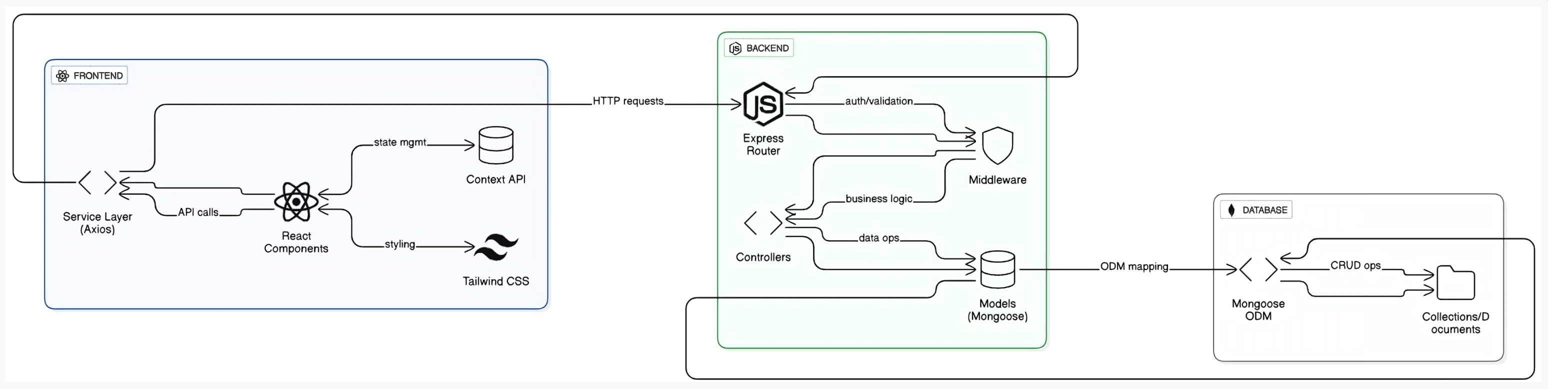
A-1 Launchpad: Laboratory Inventory Management System

Architecture & Design Document **By Team: Tech Geek**

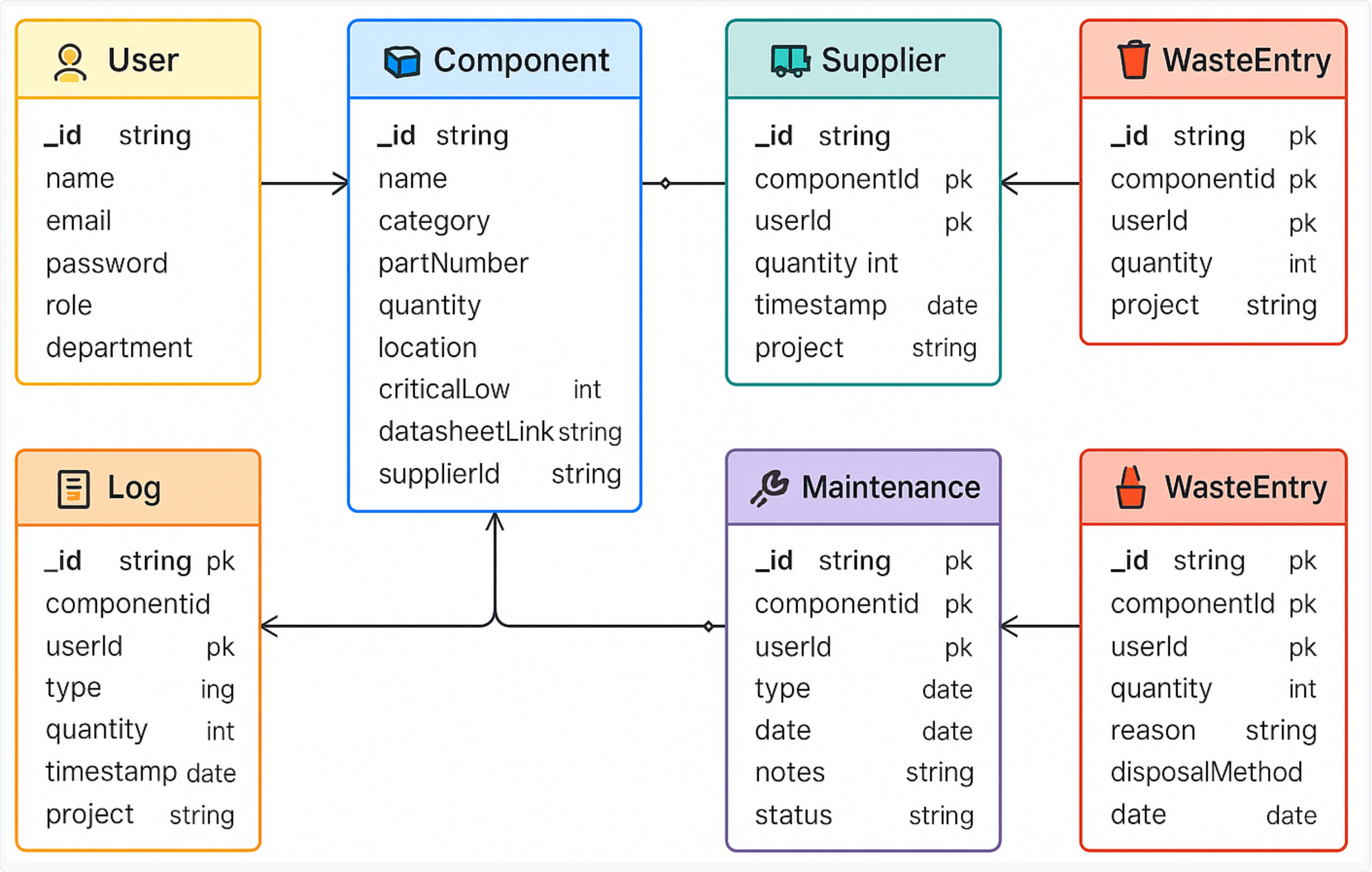
GitHub: github.com/Harshit-Patle/A-1-Launchpad

1. High-Level Architecture Diagram



Architecture: Three-tier with React.js frontend (using Vite and Tailwind CSS), Node.js/Express backend with 12 specialized controllers, and MongoDB database with 7 data models. Features AuthContext and ComponentsContext for state management, comprehensive middleware for authentication, role validation, and file uploads, with specialized components for barcode scanning, QR code generation, and advanced reporting.

2. Database Schema Diagram



Key Relationships: Users (1:many) → Components, Logs, Reservations, Maintenance, WasteEntry. Components (1:many) → Logs, Reservations, Maintenance, WasteEntry. All activities tracked via Log collection for auditing.

3. API Endpoints

Endpoint	Method	Purpose
Authentication		
/api/auth/login	POST	User authentication & JWT provision
/api/auth/me	GET	Current user information
/api/auth/register	POST	New user registration (admin only)
Component Management		
/api/components	GET	List components (filtering/sorting/paging)
/api/components	POST	Add new component
/api/components/:id	GET/PUT/ DELETE	Get/Update/Delete specific component
/api/components/categories	GET	Get all component categories
/api/components/low-stock	GET	Components below critical threshold
User Management		
/api/users	GET	List all users (admin only)
/api/users/:id	GET/PUT	Get/Update user details
/api/users/:id/role	PUT	Update user role (admin only)
Operations		
/api/logs	GET	Activity logs with filtering
/api/reservations	GET/POST	List/Create component reservations
/api/reservations/:id	PUT	Update reservation status
/api/maintenance	GET/POST	List/Add maintenance records
/api/reports/inventory-overview	GET	Inventory status summary
/api/reports/:type/export	GET	Export reports (PDF/Excel)
/api/waste	GET/POST	List/Record component disposal

4. Technology Justification

Frontend Stack

- **React.js:** Component-based architecture promotes code reusability and maintainable structure
- **Vite:** Superior development experience with hot module replacement and fast builds
- **Tailwind CSS:** Rapid UI development with utility classes and responsive design
- **Context API:** Implemented AuthContext and ComponentsContext for state management

Backend Stack

- **Node.js/Express:** Structured with 12 dedicated controllers including component, authentication, dashboard, approval workflow, and more
- **JWT Authentication:** Implemented using authMiddleware with role-based access control via roleMiddleware
- **Middleware Pattern:** Includes authentication, role validation, file upload handling, and input validation middleware

Database Stack

- **MongoDB:** Flexible schema design for User, Component, Log, Maintenance, Notification, Reservation, and WasteEntry models without requiring migrations
- **Mongoose ODM:** Schema validation, type casting, and business logic hooks for proper data management with MongoDB

Architecture Benefits

- **Full-Stack JavaScript:** Unified development experience with React frontend and Node.js backend
- **Modern Tech Stack:** Vite build system with Tailwind CSS for responsive design across desktop, tablet, and mobile views
- **Feature-Rich Design:** Includes barcode scanning, QR code generation, advanced reporting, notifications, and approval workflows
- **Specialized Components:** Custom components for inventory management, waste tracking, maintenance scheduling, and reservations

5. Scalability & Maintainability

Scalability Considerations

1. **Component Architecture:** Well-structured component hierarchy with specialized components like BarcodeScanner, NotificationCenter, and QRCodeGenerator
2. **Database Modeling:** Comprehensive data models including Component, User, ApprovalWorkflow, Log, Maintenance, Notification, Reservation, and WasteEntry
3. **Controller Organization:** Specialized controllers for each major system function (12 total) including approval, component, dashboard, and report exports
4. **Route Management:** Dedicated route files for each functional area with proper middleware application
5. **Scheduled Processing:** Background tasks managed through scheduledTasks utility for regular system maintenance
6. **Protected Routes:** Authentication-secured routes with ProtectedRoute component and role-based access control

Performance Optimization

- **Database Queries:** Efficient MongoDB queries with dedicated controller methods for specialized data access
- **API Response:** Structured API endpoints with proper error handling and consistent response formats
- **Frontend:** Responsive tables using custom useResponsiveTables hook with adaptive UI components for all device sizes
- **Monitoring:** Comprehensive logging system using winston logger with detailed activity tracking

Maintainability Approaches

1. **Modular Architecture:** Organized project with separate client and server directories, each with clear component/controller separation
2. **Context-based State:** Centralized state management using AuthContext and ComponentsContext for consistent data flow
3. **Input Validation:** Dedicated validation middleware for ensuring data integrity before processing
4. **Responsive Design:** Custom useResponsiveTables hook enabling consistent user experience across device sizes
5. **Advanced Features:** Specialized components for advanced search, barcode scanning, and QR code generation
6. **Data Management:** Comprehensive controllers for approvals, components, logs, maintenance, notifications, and more
7. **Security:** JWT-based authentication with role-based middleware for controlled access

Development Practices

- **Version Control:** Git repository hosted on GitHub with proper project structure and documentation
- **Frontend Tooling:** Vite build system with ESLint configuration and PostCSS/Tailwind integration
- **Logging:** Comprehensive logging system using custom logger utility with dedicated log storage
- **Security:** Authentication middleware with JWT token verification and role-based access control
- **File Management:** Upload middleware for handling file uploads with component documentation

Future Expansion Capabilities

The modular architecture supports seamless integration of advanced features:

- **Advanced Barcode/QR Integration:** Enhanced BarcodeScanner and QRCodeGenerator components with inventory automation
- **Supplier Management:** Extending component model relationships with supplier tracking and automated reordering
- **Enhanced Approval Workflows:** Multi-stage approval processes for sensitive component transactions
- **Mobile Applications:** Progressive Web App implementation for mobile access
- **Advanced Import/Export:** Enhanced data transfer capabilities via the ImportExport component
- **Predictive Analytics:** Inventory forecasting based on historical usage patterns
- **Enhanced Dashboard:** Further expansion of the EnhancedDashboard with customizable widgets and alerts

Deployment Architecture

Production Setup: Separate client and server deployments with Vite-optimized frontend build, MongoDB database configured via db.js, with authentication secured through JWT tokens generated via generateToken utility. The system uses scheduled tasks for regular maintenance operations and includes comprehensive error logging via the custom logger utility.