Project Management Suite

**Comprehensive Development Plan**

*Using svar.dev Gantt System*

Version 1.0

October 17, 2025

# Executive Summary

This document outlines a comprehensive plan for developing a production-ready project management suite built on the svar.dev Gantt chart system. The application will be a fully modular, enterprise-grade solution featuring team collaboration, intelligent task scheduling, and robust export capabilities.

**Key Objectives:**

* Develop a secure, scalable project management platform with real-time collaboration
* Implement intelligent auto-scheduling with dependency-aware timeline adjustments
* Create a modular architecture for easy maintenance and feature expansion
* Deploy via Docker for consistent, portable deployment on Ubuntu VPS
* Establish clear development rules and quality standards

# Project Overview

## Vision

Create a professional-grade project management suite that combines the power of svar.dev's Gantt visualization with modern team collaboration features, intelligent scheduling, and enterprise-level security.

## Core Principles

* **Modularity First:** Every component is self-contained and independently testable
* **No Mock Data:** All features work with real database operations from day one
* **Production Ready:** Built with security, performance, and scalability as primary concerns
* **Docker Native:** Containerized from the start for consistent deployments

## Target Deployment

Ubuntu VPS with Docker Compose orchestration, supporting horizontal scaling and zero-downtime deployments.

# Technology Stack

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | React 18+ with TypeScript, svar.dev Gantt, TailwindCSS, React Router v6 |
| **Backend API** | Node.js with Express.js, TypeScript, REST API architecture |
| **Database** | PostgreSQL 15+ with connection pooling |
| **Authentication** | JWT tokens with refresh tokens, bcrypt password hashing, rate limiting |
| **ORM** | Prisma ORM for type-safe database operations |
| **Caching** | Redis for session management and API caching |
| **Containerization** | Docker with Docker Compose for multi-container orchestration |
| **Reverse Proxy** | Nginx for SSL termination and load balancing |
| **Export Formats** | PDF (pdfkit), Excel (exceljs), CSV, JSON, PNG (html2canvas) |

# System Architecture & Module Design

## Architecture Overview

The system follows a modular, microservices-inspired monolithic architecture with clear separation of concerns. Each module is independently testable and can be scaled horizontally as needed.

## Core Modules

### 1. Authentication Module

**Responsibilities:**

* User registration with email verification
* Login with JWT token generation and refresh token mechanism
* Password reset via secure email tokens
* Session management with Redis
* Rate limiting to prevent brute force attacks

### 2. User Management Module

**Responsibilities:**

* User profile creation and updates
* Role-based access control (Admin, Manager, Member, Viewer)
* User preferences and settings
* Activity logging and audit trails

### 3. Team Management Module

**Responsibilities:**

* Create and manage teams with hierarchical structures
* Invite members via email
* Assign team roles and permissions
* Team performance metrics and reporting

### 4. Project Management Module

**Responsibilities:**

* Project creation with metadata (title, description, dates, budget)
* Assign teams to projects
* Project templates for rapid setup
* Project archiving and restoration

### 5. Task Management Module

**Responsibilities:**

* Task creation with full metadata (assignees, priority, status, dates)
* Subtask support with unlimited nesting
* Task dependencies (finish-to-start, start-to-start, finish-to-finish)
* Color coding based on status, priority, or custom rules
* Task comments and attachments

### 6. Gantt Chart Module

**Responsibilities:**

* Integration with svar.dev Gantt library
* Real-time rendering of tasks and dependencies
* Drag-and-drop task scheduling
* Zoom levels (day, week, month, quarter views)
* Critical path highlighting

### 7. Auto-Scheduling Engine

**Responsibilities:**

* Automatic timeline adjustments based on task dependencies
* Cascade date changes through dependency chains
* Resource conflict detection
* Working day calculations (exclude weekends/holidays)
* Buffer time calculations for risk management

### 8. Export Module

**Responsibilities:**

* PDF export with customizable templates
* Excel export with multiple sheets (Overview, Tasks, Resources)
* CSV export for data analysis
* PNG image export of Gantt chart
* JSON export for API integrations
* Email delivery of exports

### 9. Notification Module

**Responsibilities:**

* Real-time in-app notifications
* Email notifications for task assignments and updates
* Notification preferences management
* Digest emails for daily/weekly summaries

# Team Structure & Roles (Agents)

The development team follows a cross-functional structure with clear ownership and responsibilities. Each agent has specific deliverables and quality gates.

| **Role** | **Count** | **Key Responsibilities** |
| --- | --- | --- |
| **Tech Lead** | 1 | Architecture design, code reviews, technical decisions, performance optimization |
| **Backend Developer** | 2 | API development, database design, authentication, auto-scheduling engine, exports |
| **Frontend Developer** | 2 | React components, svar.dev integration, UI/UX, state management, responsive design |
| **DevOps Engineer** | 1 | Docker setup, CI/CD pipelines, monitoring, backups, security hardening |
| **QA Engineer** | 1 | Test planning, automated tests, integration testing, bug tracking |
| **Product Owner** | 1 | Requirements, user stories, prioritization, stakeholder communication |

# Development Rules & Standards

## Coding Standards

### General Rules

1. **TypeScript Strict Mode:** All code must use TypeScript with strict mode enabled
2. **ESLint Compliance:** Zero ESLint errors or warnings allowed in production code
3. **Code Comments:** JSDoc comments required for all public functions and classes
4. **No console.log:** Use proper logging library (Winston/Pino) instead
5. **Error Handling:** All async operations must have try-catch blocks

### Backend Rules

1. **Input Validation:** All API inputs validated with Zod or Joi schemas
2. **Database Transactions:** Use transactions for operations affecting multiple tables
3. **SQL Injection Prevention:** Only use Prisma ORM, never raw SQL
4. **Authentication:** All protected routes must verify JWT and check permissions
5. **Rate Limiting:** All public endpoints must have rate limiting

### Frontend Rules

* **Component Size:** Components should not exceed 250 lines
* **Props Interface:** All component props must be typed with interfaces
* **State Management:** Use Zustand for global state, useState for local state
* **API Calls:** All API calls must use the centralized API client with error handling
* **Accessibility:** All interactive elements must be keyboard accessible

## Git Workflow

* **Branch Naming:** feature/description, bugfix/description, hotfix/description
* **Commit Messages:** Use conventional commits (feat, fix, docs, refactor, test)
* **Pull Requests:** Require 2 approvals before merge, CI/CD must pass
* **Main Branch:** Protected, deployable at all times

## Testing Requirements

* **Unit Tests:** Minimum 80% code coverage for business logic
* **Integration Tests:** All API endpoints must have integration tests
* **E2E Tests:** Critical user flows must have Playwright tests
* **Test Data:** Use factories for test data, never hardcode

## Security Rules

* **Secrets Management:** Never commit secrets, use environment variables
* **HTTPS Only:** All traffic must be encrypted
* **XSS Prevention:** Sanitize all user inputs
* **CORS:** Whitelist specific origins only
* **Dependencies:** Weekly security audits with npm audit

# Detailed Feature Specifications

## 1. Authentication System

### User Registration

* Email and password with strength requirements (min 8 chars, uppercase, lowercase, number, symbol)
* Email verification via token sent to user's email
* Rate limiting: 5 registration attempts per hour per IP
* CAPTCHA integration for bot prevention

### User Login

* JWT access token (15 min expiry) and refresh token (7 days)
* Remember me option extends refresh token to 30 days
* Failed login lockout: 5 attempts triggers 15-minute lockout
* Session management with Redis for active session tracking

### Password Management

* Password reset via email with 1-hour expiry token
* Password change requires current password verification
* bcrypt hashing with 12 rounds

## 2. Team & Group Management

### Team Creation

* Team name, description, avatar
* Creator automatically becomes Team Admin
* Support for sub-teams (departments, work groups)

### Team Roles

* **Admin:** Full control, can delete team
* **Manager:** Can create projects, assign tasks, invite members
* **Member:** Can view projects, update assigned tasks
* **Viewer:** Read-only access

### Team Assignment to Projects

* Multiple teams can be assigned to a single project
* Team members automatically get access to assigned projects
* Task assignment can be to individuals or entire teams

## 3. Gantt Chart Features

### Task Visualization

* Hierarchical task structure with unlimited nesting
* Collapsible parent tasks to manage complexity
* Progress bars showing completion percentage
* Milestone markers for key deliverables

### Color Coding System

* **Status-based:** Not Started (gray), In Progress (blue), Completed (green), Blocked (red)
* **Priority-based:** Low (green), Medium (yellow), High (orange), Critical (red)
* **Team-based:** Each team gets unique color
* **Custom rules:** Users can define custom color rules

### Dependencies

* **Finish-to-Start:** Task B cannot start until Task A finishes
* **Start-to-Start:** Task B cannot start until Task A starts
* **Finish-to-Finish:** Task B cannot finish until Task A finishes
* Visual dependency arrows in Gantt chart
* Circular dependency detection and prevention

## 4. Auto-Scheduling Engine

### Core Logic

When a task's start or end date changes, the auto-scheduler automatically recalculates all dependent tasks' timelines to maintain logical consistency.

### Features

* **Cascade Updates:** Changes ripple through dependency chains
* **Working Days:** Calculations exclude weekends and holidays
* **Buffer Time:** Optional padding between dependent tasks
* **Resource Conflicts:** Detect when same person assigned to overlapping tasks
* **Critical Path:** Identify tasks that directly impact project end date
* **Manual Override:** Users can lock tasks to prevent auto-adjustments

### User Experience

* Preview changes before applying
* Highlight affected tasks in the Gantt chart
* One-click undo for auto-scheduling changes
* Toggle auto-scheduling on/off per project

## 5. Export System

### PDF Export

* Full Gantt chart visualization
* Project summary page with metadata
* Task list with details
* Team member assignments
* Customizable templates (Executive, Detailed, Status Report)

### Excel Export

* **Overview Sheet:** Project summary, stats, team members
* **Tasks Sheet:** All tasks with dates, assignees, status, priority
* **Resources Sheet:** Team member utilization and workload
* **Timeline Sheet:** Visual timeline representation
* Formatted with colors, borders, and formulas

### Other Formats

* **CSV:** Raw task data for external analysis
* **PNG:** High-resolution Gantt chart image
* **JSON:** Complete project data for API integrations

### Delivery Options

* Instant download
* Email delivery to multiple recipients
* Scheduled exports (daily, weekly, monthly)
* Cloud storage integration (future: Google Drive, Dropbox)

# Development Phases & Timeline

The project is divided into 5 major phases, each with clear deliverables and milestones. Total estimated duration: 16-20 weeks.

## Phase 1: Foundation & Infrastructure (Weeks 1-3)

### Deliverables

* Project repository with monorepo structure
* Docker Compose setup with all services
* Database schema design and Prisma models
* Basic Express API with health check endpoint
* React app with routing and basic layout
* CI/CD pipeline with GitHub Actions
* Development and testing environments on VPS

### Key Tasks

* Set up monorepo with pnpm workspaces
* Create Dockerfiles for frontend, backend, and database
* Design database ERD for all entities
* Set up ESLint, Prettier, and TypeScript configs
* Configure Nginx reverse proxy

## Phase 2: Authentication & User Management (Weeks 4-6)

### Deliverables

* Complete authentication system
* User registration and login pages
* Password reset flow
* User profile management
* Role-based access control middleware

## Phase 3: Teams & Projects (Weeks 7-10)

### Deliverables

* Team creation and management UI
* Member invitation system
* Project creation and configuration
* Team assignment to projects
* Basic task CRUD operations

## Phase 4: Gantt Chart & Auto-Scheduling (Weeks 11-15)

### Deliverables

* svar.dev Gantt integration
* Task visualization with color coding
* Subtask support and hierarchies
* Dependency management
* Auto-scheduling engine with preview
* Drag-and-drop task scheduling
* Critical path calculation

## Phase 5: Export System & Polish (Weeks 16-20)

### Deliverables

* PDF export with templates
* Excel export with multiple sheets
* CSV, PNG, JSON exports
* Email delivery system
* Notification system
* Performance optimization
* Security hardening
* Comprehensive testing
* Production deployment

# Docker Deployment Configuration

## Services Architecture

The application runs as a multi-container Docker application orchestrated by Docker Compose. Each service is independently scalable and maintainable.

| **Service** | **Description** | **Port** |
| --- | --- | --- |
| **nginx** | Reverse proxy, SSL termination, static file serving | 80, 443 |
| **frontend** | React application, production build served by nginx | 3000 |
| **backend** | Node.js Express API server | 4000 |
| **postgres** | PostgreSQL database with automated backups | 5432 |
| **redis** | Redis for caching and session management | 6379 |

## VPS Requirements

* **OS:** Ubuntu 22.04 LTS or later
* **CPU:** Minimum 2 cores, recommended 4 cores
* **RAM:** Minimum 4GB, recommended 8GB
* **Storage:** Minimum 50GB SSD
* **Network:** Static IP address, domain name configured

## Deployment Steps

1. Install Docker and Docker Compose on Ubuntu VPS
2. Clone repository and configure environment variables
3. Set up SSL certificates with Let's Encrypt
4. Build Docker images: docker-compose build
5. Run database migrations: npm run prisma:migrate
6. Start services: docker-compose up -d
7. Configure automated backups with cron jobs
8. Set up monitoring and logging

## Backup Strategy

* Daily automated PostgreSQL dumps
* Retention: 7 daily, 4 weekly, 3 monthly
* Off-site backup storage (AWS S3 or equivalent)
* Quarterly disaster recovery drills

# Testing & Quality Assurance

## Testing Pyramid

* **Unit Tests (70%):** Jest for business logic, utilities, and pure functions
* **Integration Tests (20%):** Supertest for API endpoints
* **E2E Tests (10%):** Playwright for critical user flows

## Test Coverage Goals

* **Backend:** Minimum 80% code coverage
* **Frontend:** Minimum 70% code coverage
* **Critical Features:** 100% coverage (auth, payments, data export)

## Performance Testing

* Load testing with k6 or Artillery
* Target: 100 concurrent users
* API response time: <200ms for 95th percentile
* Page load time: <2s for initial load, <500ms for subsequent

## Security Testing

* OWASP Top 10 vulnerability scanning
* Dependency audit with npm audit
* Penetration testing before production launch

# Risk Management

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| **svar.dev compatibility issues** | Delays in Gantt integration, UI bugs | Early POC, vendor support contract, fallback library evaluation |
| **Auto-scheduling complexity** | Feature delays, performance issues | Phased implementation, extensive testing, performance profiling |
| **Team member turnover** | Knowledge loss, timeline impact | Comprehensive documentation, pair programming, knowledge sharing sessions |
| **Security vulnerabilities** | Data breaches, reputation damage | Security audits, penetration testing, regular dependency updates |
| **VPS downtime** | Service unavailability | Monitoring, automated backups, disaster recovery plan, multi-region future |

# Success Metrics & KPIs

## Technical Metrics

* **Uptime:** 99.5% or higher
* **API Response Time:** <200ms for 95th percentile
* **Test Coverage:** 80%+ backend, 70%+ frontend
* **Zero Critical Bugs:** In production for 30 days post-launch

## User Experience Metrics

* **Page Load Time:** <2 seconds initial, <500ms subsequent
* **Auto-scheduling Speed:** <1 second for projects with 500 tasks
* **Export Generation:** <5 seconds for PDF/Excel

## Development Metrics

* **Code Review Time:** <24 hours average
* **Deployment Frequency:** At least weekly
* **Bug Fix Time:** <48 hours for critical, <7 days for normal

# Conclusion

This comprehensive project plan provides a roadmap for building a production-ready project management suite with advanced Gantt chart capabilities, intelligent auto-scheduling, and enterprise-grade features. The modular architecture ensures maintainability, the clearly defined phases provide realistic timelines, and the strict development rules ensure code quality and security.

By following this plan and adhering to the established standards, the team will deliver a professional, scalable solution that can be deployed confidently on Ubuntu VPS with Docker.

## Next Steps

1. Review and approve this project plan
2. Assemble the development team
3. Set up development infrastructure
4. Begin Phase 1: Foundation & Infrastructure
5. Schedule weekly progress reviews