IKUSASA TECHNOLOGY SOLUTIONS DEVELOPER ASSESSEMENT

Assessment Task 1: Build a Smart Task Manager API

Overview: Create a RESTful API for a task manager application that includes basic task management features and integrates AI-driven capabilities for automation.

Language: Any programming language (e.g., JavaScript, Python, Java, C# etc.)
Tools: Any frameworks or libraries you are is comfortable with.
Database Driver: Any relational or NoSQL database (e.g., MySQL, PostgreSQL, MongoDB, etc.)
Submission: Source code (e.g., GitHub repo).

Requirements:

1. Endpoints:

- `POST /tasks`: Create a new task (body: title, description, due date).
- `GET /tasks`: Retrieve all tasks.
- `GET /tasks/{id}`: Retrieve a single task by ID.
- `PUT /tasks/{id}`: Update a task (body: title, description, due date).
- `DELETE /tasks/{id}`: Delete a task by ID.

2. Data Model:

- Tasks should have the following fields:
  - `id` (unique identifier)
  - `title` (string)
  - `description` (string)
  - `due_date` (date)
  - `completed` (boolean)

- `priority` (string: low, medium, high)

3. AI Features:

- Smart Task Suggestions: Implement an endpoint that suggests tasks based on user input. For example, when creating a task, suggest a title or description using a simple AI model (like text completion or keyword extraction).

- `POST /tasks/suggest`: Takes user input and returns suggested titles or descriptions.

- Due Date Prediction: Use a basic algorithm to predict a due date based on historical task completion data.

- `POST /tasks/{id}/predict-due-date`: Takes task details and predicts an optimal due date based on past tasks' completion times.

4. Automation Features:

- Reminder Notifications: Implement a background job (using a cron job or similar) that checks for tasks due soon (e.g., within the next 24 hours) and sends reminders (could be simulated with console logs or an external service).

- Recurring Tasks: Allow users to set tasks as recurring (daily, weekly, monthly) and automate the creation of these tasks.

- Add a `recurrence` field to the task model.

5. Database:

- Store tasks in a database of your choice. Ensure tasks can be retrieved and manipulated correctly, and consider how to structure the data for recurring tasks.

Evaluation Criteria

- **Functionality:** Does the API meet the enhanced requirements?

- Code Quality: Is the code clean, readable, and well-organized?

- AI Integration: How effectively are the AI features implemented? Are they relevant and functional?

- Database Design: Is the database schema logical and efficient, especially for automation features?

- Error Handling: Are appropriate error messages returned for invalid requests?

- Documentation: Is there clear documentation on how to run the project and its structure, particularly around the AI and automation features?

Task 2: Build a Creative and Responsive User Interface for the Smart Task Manager

Overview: Create a visually appealing, responsive user interface (UI) for the Smart Task Manager API developed in Task 1. The UI should offer a seamless user experience while incorporating innovative design elements and utilizing the API effectively.

Requirements:

1. Technology Stack:

- Use any front-end framework or library (e.g., React, Angular, Vue.js) or vanilla HTML/CSS/JavaScript.

- Ensure the UI communicates with the API from Task 1.

2. Design Requirements:

- Creativity: Implement a unique design that goes beyond standard layouts. Use animations, transitions, and color schemes that enhance user engagement.

- Responsiveness: Ensure the UI is mobile-friendly. It should adapt gracefully to different screen sizes using responsive design techniques (like media queries or a CSS framework).

- User Experience: Incorporate intuitive navigation, clear calls to action, and feedback (like loading indicators and success/error messages).

3. Features:

- Task List: Display tasks in a visually engaging manner (e.g., card layout) with key information (title, description, due date, status).

- Task Creation: Provide a form for creating new tasks, utilizing the AI suggestion feature for title and description. Use a modal or a dedicated page for this functionality.

- **Task Editing:** Allow users to edit existing tasks with a similar form design. Consider inline editing for enhanced user experience.

- **Task Deletion:** Implement a confirmation dialog for task deletion to prevent accidental removals.

- **Due Date Prediction:** Show predicted due dates in a visually distinct way (e.g., tooltip or badge).

- **Recurring Tasks:** Allow users to set recurring tasks and display them accordingly with clear visual indicators.

4. AI Features Integration:

- Integrate the AI suggestion feature into the task creation form to provide real-time suggestions.

- Allow users to view predicted due dates as part of the task details.

5. Deployment:

- Deploy the application on a platform like Vercel, Netlify, or GitHub Pages. Ensure it's easily accessible and provide the link in your submission.

6. Testing:

- Include basic tests (if applicable) to ensure key functionalities work as intended.

Evaluation Criteria

- Creativity: How innovative is the design? Are unique elements incorporated?

- Responsiveness: Does the UI function well on various devices and screen sizes?

- Functionality: Are all required features implemented and functioning correctly?

- Integration: How well does the UI communicate with the API? Are API calls handled effectively?

- User Experience: Is the interface intuitive and engaging? Are feedback mechanisms well-implemented?

- Code Quality: Is the code organized, maintainable, and easy to read?

- Deployment: Is the application properly deployed and accessible?