coding practices.
- **API Development**: Test their ability to design, build, and document RESTful APIs.
- **Practical Application**: Ensure the integration of taught concepts into functional, modular mini-projects.

Mini-Projects for Evaluation

The students will choose or be assigned one of the following mini-projects, designed to evaluate their grasp of both the technical and practical aspects of backend development:

Below is a detailed **Problem Statement** for the selected mini-projects. Each section includes a problem overview, features, requirements, and the expected solution design for clarity and documentation purposes.

1. Personal Expense Tracker API

Overview

Managing daily expenses is crucial for budgeting and financial planning. The **Personal Expense Tracker API** helps users log, view, and analyze their expenses. It provides insights into spending patterns and assists in better decision-making.

Features

1. Add a new expense with details such as category, amount, and date.
2. Retrieve a summary of expenses filtered by category or date range.
3. Analyze spending patterns, such as the highest spending category or monthly totals.
4. Generate a weekly or monthly summary report using automated tasks.

Requirements

- **Endpoints**:
  - Add Expense (`POST /expenses`): Log a new expense.
  - Get Expenses (`GET /expenses`): Retrieve expenses based on filters like category or date range.
  - Analyze Spending (`GET /expenses/analysis`): Get insights like total by category or time.
  - Generate Summary (`CRON job`): Automated reports for daily / weekly / monthly expense summaries.
- Data Validation (Optional) :
  - Expense categories (e.g., "Food", "Travel") must be predefined.
  - Amount must be a positive number.
- Response Format: JSON structured as `{ status, data, error }`.

Solution Design

- Use arrays for in-memory storage.
- Implement sorting and filtering logic to retrieve and group expenses.
- Leverage `node-cron` for generating summary reports.

2. Real-Time Event Notifier API

Overview

This API enables users to manage and receive real-time notifications for events such as meetings, deadlines, or reminders. It ensures users stay updated about important tasks without manually tracking them.

Features

1. Create events with a title, description, and scheduled time.
2. Notify users via WebSockets when the event is about to start.
3. Log completed events for future reference.

Requirements

- **Endpoints**:
  - Add Event (`POST /events`): Create a new event with title, description, and time.
  - Get Events (`GET /events`): Fetch all upcoming events.
- Real-Time Notifications:
  - Use WebSocket / cron-job to notify users 5 minutes before an event starts.
  - Notify if events overlap.
- Logging:
  - Save completed events asynchronously to a file for historical data.

Solution Design

- Implement event storage with time-based sorting for efficient scheduling.
- Use `node-cron` to poll events and trigger notifications.
- Integrate WebSocket using the `ws` package for real-time updates.

3. Food Delivery Backend with Order Management

Overview

The **Food Delivery Backend** provides APIs for managing restaurant menus, handling orders, and simulating order status updates, helping build the foundation for a food delivery service.

Features

1. Add menu items with details like name, price, and category.
2. Place orders by selecting multiple items from the menu.
3. Track the status of an order, which automatically updates over time.

Requirements

- **Endpoints**:
    - Add Menu Item (`POST /menu`): Create or update menu items.
    - Get Menu (`GET /menu`): Retrieve menu items.
    - Place Order (`POST /orders`): Create an order with selected menu items.
    - Get Order (`GET /orders/:id`): Fetch details of a specific order.
    - Update Status (`CRON job`): Automate status updates from `Preparing` → `Out for Delivery` → `Delivered`.
- Data Validation:
    - Validate menu item fields like price (positive) and category (predefined).
    - Validate order requests to ensure item IDs exist.

Solution Design

- Use a queue for managing order statuses and processing.
- Store menu items and orders in separate in-memory arrays or collections.
- Integrate `node-cron` to simulate periodic status updates.

4. Mini Search Engine for Articles

Overview

The **Mini Search Engine** enables users to upload and search articles efficiently. This backend mimics the behavior of a simple search engine by supporting keyword searches and relevance-based sorting.

Features

1. Add articles with a title, content, and tags.
2. Search articles by keywords in the title or content.
3. Sort search results by relevance or date.

Requirements

- **Endpoints**:
  - Add Article (`POST /articles`): Add a new article with metadata.
  - Search Articles (`GET /articles/search`): Search articles by keyword or tag.
  - Get Article (`GET /articles/:id`): Retrieve full article details by ID.
- **Indexing**:
  - Maintain an in-memory index for quick searches.
  - Calculate relevance using keyword frequency.

Solution Design

- Use arrays to store articles with indexing for fast searches.
- Implement search logic with text matching and sorting by relevance.
- Use `fs` for optional persistence of articles.

5. Smart Habit Tracker with Daily Updates

Overview

The **Smart Habit Tracker** helps users maintain daily habits by providing a platform to log progress and receive reminders. It encourages consistency through automated updates and analytics.

Features

1. Add habits with a name and daily goal (e.g., "Drink 8 glasses of water").
2. Mark habits as complete for the day.
3. Send daily reminders via WebSocket.
4. Track weekly completion data and visualize progress.

Requirements

- **Endpoints**:
  - Add Habit (`POST /habits`): Create a new habit with its daily goal.
  - Update Habit (`PUT /habits/:id`): Mark a habit as complete for a day.
  - Get Habits (`GET /habits`): Fetch all active habits and their completion status.
  - Weekly Report (`GET /habits/report`): Generate a progress report for the week.
- Daily Notifications:
  - Use WebSocket / Cron job to send reminders for incomplete habits at a scheduled time.

Solution Design

- Store habits in an array or database with tracking by date.
- Implement CRON jobs to trigger daily reminders.
- Allow optional logging of weekly reports to a file for historical analysis.

General Notes for All Projects

1. **Development Environment**:
   - Use Node.js with Express for API development.
   - Optionally use `fs` for file-based persistence or SQLite for lightweight data storage.
2. **Error Handling**:
   - Ensure proper validation and error messages for invalid inputs.
   - Use consistent response formats (`{ status, data, error }`).

Submission Guidelines

1. **Project Code**: Submit the complete project code in a GitHub repository. Ensure the repository includes a README.md file detailing how to set up and test the API.
2. **API Documentation**: Provide a well-structured document or Postman collection outlining the API endpoints, request formats, and expected responses.

Milestone Outcome

The outcome of Milestone 1 will determine whether students can proceed to the next milestone. Successful completion requires:

- Scoring at least **70% overall**.
- Meeting all **functional requirements** for the assigned mini-project.