Software Design Document - TaskFlow

1. Introduction to the Project

TaskFlow is a task management application that helps organizations manage employees and their assigned tasks efficiently. It enables managers to oversee projects, assign tasks to employees, and track progress, while employees can manage their assigned tasks and update their status.

The system is designed for both managers and employees within organizations that require structured task management and employee oversight.

2. Overview of System Architecture

TaskFlow employs a microservices architecture comprising two main services:

- **User Service**: Manages user/employee information, authentication, and authorization
- **Task Service**: Handles task creation, assignment, updates, and tracking
- **Frontend Application**: Provides the user interface for interacting with both services

The system follows a client-server model where the frontend communicates with the backend services via RESTful APIs.

3. Details of Each Component

3.1 User Service (Port 8081)

Primary Responsibility: Managing user accounts and employee information

Data Handled:

- User credentials (username, password)

- Employee profiles (name, department, status)

- Role-based access control

Key Features:

- User authentication and JWT token generation

- User registration and profile management

- Employee status tracking

AOP Aspects Applied:

- Logging Aspect: Tracks method execution times

- Exception Handling Aspect: Captures and logs service exceptions

3.2 Task Service (Port 8082)

Primary Responsibility: Managing tasks and their lifecycle

Data Handled:

- Task details (title, description, due date)

- Task status (pending, in progress, completed)

- Task assignments (mapping between tasks and employees)

Key Features:

- Task creation and assignment

- Status updates and tracking

- Task filtering and categorization

AOP Aspects Applied:

- Logging Aspect: Monitors method performance

- Exception Handling Aspect: Provides uniform exception handling

3.3 Frontend Application

Primary Responsibility: Providing user interfaces for different roles

Key Interfaces:
- Authentication screens (login, signup)
- Dashboard views (manager and employee dashboards)
- Task management screens
- Employee management screens
- User profile management

4. Component Interactions

1. **Authentication Flow**:
   - The frontend sends credentials to the User Service
   - User Service validates credentials and returns a JWT token
   - Frontend stores the token and uses it for subsequent API calls

2. **Task Management Flow**:
   - Task Service communicates with User Service (via Feign Client) to validate user existence and permissions
   - When tasks are assigned, Task Service checks with User Service if the employee exists
   - Dashboard statistics are calculated by aggregating data from both services

3. **Data Flow**:
   - Frontend communicates with both services using RESTful API calls
   - Each service maintains its own database for its domain data

  - Services share information through API calls, not direct database access

5. User Interface Description

5.1 Authentication Screens

- **Login Page**: Username/email and password fields with login button

- **Signup Page**: Registration form with basic user information fields

5.2 Manager Interfaces

- **Manager Dashboard**: Overview with employee count, task statistics, and recent activity

- **Employees Page**: List of employees with status indicators and management options

- **Tasks Page**: Comprehensive task management interface with filtering and assignment capabilities

5.3 Employee Interfaces

- **Employee Dashboard**: Personal task summary and statistics

- **My Tasks Page**: List of assigned tasks with ability to update status

- **Calendar View**: Visual representation of task due dates

5.4 Common Interfaces

- **Profile Page**: User information management

- **Task Details**: Detailed view of individual tasks with history and comments

6. Special Requirements

1. **Security Requirements**:

  - JWT-based authentication for secure API access

  - Role-based access control (manager vs employee permissions)

2. **Performance Considerations**:

   - AOP-based logging for monitoring method execution times

   - Exception handling for improved system stability

3. **Scalability**:

   - Microservices architecture allows independent scaling of User and Task services

   - Services communicate via RESTful APIs for loose coupling

4. **Monitoring**:

   - Centralized logging using AOP aspects for troubleshooting

   - Performance monitoring through execution time logging

5. **Cross-cutting Concerns**:

   - Aspect-Oriented Programming (AOP) implementation for logging and exception handling

   - Standardized error handling across services

The design prioritizes separation of concerns through microservices while maintaining a cohesive user experience through the frontend application.