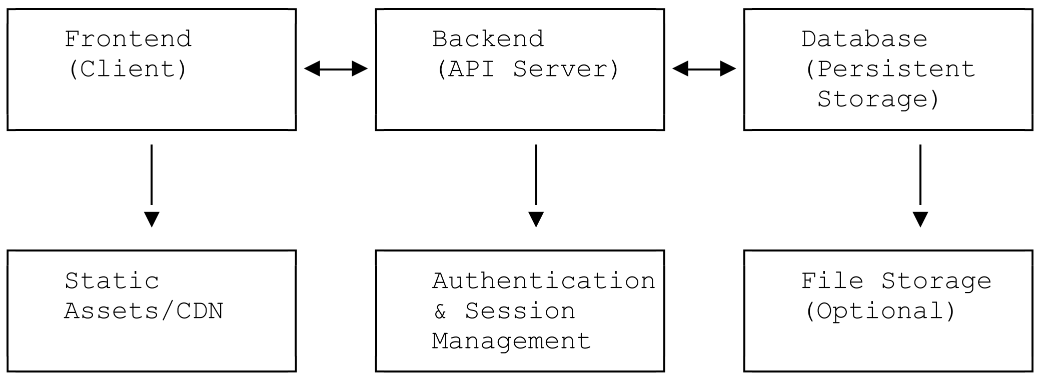
TaskManage - Technical Architecture Overview

Executive Summary

TaskManage is a comprehensive task management application designed to provide users with efficient project and task organization capabilities. This document outlines the technical architecture, system components, data flow, and infrastructure considerations for the application.

System Architecture

High-Level Architecture



Core Components

1. Frontend Layer

Technology Stack:

- Framework: React.js
- UI Components: TailwindCSS
- Build Tools: Webpack / Vite
- Package Manager: npm / yarn

Key Features:

- Responsive design for desktop and mobile
- Drag-and-drop interface for task management
- Dashboard with analytics and reporting
- User profile and settings management

Component Structure:

```
src/
```

```
├── components/
│   ├── common/
│   ├── task/
│   ├── project/
│   └── user/
├── pages/
├── services/
├── utils/
└── assets/
```

2. Backend Layer

Technology Stack:

- Runtime: Node.js
- Framework: Express.js
- Authentication: JWT / OAuth 2.0
- Testing: Jest / Pytest / JUnit

Core Services:

Task Management Service

- CRUD operations for tasks
- Task status management
- Task assignment and delegation
- Due date and reminder handling

3. Database Layer

Primary Database:

- Type: MongoDB
- Purpose: Main application data storage

Database Schema (Relational Model):

```
-- Tasks table
CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    project_id INTEGER REFERENCES projects(id),
    assignee_id INTEGER REFERENCES users(id),
    creator_id INTEGER REFERENCES users(id),
    status VARCHAR(20) DEFAULT 'todo',
    priority VARCHAR(10) DEFAULT 'medium',
    due_date TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Project members table
CREATE TABLE project_members (
    id SERIAL PRIMARY KEY,
    project_id INTEGER REFERENCES projects(id),
    user_id INTEGER REFERENCES users(id),
    role VARCHAR(20) DEFAULT 'member',
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4. API Design

RESTful API Endpoints:

```
Tasks:
GET    /api/tasks
POST   /api/tasks
GET    /api/tasks/:id
PUT    /api/tasks/:id
DELETE /api/tasks/:id
PUT    /api/tasks/:id/status
```

Response Format:

```
{
  "success": true,
  "data": {},
  "message": "Operation completed successfully",
  "timestamp": "2025-07-25T10:30:00Z"
}
```

Security Considerations

Authentication & Authorization

- JWT-based authentication with refresh tokens
- Role-based access control (Admin, Project Manager, Member)
- Password hashing using bcrypt
- Rate limiting on API endpoints

Data Protection

- Input validation and sanitization
- SQL injection prevention using parameterized queries
- XSS protection with content security policies
- HTTPS enforcement for all communications

Privacy & Compliance

- GDPR compliance for user data handling
- Data encryption at rest and in transit
- Regular security audits and updates
- Secure session management

Performance Optimization

Frontend Optimization

- Code splitting and lazy loading
- Image optimization and compression
- Browser caching strategies
- Minification and bundling

Backend Optimization

- Database query optimization
- API response caching
- Connection pooling
- Asynchronous processing for heavy operations

Database Optimization

- Proper indexing strategy
- Query optimization
- Database connection pooling
- Read replicas for scaling

Infrastructure

- **Cloud Provider:** AWS / Google Cloud / Azure
- **Container Orchestration:** Docker + Kubernetes
- **CI/CD Pipeline:** GitHub Actions / GitLab CI / Jenkins
- **Monitoring:** Prometheus + Grafana / DataDog
- **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana)

Technology Stack Summary

Layer	Technology	Purpose
Frontend	React	User interface
Backend	Node.js	Business logic

Data Flow

Task Creation Flow

1. User submits task creation form (Frontend)
2. Frontend validates input and sends POST request to API
3. Backend validates authentication and authorization
4. Backend validates task data
5. Backend creates task record in database
6. Backend returns success response with task data
7. Frontend updates UI with new task

8. **Task Status Update Flow**

1. User updates task status (drag-and-drop or status selector)
2. Frontend sends PUT request to API
3. Backend validates permissions
4. Backend updates task status in database

Scalability Considerations

Horizontal Scaling

- Stateless API servers for easy horizontal scaling
- Database read replicas for improved read performance
- CDN for static asset distribution
- Microservices architecture for future growth

Vertical Scaling

- Database performance tuning
- Server resource optimization
- Caching strategies implementation

Monitoring & Analytics

System Monitoring

- API response times and error rates
- Database performance metrics
- Server resource utilization
- User activity patterns

Business Analytics

- Task completion rates
- Project timeline analysis
- User engagement metrics
- Feature usage statistics

Future Enhancements

Planned Features

- Mobile application (React Native / Flutter)
- Third-party integrations (Slack, Microsoft Teams)
- Advanced reporting and analytics

- AI-powered task recommendations
- Workflow automation
- Time tracking capabilities

Technical Improvements

- Migration to microservices architecture
- Implementation of event-driven architecture
- Advanced caching strategies
- Enhanced security measures
- Performance optimization initiatives

Conclusion

This technical architecture provides a solid foundation for the TaskManage application, ensuring scalability, maintainability, and security. The modular design allows for incremental improvements and feature additions while maintaining system stability and performance.

Appendix

Development Setup

1. Clone the repository
2. Install dependencies
3. Set up environment variables
4. Initialize database
5. Run development servers

API Documentation

- Detailed API documentation available at `/api/docs`
- Postman collection available for testing
- Authentication guide for developers

Deployment Guide

- Step-by-step deployment instructions
- Environment configuration
- Database migration procedures
- Monitoring setup guidelines