Tasks: -

1.Requirements Analysis

• Identify and list 5 key functional requirements for the task management system.

• Specify 1 non-functional requirement, such as scalability or security.

2. Software Design

• Design a high-level architecture diagram for the system, including core components like a database, backend services, and frontend.

• Describe each component briefly.

3. Implementation Details

• Write pseudocode for a core functionality, such as adding a task or listing all tasks.

• Use proper naming conventions and comments in your pseudocode.

4. Testing and Tools

• Suggest 2 testing strategies to ensure the quality of the system (e.g., unit testing, integration testing).

• List 2 tools you would use during development, such as Git for version control or Jenkins for CI/CD.

Functional Requirements:

❖ Users should be able to create new tasks with details such as title, description, due date, and priority.

❖ Users should be able to view a list of all tasks with filtering options (e.g., by priority or due date).

❖ Users should be able to edit task details.

-: Non-Functional Requirement: -

❖ The system should handle up to 10,000 concurrent users without performance degradation

: Software Design: -

High-Level Architecture Diagram:

Components:

✓ ***Frontend:*** A web-based user interface for users to interact with the task management system

✓ ***Backend:*** Handles business logic and processes user requests. ✓ Database: Stores user and task data persistently.

✓ ***Authentication Service:*** Ensures secure access to user-specific data. Diagram Description:

✓ ***Frontend:*** Built using a framework like React.js to provide a responsive UI.

✓ ***Backend:*** Developed with Node.js, exposing APIs for task management operations.

✓ ***Database:*** A relational database (e.g., PostgreSQL) for structured storage of tasks and user data.

✓ ***Authentication Service:*** Handles user login and token-based authentication using OAuth 2.0 or JWT

Database Diagram:- **Diagram Description**

Frontend	Framework (React , Java)
Backend	Developed with (Node Js)
Database	Relational Database(Postgre SQL)
Authentication Services	Handler user base (Signup && Login)

-:Implementation Details:-

Pseudocode for Adding a Task:-

```
# Function to add a new task
function addTask(userId, title, description, dueDate, priority):
    # Validate input fields
    if title is empty or dueDate is invalid:
        return "Error: Invalid input"

    # Create a new task object
    task = {
        "id": generateUniqueId(),
        "userId": userId,
        "title": title,
        "description": description,
        "dueDate": dueDate,
        "priority": priority,


        "status": "Pending"
    }

    # Save task to database
    database.save(task)

    # Return success message
    return "Task successfully added"
```

-: Testing and Tools: -

Testing Strategies:

✓ **Unit Testing**: Test individual components (e.g., add Task function) to ensure they work as expected.

✓ **Integration Testing:** Test interactions between frontend, backend, and database to ensure seamless communication

Tools:-

✓ Git: For version control to track changes and collaborate with team members.

✓ **Jenkins**: For Continuous Integration/Continuous Deployment (CI/CD) to automate testing and deployment processes