Software Requirements Specification

Project: ClickUp Clone

Team Members:

Muhammad Akbar

Abdulrehman

Haris Hassan

Fatima Tariq

Ahmed Bin Khalid

Date: [20/05/2025]

2. Table of Contents

1. Introduction
2. Team Overview
3. Overall Description
4. Functional Requirements
5. Non-Functional Requirements
6. System Architecture
7. Diagrams & Models
8. Project Timeline
9. Glossary
10. References

1. Introduction

1.1 Purpose

The purpose of this document is to define the Software Requirements Specification (SRS) for the development of a project management application inspired by ClickUp. This application is designed to help teams and individuals efficiently manage their tasks, collaborate in real-time, and stay organized within a structured workspace environment.

1.2 Scope

The system will allow users to register, create and join workspaces, form teams, create and manage projects, assign tasks, and collaborate through comments and status updates. It will support functionalities like task categorization into lists, multi-user task assignment, customizable statuses, and notifications for task activities. The platform aims to streamline productivity by integrating key project management features in a user-friendly interface.

1.3 Intended Audience

This document is intended for:

- **Developers** involved in building the application.
- **Testers** responsible for validating system behavior.
- **Documentation writers** who will create user manuals and guides.

1.4 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **UI**: User Interface
- **CRUD**: Create, Read, Update, Delete
- **UUID**: Universally Unique Identifier

2. Team Overview

The development team for this project consists of five dedicated members, each contributing their expertise to ensure the successful implementation of the ClickUp clone. The team is divided into frontend and backend roles to streamline development and maintain a clear separation of concerns.

Team Members and Roles:

- **Muhammad Akbar** – Backend Developer
- **Abdulrehman** – Frontend Developer
- **Haris Hassan** – Frontend Developer
- **Fatima Tariq** – Backend Developer
- **Ahmed Bin Khalid** – Backend Developer

Each team member is responsible for delivering high-quality code within their respective domains, collaborating closely to ensure integration, testing, and deployment processes are efficient and effective.

3. Overall Description

This section provides a broad overview of the ClickUp clone project, outlining the system's functionality, user roles, environment, constraints, and design considerations.

3.1 Product Perspective

The project is a web-based project management tool inspired by ClickUp. It is being developed as a standalone system but may be integrated with other tools or services in the future. The system is modular, with separate components for user management, workspace collaboration, task handling, and notifications.

3.2 Product Features

Key features of the system include:

- User authentication (sign up, login, password recovery)
- Workspace creation and team invitations
- Team creation (public and private)
- Project, list, and task management
- Multi-user task assignment and status updates
  Commenting on tasks for real-time collaboration
- Notifications for task activity and team updates

3.3 User Classes and Characteristics

The primary users of the system are:

- **Workspace Owners:** Users who create and manage workspaces.
- **Team Members:** Users who collaborate on tasks within workspaces.
- **Administrators (optional):** Users with elevated permissions to oversee system management.

3.4 Operating Environment

The system will run in modern web browsers (Chrome, Firefox, Edge) and be hosted on a cloud server using the following stack:

- **Frontend:** Vue.js
- **Backend:** Laravel (PHP Framework)
- **Database:** MongoDB, PostgreSQL
- **Real-time Features:** Socket.IO

3.5 Design and Implementation Constraints

- The application must follow MVC architecture (as per Laravel best practices).
- All data identifiers (e.g., workspace_id) will use UUIDs.
- The system must ensure secure authentication and authorization.
- The database schema should be optimized for scalability and performance.

3.6 User Documentation

User guides, FAQs, and onboarding tutorials will be provided to assist users in understanding how to use the system effectively.

3.7 Assumptions and Dependencies

- Users have access to a stable internet connection.
- The system assumes the backend server and database services are always available.
- Real-time functionality depends on proper WebSocket support on the client and server.

4. Functional Requirements

4.1 User Authentication and Profile Management

- **User Registration and Login**: Users can sign up, log in, and manage authentication using secure password hashing and email verification.
- **Password Management**: Users can reset forgotten passwords via email-based token authentication.
- **Profile Management**: Users can view, update personal details, and upload a profile picture.

4.2 Workspace Management

- **Create Workspace**: Users can create new workspaces with names, descriptions, and logos.
- **Workspace Invites**: Owners/admins can invite members via email or a unique invite link with role designation (owner, admin, member, guest).
- **Join Workspace**: Users can accept workspace invitations and become members.
- **Manage Members**: Admins can view, add, update roles, or remove workspace members.
- **Leave Workspace**: Members can leave a workspace voluntarily.
- **Delete Workspace**: Authorized users can delete a workspace along with all its associated data.

4.3 Team Management

- **Create Team**: Teams can be created within a workspace with name, description, visibility (public/private), and color coding.
- **View & Edit Team**: Team details can be updated and deleted.
- **Manage Team Members**: Admins can add or remove team members and define their roles.

4.4 Project Management

- **Create Project**: Projects are created under teams and can include details like name, description, status, start/end dates, and color code.
- **Update/Delete Project**: Projects can be updated or archived/completed/deleted.
- **View Project**: Projects and associated lists and tasks can be viewed by authorized users.

4.5 List Management

- **Create List**: Lists represent task groupings within a project and include a name, description, status, and position.
- **Update/Delete List**: Lists can be modified or removed.
- **View Lists**: Users can retrieve lists by project ID or individually.

4.6 Task Management

- **Create Task**: Tasks can be created under specific lists with title, description, priority, due date, time estimate, and initial status.
- **Update/Delete Task**: Tasks can be updated, deleted, or marked as completed.
- **Change Task Status**: Task status can be updated (e.g., todo → in_progress).
- **Task Prioritization**: Tasks can be sorted/arranged using a `position` attribute.

4.7 Task Assignment

- **Assign/Unassign Users**: Tasks can be assigned to multiple users (workspace members) with roles recorded via a mapping table.
- **View Assignees**: Users can view who is assigned to a particular task.

4.8 Commenting System

- **Post Comments**: Users can add comments to tasks to collaborate in real-time.
- **View Comments**: All comments associated with a task can be retrieved.

4.9 Access Control and Roles

- **Role-Based Permissions**: Users are assigned roles at both workspace and team levels, which determine their ability to manage members, projects, and tasks.
- **Secure Actions**: Actions like delete, update, and invite are restricted based on roles.

4.10 Notification and Invite Management

- **Send Invitations**: Invitations can be sent via email for joining a workspace.
- **Generate Invite Link**: Admins can generate reusable invite links with expiry tokens.
- **Verify Invitations**: Users can verify the validity of an invite token before joining.
- **Fetch Notifications**: Users can retrieve all notifications, optionally filtered by read status.
- **Mark as Read**: Users can mark a specific notification as read.
- **Bulk Update**: Users can mark all notifications as read with a single action.
- **Real-time Notification Support** *(planned)*: Notifications will eventually support real-time delivery using WebSocket or similar technology.

5. Non-Functional Requirements

5.1 Performance and Scalability

The system must deliver fast and consistent response times, even during high-traffic usage. API requests should respond within an average of **300ms** for common operations.
The architecture should support both **vertical and horizontal scaling**, ensuring smooth performance as the number of users, tasks, and projects grows. Modules like notifications and comments are independently deployable and scalable.

5.2 Security

All sensitive user data must be protected using modern security best practices. This includes:

- Secure password hashing (e.g., bcrypt)
- Role-based access control across all endpoints
- HTTPS communication enforcement
- Token expiration mechanisms for password reset and invitations
- Protection against common vulnerabilities (e.g., XSS, CSRF, SQL Injection)

5.3 Reliability and Availability

The platform should ensure **99.9% uptime**, with robust error handling and clear user notifications in case of failures.
Database backups, server monitoring, and failover strategies must be in place to minimize downtime and data loss.

5.4 Usability and Accessibility

The user interface should be intuitive, responsive, and consistent across devices and browsers. Key goals include:

- Easy navigation and minimal learning curve for new users
- Mobile and desktop compatibility
- Visual feedback for actions and loading states
- Accessibility considerations for diverse users (e.g., keyboard navigation, color contrast)

5.5 Maintainability

The system must be easy to update, debug, and extend. This will be achieved through:

- Modular code structure and proper documentation
- Clear API design following RESTful principles
- Use of version control (Git) and CI/CD pipelines (e.g., GitLab)
- Environment-based configuration separation.

5.6 Monitoring and Logging

The system should implement detailed logging for errors and critical events. Logs must help developers identify and resolve issues quickly.
Real-time monitoring tools should track server health, database performance, and API usage to ensure proactive system maintenance.

6. System Architecture

6.1 Overview of Architecture

The application follows a **modular monolith with service separation** architecture. Core business logic and persistent data handling are managed through a centralized **Laravel (PHP) application** following the **Model-View-Controller (MVC)** pattern. A decoupled **Node.js (Express + Socket.IO)** service is integrated for handling real-time features such as live notifications and comments using WebSockets and MongoDB.

System Interaction:

- **Frontend (Vue.js)** communicates with:
    - **Laravel** through **REST APIs** for standard CRUD operations (authentication, workspace, tasks, projects, etc.)
    - **Node.js** via **WebSocket connections** for real-time updates (e.g., new comment events, task assignments)

- **Laravel** and **Node.js** interact via **internal REST APIs or event triggers** to ensure data consistency and broadcast updates.

6.2 Technology Stack

Below is a breakdown of the technologies used across different layers of the application:

- **Frontend**
    - Vue 3 (Composition API)
      Tailwind CSS (utility-first styling)
    - Pinia (state management)
- **Backend**
    - Laravel (PHP) – Primary backend for API, logic, and persistence
    - Node.js with Express – Secondary service for real-time handling
    - Socket.IO – WebSocket library for live communication
- **Databases**
    - PostgreSQL – Relational database managed by Laravel
    - MongoDB – NoSQL database for event-driven data like notifications
- **Infrastructure & DevOps**
    - Docker – Containerized deployment for all services
    - GitLab CI/CD – Automated build, test, and deployment pipelines

6.3 Module Breakdown

The system is organized into modular components, each responsible for a distinct domain of functionality. The table below outlines the key modules and their responsibilities:

Module	Key Responsibilities
Authentication	User registration, login, password reset, JWT issuance, email verification, OAuth login.
User Profile Management	Update profile details, upload profile picture, view user metadata.
Workspace Management	Create and manage workspaces, invite members, assign roles (owner/admin/member/guest).
Team Management	Create public/private teams, manage team members, assign roles within teams.
Project Management	CRUD for projects, manage status (active/completed/archived), color-coding, team linkage.
List Management	Create project-specific task lists, manage list positions, archive/restore lists.
Task Management	Create, assign, update, and delete tasks. Set due dates, priorities, statuses, and positions.
Task Assignment	Assign/unassign workspace members to tasks, track assignment history.
Comment System	Add and retrieve comments on tasks, store comments in MongoDB.
Notification System	Real-time WebSocket notifications for events (e.g., task assignments, comments), marked as read.
Role & Permission Handling	Enforce access control at workspace/team/task level based on user role.

6.4 Data Flow & Communication

All system components communicate in a consistent, structured manner:

- **Vue.js Frontend** communicates with **Laravel** for all API-based CRUD operations using RESTful endpoints.
- For **real-time communication**, Vue.js maintains a persistent **WebSocket connection to the Node.js server**.
- **Laravel emits internal API calls or events to Node.js**, which in turn updates the relevant MongoDB collections and broadcasts notifications to connected clients.

6.5 Deployment Strategy

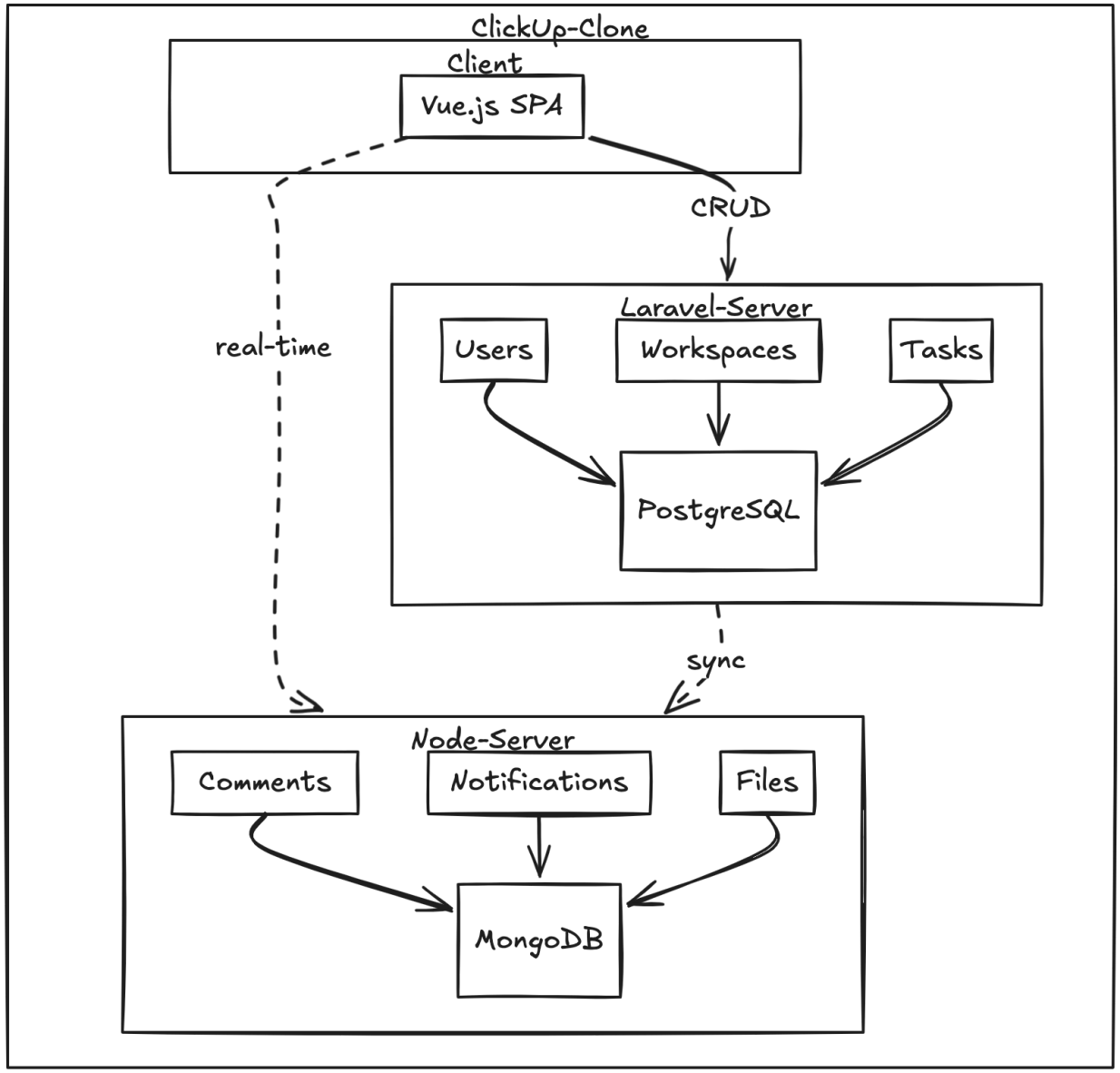
The deployment environment is based on **Dockerized containers**, ensuring consistency across development, staging, and production. Each component (Laravel backend, Node.js server, PostgreSQL, and MongoDB) runs in its own container.

Key points:

- **Containers** are orchestrated and deployed using **GitLab CI/CD pipelines**.
- **Environment variables** and Docker volumes are used to separate secrets and manage data persistently.

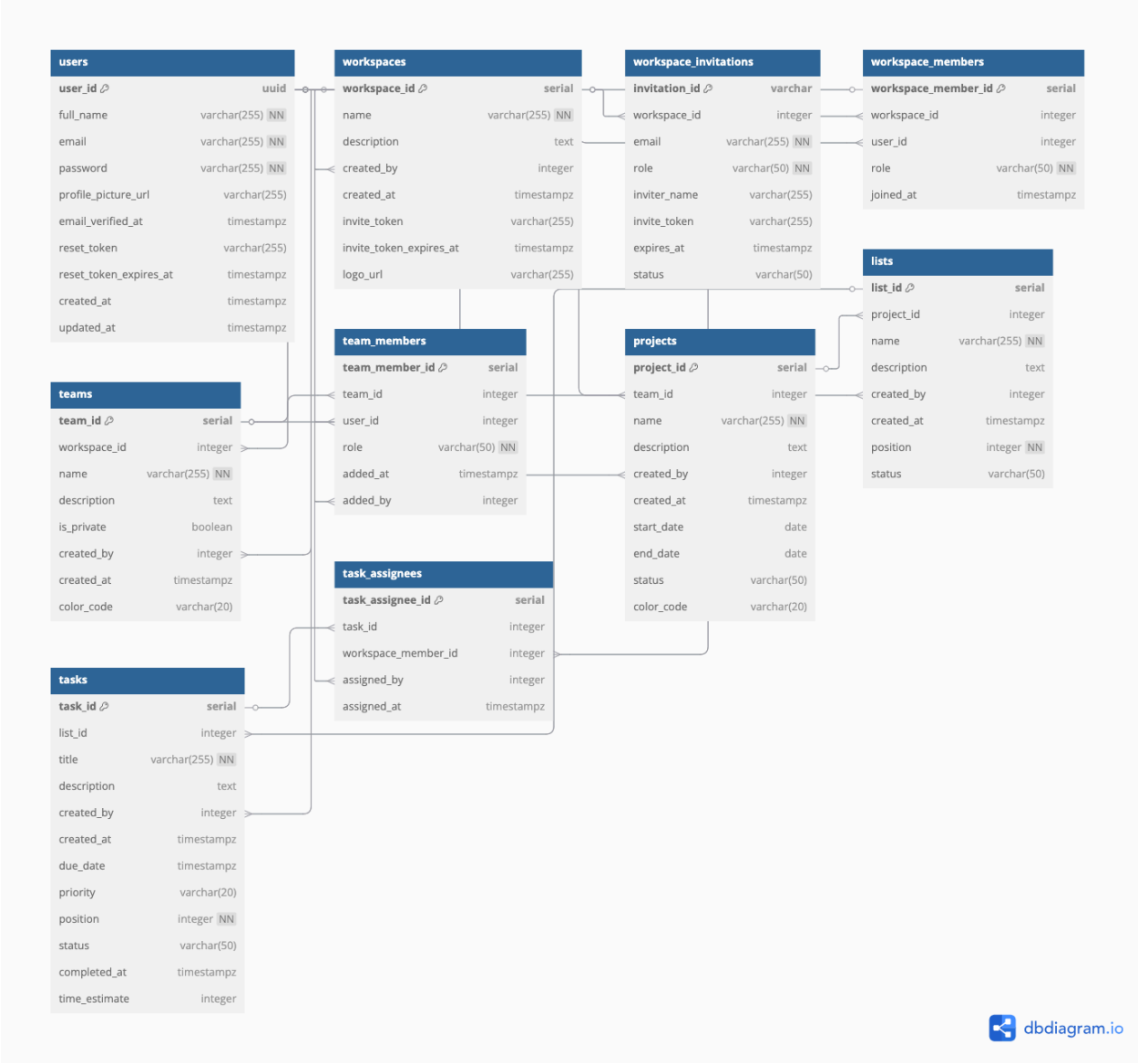
7. Diagrams & Models

7.1 System Architecture Diagram



7.2 Database Schema

PostgreSQL Database Schema



MongoDB Schema