Smart Sprint - Project Report

Executive Summary

Smart Sprint is a comprehensive team management application built with the MERN stack (MongoDB, Express, React, Node.js). The platform facilitates efficient project and task management with a focus on role-based access control, team organization, and visual task tracking through a Kanban board interface. This report provides an overview of the system architecture, key features implemented, technical decisions, and future development opportunities.

System Architecture

Technology Stack

Smart Sprint employs a modern web development stack:

- **Frontend**: React.js with Material UI and React Bootstrap
- **Backend**: Node.js with Express.js
- **Database**: MongoDB (document-based NoSQL)
- **Authentication**: JSON Web Tokens (JWT)
- **State Management**: React Hooks (useState, useEffect)
- **Drag & Drop Functionality**: @hello-pangea/dnd library
- **API Communication**: Axios

System Components

The application architecture follows a standard client-server model:

1. **Frontend Application**:
    - Single Page Application (SPA) built with React
    - Responsive UI using Bootstrap and Material UI components
    - Component-based architecture for reusability
    - Client-side routing with React Router
2. **Backend API Server**:
    - RESTful API endpoints built with Express.js
    - JWT-based authentication middleware
    - Role-based access control
    - File upload capabilities with Multer
3. **Database**:
    - MongoDB document collections for Users, Projects, and Tasks
    - Mongoose ODM for schema validation and model interactions
    - Indexes for optimized queries
4. **Development & Deployment**:
    - Concurrent development servers with concurrently package
    - Environment variable management for configuration
    - Modular component structure for maintainability

Key Features Implemented

1. User Authentication and Management

- **Secure Login System**:
  - JWT-based authentication
  - Password hashing with bcrypt
  - First login password reset functionality
  - Persistent sessions with localStorage
- **User Management**:
  - Role-based user creation (Admin, Project Manager, Developer)
  - Team and level assignments
  - Profile picture upload and management
  - Role-specific UI adaptations
- **Drag & Drop User Organization**:
  - Visual team management interface
  - Move users between teams via drag and drop
  - Admin-only deletion through drag-and-drop operations
  - Automatic team/level assignments based on role

2. Project Management

- **Project Creation and Configuration**:
  - Project metadata management
  - Team member assignment
  - Project visualization with status indicators
  - Permission control based on user roles
- **Member Request System**:
  - User requests to join projects
  - Drag & drop approval/rejection mechanism
  - Notification system for request status

3. Task Management with Kanban Board

- **Visual Task Management**:
  - Kanban board with status columns
  - Drag & drop task movement between statuses
  - Automatic status transitions based on user roles
  - Task sorting and filtering capabilities
- **Task Creation and Assignment**:
  - Intuitive task creation interface
  - Auto-assignment of team based on assignee
  - Priority and due date management
  - Role-specific permission controls
- **Special Role Behaviors**:
  - Developer role restrictions (cannot directly move to Completed)

    - Project Manager approval capabilities
    - Admin override options

4. Responsive UI Design

- **Cross-Device Compatibility**:
    - Optimized layouts for desktop and mobile devices
    - Adaptive component arrangement
    - Touch-friendly interactions for mobile
- **Visual Consistency**:
    - Unified color scheme and component styling
    - Status and role indication through color-coded badges
    - Intuitive drag and drop interfaces

Technical Implementations

Database Schema Design

1. **User Model**:

```
{
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  isFirstLogin: { type: Boolean, default: true },
  role: { type: String, enum: ['Admin', 'Project Manager', 'Developer'] },
  team: { type: String, enum: ['Design', 'Database', 'Backend', 'Frontend', 'DevOps',
'Tester/Security', 'None', 'admin', 'pm'] },
  level: { type: String, enum: ['Lead', 'Senior', 'Dev', 'Junior', 'admin', 'pm'] },
  profilePicture: { type: String, default: '' },
  // Additional user fields
}
```

2. **Project Model**:

```
{
  name: { type: String, required: true },
  description: { type: String },
  status: { type: String, enum: ['Planning', 'In Progress', 'On Hold', 'Completed']
},
  members: [{
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    role: { type: String }
  }],
  requests: [{
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    status: { type: String, enum: ['Pending', 'Approved', 'Rejected'] }
  }],
  // Additional project fields
}
```

3. **Task Model**:

```
{
  title: { type: String, required: true },
  description: { type: String },
  status: { type: String, enum: ['Todo', 'In Progress', 'Review', 'Needs Work',
'Completed'] },
  priority: { type: String, enum: ['Low', 'Medium', 'High', 'Critical'] },
  project: { type: mongoose.Schema.Types.ObjectId, ref: 'Project', required: true },
  assignee: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  team: { type: String, required: true },
  dueDate: { type: Date },
  // Additional task fields
}
```

Authentication Flow

1. User submits login credentials
2. Backend validates credentials against database
3. On success, JWT token is generated with user id and role
4. Token is stored in localStorage on the client
5. Subsequent API requests include the token in Authorization header
6. Protected routes verify token validity and role permissions
7. First-time login triggers password change requirement

Drag & Drop Implementation

The application implements drag and drop functionality in multiple components:

1. **KanbanBoard.js**:
   - Uses DragDropContext to manage drag operations
   - Implements task movement between status columns
   - Handles role-specific restrictions (e.g., Developer cannot move to Completed)
   - Updates task status in the database on successful drop
2. **UserList.js**:
   - Provides drag interface for user management
   - Supports two view modes: List View and Team View
   - Allows users to be dragged between teams in Team View
   - Updates user team assignments in the database
3. **ProjectRequests.js**:
   - Implements drag to approve/reject project join requests
   - Updates request status in the database

Backend API Endpoints

The backend provides RESTful API endpoints for all operations:

1. **Authentication Routes**:
   - `POST /api/auth/login` - User login

- `POST /api/auth/change-password` - Password change

2. **User Routes**:
   - `GET /api/users` - List all users
   - `POST /api/users` - Create new user
   - `PATCH /api/users/:id` - Update user
   - `DELETE /api/users/:id` - Delete user
   - `POST /api/users/profile-picture` - Upload profile picture
3. **Project Routes**:
   - `GET /api/projects` - List all projects
   - `POST /api/projects` - Create new project
   - `PUT /api/projects/:id` - Update project
   - `DELETE /api/projects/:id` - Delete project
   - `POST /api/projects/:id/requests` - Submit join request
   - `PATCH /api/projects/:id/requests/:requestId` - Update request status
4. **Task Routes**:
   - `GET /api/tasks/project/:projectId` - Get tasks by project
   - `POST /api/tasks` - Create new task
   - `PUT /api/tasks/:id` - Update task
   - `PATCH /api/tasks/:id/status` - Update task status only
   - `DELETE /api/tasks/:id` - Delete task

Challenges and Solutions

Challenge 1: Role-Based Permission Management

Problem: Implementing granular permissions for different user roles across the system.

Solution:

- Implemented middleware that checks user role before allowing access to routes
- Created component-level permission checks in the UI
- Used role-specific redirects and component rendering
- Created a centralized permission system that adapts UI elements based on user role

Challenge 2: Drag-and-Drop Implementation

Problem: Implementing intuitive drag-and-drop interfaces with appropriate backend updates.

Solution:

- Utilized @hello-pangea/dnd library for drag-and-drop functionality
- Implemented optimistic UI updates with error handling
- Created specialized droppable zones for different purposes
- Added role-specific restrictions for drag operations

Challenge 3: Maintaining Data Integrity

Problem: Ensuring data consistency when multiple users interact with the system simultaneously.

Solution:

- Implemented proper validation on both frontend and backend
- Used mongoose schema validation for data integrity
- Added transaction-like operations for critical updates
- Implemented error handling and recovery mechanisms

Future Development Opportunities

1. **Real-time Collaboration**:
    - Implement WebSockets for real-time updates
    - Add collaborative editing features
    - Implement notification system for changes
2. **Enhanced Analytics**:
    - Add project and task analytics dashboard
    - Implement productivity metrics and reporting
    - Create visual progress tracking over time
3. **Advanced Task Management**:
    - Implement subtasks and dependencies
    - Add time tracking features
    - Create automated workflow capabilities
4. **Mobile Application**:
    - Develop native mobile applications
    - Implement offline capabilities
    - Add push notifications

Conclusion

Smart Sprint successfully implements a comprehensive team management solution with a focus on visual task management and role-based permissions. The application demonstrates effective use of modern web technologies and provides a solid foundation for future enhancements. The drag-and-drop interfaces provide an intuitive user experience for task management, team organization, and project membership control.

The application satisfies all initial requirements and adds additional features to enhance usability and user experience. With its modular architecture and clean code organization, Smart Sprint is well-positioned for continued development and feature expansion.

Appendix

Technology Version Information

- Node.js: v14.x or higher
- React: v18.x
- MongoDB: v5.x
- Express: v4.x
- Material UI: v5.x
- React Bootstrap: v2.x
- @hello-pangea/dnd: v16.x

Setup and Installation Guide

For detailed setup instructions, please refer to the README.md file in the project repository.