Team Task Manager - Phase 2 Execution PRD

Version: 2.0
Date: 2025-09-27
Repository: https://github.com/1985ML/team-task-manager
Branch: main
AI Execution Optimized

🎯 Phase 2 Objectives

Build upon the successful Phase 1 foundation with advanced project management, API capabilities, and enterprise features. Focus on extending existing team-based structure with sophisticated workflows and integrations.

Success Criteria

- Maintain 100% backward compatibility with Phase 1
- Add comprehensive API with documentation
- Implement advanced project management features
- Enhance reporting and analytics capabilities
- Add external integrations (GitHub, file storage)

🛠️ Technical Foundation (Already Implemented)

Existing Architecture ✅

```
// Current Database Schema (Prisma)
User, Team, TeamMember, Task, Comment, Notification
Account, Session, VerificationToken (NextAuth)
Enums: UserRole, TeamMemberRole, TaskStatus, Priority, NotificationType
```

Current Tech Stack ✅

- Next.js 14 + TypeScript + Tailwind CSS + Radix UI
- PostgreSQL + Prisma ORM + NextAuth.js
- Repository: https://github.com/1985ML/team-task-manager

🚀 Phase 2 Feature Implementation Plan

2.1 Projects Layer (HIGH PRIORITY)

Goal: Add project organization on top of existing team structure

Database Changes Required:

```
-- Add Projects table
projects (
  id STRING PRIMARY KEY,
  name STRING NOT NULL,
  description TEXT,
  status ProjectStatus DEFAULT 'ACTIVE',
  startDate DateTime,
  dueDate DateTime,
  teamId STRING REFERENCES teams(id),
  createdById STRING REFERENCES users(id),
  createdAt DateTime DEFAULT NOW(),
  updatedAt DateTime DEFAULT NOW()
);

-- Add project relationship to tasks
ALTER TABLE tasks ADD COLUMN projectId STRING REFERENCES projects(id);

-- Add ProjectStatus enum
enum ProjectStatus { PLANNING, ACTIVE, ON_HOLD, COMPLETED, CANCELLED }
```

Implementation Steps:

1. Update Prisma schema with Project model
2. Create project management pages: `/dashboard/projects`, `/dashboard/projects/new`, `/dashboard/projects/[id]`
3. Add project selection to task creation/editing
4. Update task list/kanban views to filter by project
5. Add project overview dashboard
6. Update navigation sidebar with projects section

Components to Create:

- `components/projects/project-form.tsx`
- `components/projects/project-list.tsx`
- `components/projects/project-card.tsx`
- `components/projects/project-overview.tsx`

2.2 REST API Development (HIGH PRIORITY)

Goal: Comprehensive REST API for external integrations

API Endpoints to Implement:

```
// Authentication
POST /api/auth/api-key - Generate API key for user

// Projects
GET /api/v1/projects - List projects with filtering
POST /api/v1/projects - Create project
GET /api/v1/projects/[id] - Get project details
PATCH /api/v1/projects/[id] - Update project
DELETE /api/v1/projects/[id] - Delete project

// Tasks (Enhanced)
GET /api/v1/tasks - List with advanced filtering
POST /api/v1/tasks - Create task
GET /api/v1/tasks/[id] - Get task details
PATCH /api/v1/tasks/[id] - Update task
DELETE /api/v1/tasks/[id] - Delete task
POST /api/v1/tasks/[id]/comments - Add comment

// Teams
GET /api/v1/teams - List user's teams
GET /api/v1/teams/[id] - Team details
GET /api/v1/teams/[id]/members - Team members

// Analytics
GET /api/v1/analytics/tasks - Task completion analytics
GET /api/v1/analytics/teams - Team performance metrics
```

Implementation Steps:

1. Create API key generation system in database
2. Add API authentication middleware
3. Implement all endpoints with proper error handling
4. Add rate limiting (300 requests/minute)
5. Generate OpenAPI documentation
6. Create API settings page in dashboard

Database Changes:

```
-- API Keys table
api_keys (
  id STRING PRIMARY KEY,
  userId STRING REFERENCES users(id),
  name STRING NOT NULL,
  keyHash STRING UNIQUE NOT NULL,
  scopes STRING[],
  active BOOLEAN DEFAULT true,
  lastUsed DateTime,
  createdAt DateTime DEFAULT NOW(),
  expiresAt DateTime
);
```

2.3 File Storage & Attachments (MEDIUM PRIORITY)

Goal: Task and project file attachments with cloud storage

Database Changes:

```
-- Attachments table
attachments (
  id STRING PRIMARY KEY,
  entityType AttachmentEntityType,
  entityId STRING NOT NULL,
  filename STRING NOT NULL,
  originalName STRING NOT NULL,
  fileSize INT,
  mimeType STRING,
  cloudStoragePath STRING NOT NULL,
  uploadedById STRING REFERENCES users(id),
  uploadedAt DateTime DEFAULT NOW()
);

enum AttachmentEntityType { TASK, PROJECT, COMMENT }
```

Implementation Steps:
1. Initialize cloud storage using `initialize_cloud_storage` tool
2. Create file upload API routes with validation
3. Add file upload components to task and project forms
4. Implement file preview and download functionality
5. Add file management to existing task/project detail views

Components to Create:
- `components/attachments/file-upload.tsx`
- `components/attachments/file-list.tsx`
- `components/attachments/file-preview.tsx`

2.4 Advanced Reporting Dashboard (MEDIUM PRIORITY)

Goal: Enhanced analytics and reporting capabilities

Features to Implement:
- Team performance metrics (task completion rates, overdue trends)
- Project timeline tracking and health indicators
- Individual productivity analytics
- Customizable date range filtering
- Export capabilities (PDF, CSV)

Implementation Steps:
1. Create analytics API endpoints
2. Build interactive charts using Chart.js/Recharts
3. Add filtering and date range selection
4. Implement export functionality
5. Create dedicated analytics page at `/dashboard/analytics`

New Page:
- `app/dashboard/analytics/advanced/page.tsx`

2.5 Recurring Tasks System (MEDIUM PRIORITY)

Goal: Automated recurring task generation

Database Changes:

```
-- Recurring task series
recurring_task_series (
  id STRING PRIMARY KEY,
  taskId STRING REFERENCES tasks(id),
  frequency RecurrenceFrequency,
  interval INT DEFAULT 1,
  daysOfWeek INT[], -- for weekly patterns
  dayOfMonth INT, -- for monthly patterns
  endDate DateTime,
  nextDueDate DateTime,
  active BOOLEAN DEFAULT true,
  createdAt DateTime DEFAULT NOW()
);

enum RecurrenceFrequency { DAILY, WEEKLY, MONTHLY }
```

Implementation Steps:
1. Add recurrence settings to task creation form
2. Create background job system for task generation
3. Implement recurrence logic with proper date calculations
4. Add recurrence management interface
5. Update task list to show recurring indicators

2.6 GitHub Integration (LOW PRIORITY - If Requested)

Goal: Connect tasks to GitHub issues and pull requests

Features:
- Link tasks to GitHub issues
- Sync task status with issue status
- Create GitHub issues from tasks
- Display GitHub activity in task comments

Implementation Steps:
1. Use existing GitHub API token configuration
2. Create GitHub service wrapper
3. Add GitHub linking to task forms
4. Implement bidirectional sync
5. Add GitHub activity display

2.7 Advanced Search & Filtering (LOW PRIORITY)

Goal: Comprehensive search across all entities

Features:
- Full-text search across tasks, projects, comments
- Advanced filter combinations
- Saved search presets
- Search within specific teams/projects

Implementation Steps:
1. Add search indexes to database
2. Create search API endpoint
3. Build advanced search UI

4. Implement search result highlighting
5. Add saved search functionality

📝 Implementation Priority

Sprint 1 (Weeks 1-3): Foundation

1. ✅ Projects layer implementation
2. ✅ Basic API endpoints (CRUD operations)
3. ✅ API documentation setup

Sprint 2 (Weeks 4-6): Core Features

1. ✅ File storage and attachments
2. ✅ Advanced reporting dashboard
3. ✅ API authentication and rate limiting

Sprint 3 (Weeks 7-9): Advanced Features

1. ✅ Recurring tasks system
2. ✅ Enhanced search and filtering
3. ✅ GitHub integration (if requested)

Sprint 4 (Weeks 10-12): Polish & Testing

1. ✅ Performance optimization
2. ✅ Comprehensive testing
3. ✅ Documentation completion
4. ✅ Deployment and monitoring

🔧 Technical Implementation Guidelines

Code Quality Standards

- Maintain TypeScript strict mode
- Follow existing component patterns
- Use server-side rendering where appropriate
- Implement proper error boundaries
- Add loading states for all async operations

Database Guidelines

- Use Prisma migrations for schema changes
- Maintain referential integrity
- Add proper indexes for performance
- Implement cascade deletes where appropriate

API Standards

- RESTful design principles
- Consistent error response format

- Proper HTTP status codes
- Comprehensive input validation
- Rate limiting and authentication

Security Requirements

- API key authentication for external access
- Input sanitization and validation
- Secure file upload handling
- Proper authorization checks
- CORS configuration for API endpoints

📊 Success Metrics

Functional Metrics

- [ ] All API endpoints respond within 200ms p95
- [ ] File uploads handle files up to 25MB
- [ ] Recurring tasks generate accurately
- [ ] Advanced search returns results in < 500ms
- [ ] Reports generate and export successfully

User Experience Metrics

- [ ] No breaking changes to existing functionality
- [ ] New features integrate seamlessly with existing UI
- [ ] Mobile responsiveness maintained
- [ ] Performance remains optimal (< 1s page loads)

Technical Metrics

- [ ] API documentation coverage > 95%
- [ ] Code test coverage > 70%
- [ ] Database query performance maintained
- [ ] Zero data integrity issues

🚀 Deployment Strategy

Development Environment

```
# Repository already configured
git clone https://github.com/1985ML/team-task-manager.git
cd team_task_manager/app
yarn install
yarn dev
```

Database Migrations

```
# Apply new schema changes
yarn prisma db push
yarn prisma generate
```

API Testing

```
# Test API endpoints
yarn test:api
```

Production Deployment

- Use existing deployment pipeline
- Run database migrations
- Update environment variables for new features
- Monitor performance metrics post-deployment

🔗 Integration Points

External Services Required

1. **Cloud Storage**: For file attachments (use `initialize_cloud_storage`)
2. **GitHub API**: For GitHub integration (already configured)
3. **Email Service**: For enhanced notifications (if implemented)

Existing Integrations Maintained

- NextAuth.js authentication
- PostgreSQL database
- GitHub version control
- Tailwind CSS + Radix UI components

This Phase 2 PRD is optimized for AI execution with clear, actionable implementation steps, specific database schemas, and comprehensive technical guidelines. Each feature is designed to extend the existing Phase 1 foundation without breaking changes.

Ready for Implementation: This PRD provides all necessary details for autonomous development execution.