***Task Management***

1. User Interface (UI) Components

The user interface components will be implemented using Angular and TypeScript. The UI will include the following screens and features:

* Registration Page: A form where users can provide their details to register a new account.
* Login Page: A form for users to enter their credentials and log into the system.
* Dashboard: The main page after login, displays the user's task list, options to create new tasks, and modify and delete existing ones.
* Task Details Page: A page showing the details of a specific task, including its status, comments, attachments, and related information.
* Task Creation/Modification Form: Users can enter task details such as title, description, due date, and priority. If it is an existing one it will get updated and if it is new it will get saved.
* Task Filtering and Sorting Options: UI components allow users to filter and sort tasks based on different criteria.
* Notifications Panel (Optional): A panel displaying real-time notifications for task assignments, updates, and reminders.

1. Backend Components

The backend components will be implemented using the Spring Boot framework and various libraries, microservices, and modules:

* Task Service: Handles the creation, updating, and retrieval of tasks. Implements logic for task assignments, due dates, priorities, and dependencies.
  + API’s:
    - Create Task `tms/api/v1/tasks`: create a task using a post request.
    - List Tasks `tms/api/v1/tasks`: get a list of tasks using get request and using this query params we can filter and sort the data.
    - Get a Task by id `tms/api/v1/tasks/{id}`: get a task using a unique id by get request.
    - Update a Task `tms/api/v1/tasks/{id}`: update a task using a put request.
    - Delete a Task `tms/api/v1/tasks/{id}`: delete a task using a delete request and unique id.
  + Data Access Objects:
    - Task: An entity class used to perform crud operations related to tasks on db. Attributes of the class are:
      * Id: unique id.
      * Title: title of the task.
      * Description: description of the task.
      * Due date: due date of the task.
      * Status: status of the task.
      * User id: user unique id who created the task.
  + Services:
    - Task Service: The class which is used to implement business logic.
  + Models:
    - Create Task: Model class which is used as a request payload for task creation.
    - Task Response Model: Class which is used as a response for task API.
  + Configuration:
    - Authentication Conf: The configuration which is used to authenticate API calls before responding to a request.
* User Service: Handles user registration, authentication, and profile management. Integrates with Spring Security for user authentication and authorization.
  + API’s:
    - Register User `auth/api/v1/sign-up`: Register user using valid payload and post request.
    - Login User `auth/api/v1/sign-in`: Login user using valid payload and post request.
    - Validate Token `auth/api/v1/validate`: Validate JWT token using get request.
  + Data Access Objects:
    - User: An entity class used to perform crud operations related to tasks on db. Attributes of the class are:
      * Id: unique id.
      * Username: unique username of a user.
      * Email: unique email address of a user.
      * First Name: User first name.
      * Last Name: User last name.
      * Password: Hashed password.
  + Models:
    - Auth Model: The class is used as a response model for sign-in and sign-up API which consists of user details and a JWT token.
    - Token Model: The class which is used as a response model for validating API which consists of user details,` issued at`, and expiry timestamp.
    - Login Model: The class which is used as a request model for sign-in API which consists of email and password.
    - Register Model: The class which is used as a request model for sign-up API which consists of user details.
  + Security Configuration: Configure Spring Security to handle user authentication and authorization using JWT
* Notification Service: Sends email notifications for task assignments, updates, and reminders. Utilizes third-party email service providers asynchronously.
  + Services
    - Cronjob Service: service which will send emails to users if the due date of any task is near or get overdue.
    - Get details service: service which will send requests to task service and auth service to fetch data.
* Error Handling and Logging: The Task Management System will implement robust error handling and logging mechanisms. Custom exception handling will be implemented to catch and process exceptions raised during runtime. Global exception handlers will be used to centralize error handling and provide appropriate error responses to clients. The backend components will also generate log statements at various levels (e.g. INFO, DEBUG, WARN, ERROR) to capture important events, error details, and diagnostic information. Logging frameworks such as Logback or Log4j will be used to capture and manage these log statements. The logs will be stored in a centralized location or analyzed using log management tools like ELK Stack or Splunk, allowing for efficient log storage, search, analysis, and troubleshooting.
* Caching: To improve performance and reduce database load, caching mechanisms can be implemented in the backend. Spring Cache can be utilized to cache frequently accessed data, such as task lists, user profiles, or authentication tokens. Caching can help minimize the number of database queries and enhance the overall system responsiveness.

1. Error Handling and Logging

The system will implement proper error handling and logging mechanisms. This includes:

* Custom Exception Handling: Implement global exception handlers to handle and process exceptions raised during runtime.
* Logging: Utilize logging frameworks such as Logback or Log4j to capture application logs. Configure logging levels and log messages to provide insights into system behavior, errors, and important events.
* Centralized Logging: Consider using a centralized logging system like ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk to aggregate and analyze logs from multiple instances.

1. Testing Strategy

The testing strategy for the Task Management System may include the following types of tests

* Unit Testing: Write unit tests to verify the functionality of individual components such as controllers, services, and data access objects.
* End-to-End Testing: Conduct end-to-end tests to simulate user interactions and test the system's behavior as a whole.

1. Deployment and Scalability

The Task Management System can be deployed on a cloud platform like AWS, Azure, or Google Cloud. The deployment strategy may involve:

* Containerization: Package the application using Docker containers for easy deployment and scalability.
* Container Orchestration: Utilize a container orchestration platform like Kubernetes for managing scalability, high availability, and automatic scaling of the application.
* Load Balancing: Use load balancers to distribute incoming traffic across multiple instances of the application for improved performance and scalability.
* Continuous Integration and Deployment (CI/CD): Implement CI/CD pipelines to automate the build, testing, and deployment processes.