Software Architecture and Design Specification

**Project:** Software Change Management System (SCMS)

**Version:** 1.0

**Authors:** Manchikanti Sharvani (PES1UG23AM165), N Riddhi Samitha (PES1UG23AM175), K S Niharika (PES1UG23AM132)

**Date:** October 02, 2025

**Status:** Draft

# Revision History

| **Version** | **Date** | **Author** | **Change Summary** |
| --- | --- | --- | --- |
| 1.0 | 02-10-2025 | Manchikanti Sharvani, N Riddhi Samitha, K S Niharika | Initial architecture document for academic implementation |

# Approvals

| **Role** | **Name** | **Signature/Date** |
| --- | --- | --- |
| Project Guide | Prof. Anand M S |  |

# 1. Introduction

### 1.1 Purpose

This document specifies the architecture and design of the Software Change Management System (SCMS) as an academic project. It provides a practical, implementable view of the system's structure suitable for a 2-month development timeline.

### 1.2 Scope

This document covers:

* Simplified architecture suitable for rapid prototyping
* Core functional components for demonstration
* Essential integrations (Git webhook simulation, basic CI/CD)
* Security fundamentals
* Technology stack appropriate for student implementation

**Out of Scope for Initial Implementation:**

* Enterprise-scale features (full Kubernetes orchestration, multi-region)
* Complex ML-based analytics
* Production-grade high availability
* Multiple CI/CD platform integrations (focus on one)

### 1.3 Audience

Project team members, course instructors, and evaluators.

### 1.4 Definitions

* **SCMS**: Software Change Management System
* **CR**: Change Request
* **API**: Application Programming Interface
* **MVC**: Model-View-Controller
* **REST**: Representational State Transfer
* **JWT**: JSON Web Token

# 2. Document Overview

### 2.1 How to Use This Document

* Section 3 describes the practical architecture for 2-month implementation
* Section 4 provides design details with focus on implementable features
* Use technology stack recommendations to set up development environment quickly
* Reference the simplified component descriptions for development task allocation

### 2.2 Related Documents

* SCMS Software Requirements Specification (SRS) v1.0
* SCMS Software Test Plan (STP) v1.0
* Requirements Traceability Matrix (RTM)

# 3. Architecture

**3.1 Goals & Constraints**  
**Goals for 2-Month Timeline**

1. **Core Functionality**: CR submission, basic approval workflow, status tracking
2. **Essential Integration**: Git webhook simulation, basic deployment tracking
3. **Basic Security**: User authentication, role-based access, audit logging
4. **Demonstration Ready**: Working prototype with 5-10 key features
5. **Learning Focus**: Understanding change management concepts practically

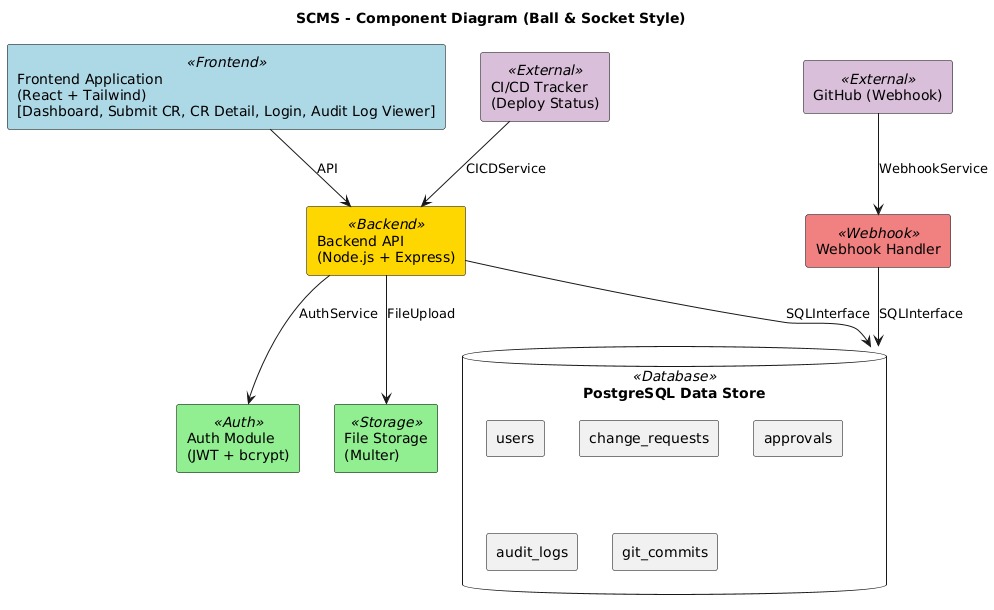
#### Practical Constraints

* **Time**: 2 months with limited daily hours due to coursework
* **Team Size**: 3 undergraduate students
* **Resources**: Free-tier cloud services or local development
* **Complexity**: Avoid over-engineering; focus on core features
* **Learning Curve**: Team learning technologies while building

**3.2 Stakeholders & Concerns**

| **Stakeholder** | **Primary Concerns** |
| --- | --- |
| **Project Team** | Feasible implementation, learning outcomes, clear task division |
| **Course Instructor** | Understanding of concepts, working prototype, documentation quality |
| **Developers (simulated)** | Submit CRs, link to Git commits |
| **Approvers (simulated)** | Review and approve/reject CRs |

## 3.3 Component (UML) Diagram



## 3.4 Component Descriptions

#### 3.4.1 Frontend Application

* **Technology**: React with React Router
* **Purpose**: User interface for all interactions
* **Key Pages**:
  + Dashboard (CR list with status)
  + Submit CR Form
  + CR Detail View with approval actions
  + Simple reports page
* **State Management**: React Context API or Redux (simple setup)
* **Styling**: Tailwind CSS or Material-UI for rapid development

#### 3.4.2 Backend API Server

* **Technology**: Node.js + Express
* **Purpose**: Business logic, API endpoints, data management
* **Key Responsibilities**:
  + CR CRUD operations
  + User authentication (JWT-based)
  + Workflow state management
  + Basic Git webhook receiver
  + Simple audit logging

#### 3.4.3 Database

* **Technology**: PostgreSQL
* **Purpose**: Persist all application data
* **Key Tables**:
  + users (id, email, password\_hash, role)
  + change\_requests (id, cr\_number, title, description, status, category, created\_by, created\_at)
  + approvals (id, cr\_id, approver\_id, decision, comment, approved\_at)
  + audit\_logs (id, action, user\_id, cr\_id, timestamp, details)
  + git\_commits (id, cr\_id, commit\_sha, message, repository, committed\_at)

#### 3.4.4 Authentication Module

* **Technology**: JWT tokens with bcrypt for password hashing
* **Purpose**: User login and session management
* **Implementation**:
  + Login endpoint returns JWT token
  + Protected routes verify token
  + Basic RBAC (Developer, Approver, Admin roles)

#### 3.4.5 Git Integration (Simplified)

* **Technology**: Mock webhook receiver or simple GitHub webhook
* **Purpose**: Demonstrate commit linking
* **Implementation**:
  + Webhook endpoint receives commit data
  + Extracts CR ID from commit message
  + Links commit to CR in database
  + Updates CR status if configured

## 3.5 Chosen Architecture Pattern and Rationale

#### Selected Pattern: Monolithic MVC with Modular Services

**Rationale for Academic Project:**

1. **Simplicity**: Single codebase, easier to understand and debug
2. **Rapid Development**: No distributed system complexity
3. **Resource Efficient**: Runs on single server/laptop for demo
4. **Team Coordination**: Easier for 3-person team to collaborate
5. **Learning Focus**: Concentrate on domain logic, not infrastructure

**Architecture Decision:**

**ADR-001: Monolithic Architecture for Academic Prototype**

* **Context**: 2-month timeline, 3 undergraduate students, learning project
* **Decision**: Build monolithic application with clear module separation
* **Consequences**:
  + Faster development and debugging
  + Simpler deployment (single application)
  + Lower infrastructure complexity
  + Not production-ready for large scale
  + Limited independent service scaling

**ADR-002: Node.js + Express for Backend**

* **Context**: Need rapid development with simple async I/O
* **Decision**: Use Node.js with Express framework
* **Consequences**:
  + JavaScript full-stack (same language as frontend)
  + Large ecosystem (npm packages)
  + Easy webhook handling
  + Fast prototyping
  + Less strict typing (can use TypeScript to mitigate)

## 3.6 Technology Stack & Data Stores

#### Simplified Technology Stack

| **Layer** | **Technology** | **Justification** |
| --- | --- | --- |
| **Frontend** | React 18 + React Router + Axios | Standard modern web framework, large community support |
| **UI Framework** | Tailwind CSS or Material-UI | Rapid UI development with pre-built components |
| **Backend** | Node.js 18+ with Express.js | JavaScript full-stack, easy async operations, fast setup |
| **Database** | PostgreSQL 13+ | Free, robust, JSON support, good for learning SQL |
| **Authentication** | jsonwebtoken + bcrypt | Industry-standard JWT auth, secure password hashing |
| **Git Integration** | Simple webhook endpoint | Simulated or real GitHub webhook for demo |
| **File Upload** | Multer (Node.js middleware) | Simple file handling for CR attachments |
| **Testing** | Jest (backend) + React Testing Library | Built-in with Node/React projects |
| **API Documentation** | Postman Collection | Easy to share and test APIs |
| **Deployment** | Heroku, Render, or Railway (free tier) | Simple deployment for demos |
| **Version Control** | Git + GitHub | Source code management, collaboration |

## 3.7 Risks & Mitigations

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| **Insufficient time** | High | Focus on core features only; use MVP approach |
| **Technology learning curve** | Medium | Start with tutorials; use familiar technologies where possible |
| **Scope creep** | High | Strict feature prioritization; defer non-essential features |
| **Integration complexity** | Medium | Simulate external systems if real integration too complex |
| **Team coordination** | Medium | Daily standups, clear task division, Git workflow |
| **Database design errors** | Medium | Design schema early, get feedback, use migrations |

## 3.8 Traceability to Requirements

**Core Features for 2-Month Implementation:**

| **Requirement ID** | **Priority** | **Component** | **Implementation Notes** |
| --- | --- | --- | --- |
| SCMS-F-001 | **HIGH** | Backend + Frontend | CR submission with basic file upload |
| SCMS-F-002 | **HIGH** | Backend | Auto-generate CR-YYYY-NNNN format |
| SCMS-F-003 | **HIGH** | Backend | Category selection (Emergency/Standard/Normal) |
| SCMS-F-007 | **HIGH** | Backend | Basic approval workflow (1-step for simplicity) |
| SCMS-F-008 | **HIGH** | Backend + Frontend | Approve/reject with comments |
| SCMS-F-010 | **MEDIUM** | Backend | Simulated Git webhook or manual commit linking |
| SCMS-F-018 | **HIGH** | Backend + Frontend | Status tracking display |
| SCMS-SR-001 | **MEDIUM** | Backend | HTTPS in deployment (handled by hosting platform) |
| SCMS-SR-002 | **LOW** | Defer | MFA not essential for academic demo |
| SCMS-SR-004 | **HIGH** | Backend | Basic audit logging to database |

## 3.9 Security Architecture

#### Realistic Security for Academic Project

**Essential Security Controls:**

1. **Authentication**:
   * Password hashing with bcrypt (12 rounds)
   * JWT tokens with 24-hour expiry
   * Login endpoint with basic rate limiting
   * Secure password requirements (min 8 characters, at least one number)
2. **Authorization (RBAC)**:
   * Three distinct roles: Developer, Approver, Admin
   * Role-based middleware for protected routes
   * Resource ownership verification for edit/delete operations
   * Admin-only access to audit logs and user management
3. **Data Protection**:
   * HTTPS in production (free with Heroku/Render)
   * Parameterized SQL queries (prevent SQL injection)
   * Input validation with joi or express-validator
   * Output sanitization to prevent XSS
4. **Audit Trail**:
   * Log all CR create/update/approve actions with user ID
   * Log authentication attempts (success and failures)
   * Log authorization failures (access denied scenarios)
   * Store user ID and timestamp for all actions
   * Admin-only audit log viewer

**Simplified Threat Mitigation:**

| **Threat** | **Basic Mitigation** | **Implementation** |
| --- | --- | --- |
| **SQL Injection** | Parameterized queries | Use pg library with $1, $2 placeholders |
| **XSS** | Input validation + React escaping | React escapes by default; validate all inputs |
| **CSRF** | SameSite cookies | Set cookie options: sameSite: 'strict' |
| **Password Attacks** | Bcrypt + password requirements | Min 8 chars, complexity rules, bcrypt 12 rounds |
| **Unauthorized Access** | JWT + RBAC middleware | Verify token and role on all protected routes |
| **Privilege Escalation** | Role verification + audit logging | Check role in JWT, log all access attempts |
| **Session Hijacking** | Short token expiry | 24-hour JWT expiry, secure token storage |

# 4. Design

## 4.1 Design Overview

**Layered Architecture:**

1. **Presentation Layer**: React components
2. **API Layer**: Express.js routes
3. **Business Logic Layer**: Service functions
4. **Data Access Layer**: Database queries
5. **Database Layer**: PostgreSQL

**Folder Structure (Backend):**

**scms-backend/**

**├── src/**

**│ ├── routes/ # Express route handlers**

**│ │ ├── auth.js**

**│ │ ├── changeRequests.js**

**│ │ └── approvals.js**

**│ ├── controllers/ # Request/response logic**

**│ ├── services/ # Business logic**

**│ ├── models/ # Database queries**

**│ ├── middleware/ # Auth, validation**

**│ ├── utils/ # Helper functions**

**│ └── server.js # App entry point**

**├── tests/**

**├── package.json**

└**── .env # Environment variables**

**Folder Structure (Frontend):**

**scms-frontend/**

**├── src/**

**│ ├── components/ # Reusable UI components**

**│ ├── pages/ # Page components**

**│ │ ├── Dashboard.jsx**

**│ │ ├── SubmitCR.jsx**

**│ │ ├── CRDetail.jsx**

**│ │ └── Login.jsx**

**│ ├── services/ # API calls**

**│ ├── context/ # Auth context**

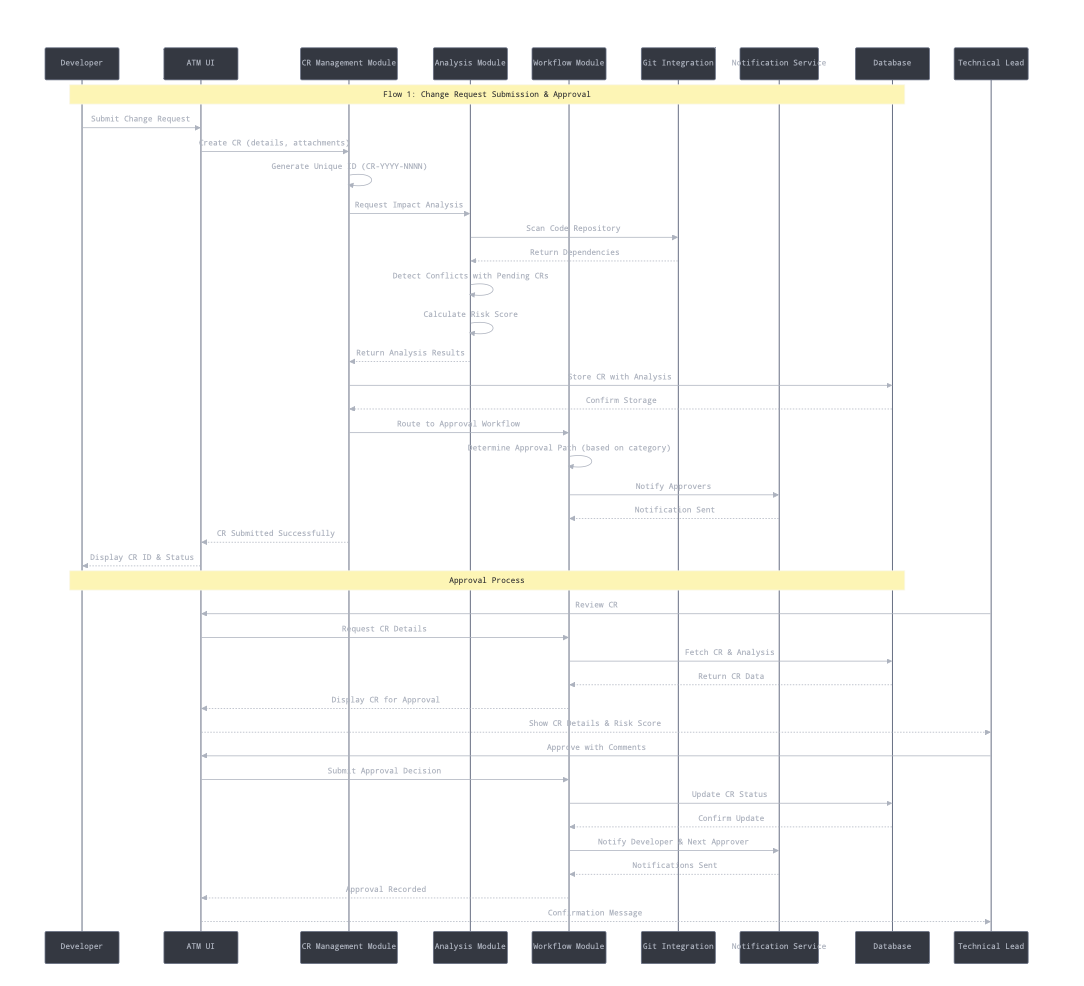
**│ ├── App.jsx**

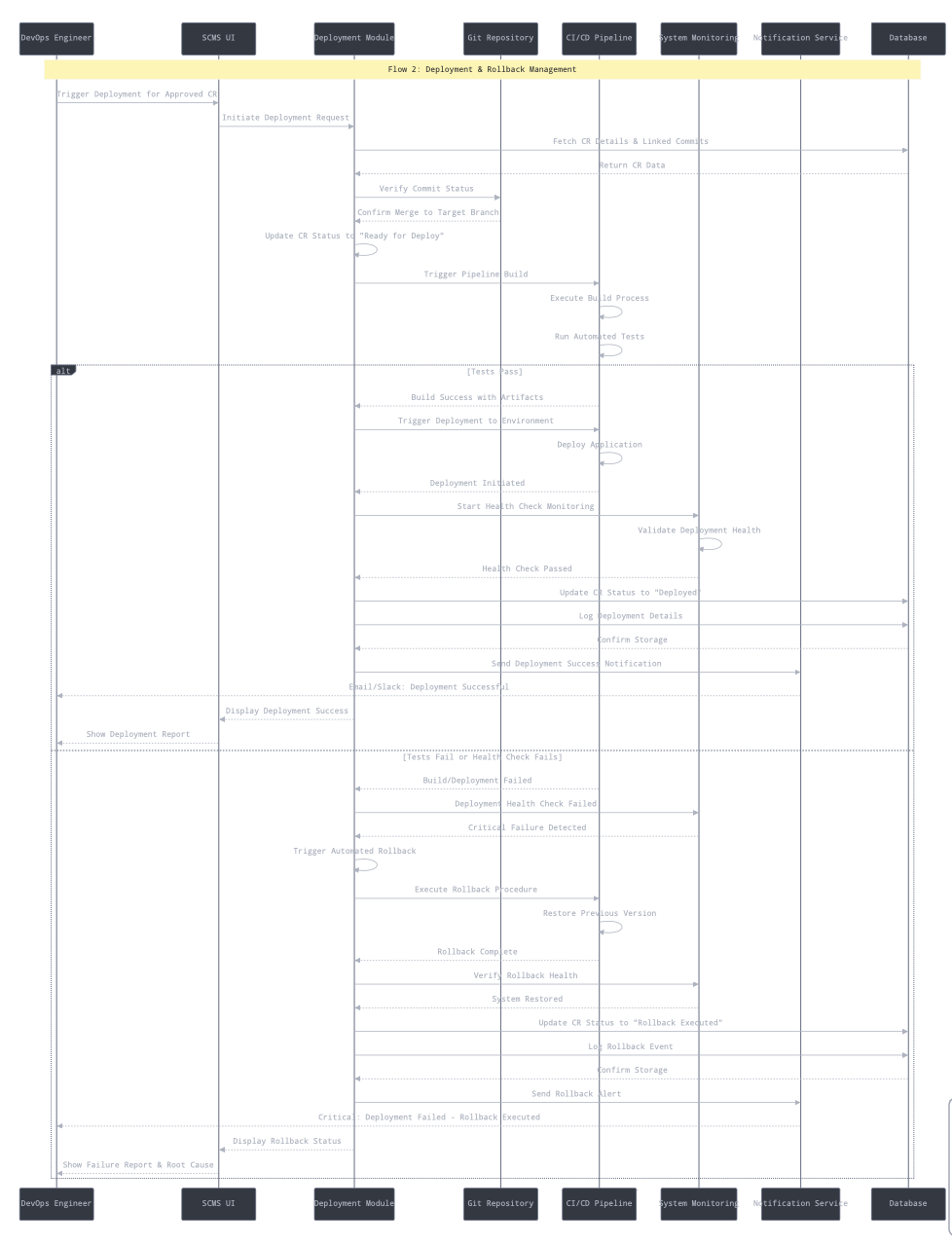
**│ └── index.jsx**

**├── public/**

**└── package.json**

## 4.2 UML Sequence Diagrams





## 4.3 API Design

#### Core API Endpoints (RESTful)

**Authentication:**

POST /api/auth/register

POST /api/auth/login

GET /api/auth/me (requires JWT)

**Change Requests:**

GET /api/change-requests (list all, with filters)

GET /api/change-requests/:id (get single CR)

POST /api/change-requests (create new)

PUT /api/change-requests/:id (update - owner only)

DELETE /api/change-requests/:id (delete draft - owner only)

**Approvals:**

POST /api/change-requests/:id/approve

POST /api/change-requests/:id/reject

GET /api/change-requests/:id/approvals

**Audit Logs:**

GET /api/audit-logs (admin only)

GET /api/audit-logs?cr\_id=:id

**Git Integration (Webhook):**

POST /api/webhooks/git (receives commit data)

## 4.4 Error Handling, Logging & Monitoring

#### Error Handling (Simplified)

**Standard Error Response:**

json

{

"error": "Validation Error",

"message": "Title is required and must be between 10-500 characters",

"statusCode": 400

}

**Common HTTP Status Codes:**

* 200 OK - Success
* 201 Created - Resource created
* 400 Bad Request - Validation error
* 401 Unauthorized - Invalid or missing token
* 403 Forbidden - Insufficient permissions
* 404 Not Found - Resource doesn't exist
* 500 Internal Server Error - Unexpected error

**Backend Error Handling Pattern:**

javascript

try {

*// Business logic*

} catch (error) {

console.error('Error creating CR:', error);

res.status(500).json({

error: 'Internal Server Error',

message: 'Failed to create change request'

});

}

#### Logging (Simplified)

**Console Logging** (adequate for academic project):

javascript

console.log('[INFO]', 'CR created:', crNumber);

console.error('[ERROR]', 'Database error:', error.message);

#### Monitoring (Basic)

* Console logs during development
* Heroku/Render logs viewer in production
* Manual testing and debugging
* Optional: Simple health check endpoint

## 4.5 UX Design

## Simplified User Interface

**1. Login Page**

* Email and password fields
* Login button
* Link to register (if implementing registration)
* Basic validation messages

**2. Dashboard**

* Table showing all CRs with columns:
  + CR Number
  + Title
  + Category
  + Status
  + Created By
  + Created Date
  + Actions (View button)
* Filter by status (Draft, Pending, Approved, Rejected)
* "Submit New CR" button prominently placed

**3. Submit CR Form**

* Simple form with fields:
  + Title (required)
  + Description (textarea)
  + Category (dropdown: Emergency, Standard, Normal)
  + Priority (dropdown: High, Medium, Low)
  + Impact Scope (text input or multi-select)
* Submit and Cancel buttons
* Client-side validation with error messages

**4. CR Detail View**

* Display all CR information
* Approval section (if user is Approver and CR is Pending):
  + Comment textarea
  + Approve and Reject buttons
* Approval history (list of past approvals)
* Linked commits section (if any)
* Edit button (if owner and status is Draft)

**5. Audit Log Viewer (Admin)**

* Simple table with:
  + Timestamp
  + Action
  + User
  + CR Number
  + Details
* Basic date filter

#### Accessibility Basics

* Semantic HTML (proper headings, labels)
* Form labels for screen readers
* Keyboard navigation (tab order)
* Sufficient color contrast
* Responsive design (mobile-friendly)

## 4.6 Open Issues & Next Steps

#### Implementation Plan (2 Months)

**Week 1-2: Setup & Core Backend**

* Set up project repositories (frontend & backend)
* Initialize Node.js + Express backend
* Set up PostgreSQL database
* Implement user authentication (register, login, JWT)
* Create database schema and migrations
* **Deliverable**: Working auth system

**Week 3-4: CR Management**

* Implement CR CRUD APIs
* Build React frontend structure
* Create Dashboard and Submit CR pages
* Implement CR listing and creation
* **Deliverable**: Can create and view CRs

**Week 5-6: Approval Workflow**

* Implement approval endpoints
* Build CR Detail page with approve/reject
* Add audit logging
* Implement role-based access control
* **Deliverable**: Complete approval workflow

**Week 7: Integration & Polish**

* Implement simulated Git webhook
* Add basic commit linking
* Improve UI/UX
* Bug fixes and testing
* **Deliverable**: Functional prototype

**Week 8: Testing & Documentation**

* Write test cases (at least 20 key tests)
* Prepare demo scenarios
* Complete documentation
* Prepare presentation
* Deploy to free hosting platform
* **Deliverable**: Demo-ready application

#### Success Criteria

**Minimum Viable Demo:**

1. User can register and login
2. User can create a CR with all required fields
3. CR gets unique number (CR-2025-NNNN)
4. Approver can view CR and approve/reject with comment
5. CR status updates based on approval
6. Dashboard shows all CRs with current status
7. Basic audit log shows key actions
8. Simulated Git commit can be linked to CR

**Bonus Features (if time permits):** 9. File upload for CR attachments 10. Email notifications (using nodemailer) 11. Advanced dashboard with statistics 12. Export CR data to CSV 13. Password reset functionality

# 5. Appendices

### 5.1 Glossary

| **Term** | **Definition** |
| --- | --- |
| **CR** | Change Request - formal request to modify software |
| **JWT** | JSON Web Token - authentication token format |
| **REST** | Representational State Transfer - API architecture style |
| **CRUD** | Create, Read, Update, Delete - basic data operations |
| **MVP** | Minimum Viable Product - simplest version with core features |

### 5.2 References

1. Express.js Documentation:<https://expressjs.com/>
2. React Documentation:<https://react.dev/>
3. PostgreSQL Tutorial:<https://www.postgresqltutorial.com/>
4. JWT.io:<https://jwt.io/>
5. REST API Best Practices
6. SCMS SRS v1.0
7. SCMS STP v1.0

### 5.3 Tools & Setup

#### Development Tools

| **Tool** | **Purpose** | **Setup Time** | **Link** |
| --- | --- | --- | --- |
| **VS Code** | Code editor | 30 min | <https://code.visualstudio.com/> |
| **Node.js 18+** | Runtime | 30 min | <https://nodejs.org/> |
| **PostgreSQL** | Database | 1 hour | <https://www.postgresql.org/> |
| **Postman** | API testing | 30 min | <https://www.postman.com/> |
| **Git + GitHub** | Version control | 1 hour | <https://github.com/> |
| **React DevTools** | Debugging | 15 min | Browser extension |

#### UML & Diagram Tools

| **Tool** | **Purpose** | **Recommended For** | **Learning Curve** |
| --- | --- | --- | --- |
| **Draw.io (diagrams.net)** | Component diagrams, sequence diagrams | All diagram types | Easy (30 min) |
| **PlantUML** | Text-based UML diagrams | Sequence diagrams | Medium (2 hours) |
| **StarUML** | Professional UML tool | Complete UML suite | Medium (2-3 hours) |