# TASK MANAGEMENT SYSTEM

***SDLC Based Project Report with Code Overview***


## Abstract

This project presents a scalable Task Management System developed using Node.js, Express, MongoDB, and React.js. The system implements secure authentication using JWT, role-based access control, and RESTful APIs. The project follows the Software Development Life Cycle (SDLC) methodology, ensuring structured development, scalability, and security.


## 1. Software Development Life Cycle (SDLC)

**Requirement Analysis:** Identification of user roles, authentication needs, and CRUD operations.
**System Design:** Designing REST APIs, database schemas, and frontend-backend interaction.
**Implementation:** Developing backend APIs and frontend UI using modern web technologies.
**Testing:** Manual API testing using Postman and Swagger UI.
**Deployment & Maintenance:** Docker-ready and scalable cloud deployment architecture.

## 2. System Architecture

The system follows a client-server architecture. The React frontend communicates with the Node.js backend through REST APIs. MongoDB is used as the database layer. JWT ensures secure, stateless communication between client and server.

## 3. Database Design

**User Collection:** name, email, password, role
**Task Collection:** title, description, status, createdBy

## 4. Module Description

**Authentication Module:** Handles user registration and login using bcrypt and JWT.
**Authorization Module:** Implements role-based access using middleware.
**Task Management Module:** Supports CRUD operations on tasks.
**Frontend Module:** React UI for interaction with APIs.

## 5. Code Snippets (Overview)

### *JWT Authentication Middleware*

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(401).json({ message: "Unauthorized" });

  const decoded = jwt.verify(token, process.env.JWT_SECRET);
  req.user = decoded;
  next();
};
```

### *Task CRUD Controller (Sample)*

```
exports.createTask = async (req, res) => {
  const task = await Task.create({
    title: req.body.title,
    createdBy: req.user.id
  });
  res.status(201).json(task);
};
```

## 6. Security & Scalability

The system uses bcrypt for password hashing, JWT for stateless authentication, and role-based access control for authorization. The modular architecture supports horizontal scaling, caching using Redis, containerization using Docker, and CI/CD integration.

## 7. Conclusion

This project demonstrates a complete backend system with authentication, authorization, CRUD functionality, and frontend integration. Following SDLC principles ensures the system is scalable, secure, and maintainable, making it suitable for real-world applications.