Claims Processing System

Coordinators:
Naveenkumar Kandala
Roopesh Burra
Srigowri N

1. Introduction

This document outlines the software requirements for the Claims Processing System (CPS), designed to efficiently manage insurance claims. The system supports claim submission, validation, tracking, approval workflows, and role-based access control.

2. Actors and Roles

- **Insurance Coordinator**: Assists in claim submissions and tracks claim statuses.
- **Policy Holde**r: Submits claims related to their insurance policies and tracks claim progress.
- **Claims Processing Officer**: Validates claims, verifies documentation, and manages claim approval or rejection.
- **Medical Practitioner / Internal Medical Reviewer**: Verifies treatment authenticity and submits medical proofs linked to claims.
- **Claims Approver / Manager / Regional Head**: Reviews claims and grants final approval or escalates for revalidation.
- **System Administrator/ Admin**: Manages user accounts, roles, and system settings to enforce access control and workflows.

3. Functional Requirements

A. Claim Submission

Allow Claimants, Insurance Coordinators, and Policy Holders to submit new insurance claims with required details and supporting documents. The system supports uploading multiple document types such as images, PDFs, medical reports, and receipts, with version control.

B. Claim Tracking

Enable users to track the status of their claims in real-time through dashboards or notifications. This provides transparency and reduces inquiry calls.

C. Claim Validation

Claims Processing Officers validate claims by checking documentation completeness, verifying policy coverage, and performing fraud checks.

D. Medical Verification

Medical Practitioners and Internal Medical Reviewers verify treatment authenticity and submit medical proofs, which the system securely stores and links to claims.

E. Claim Approval Workflow

Claims Approvers, Managers, and Regional Heads review claims, especially high-value ones, and grant final approval or escalate for revalidation in case of conflicts.

F. User and Role Management

System Administrators create and manage user accounts, assign roles such as Claimant, Claims Officer, Medical Reviewer, and Approver, and configure system settings to enforce access control and workflow rules.

G. Notifications

The system sends automated alerts and updates to users about claim status changes, required actions, approvals, or rejections.

H. Document Management

All claim-related documents are securely stored and managed with access control, search, and retrieval capabilities.

4. User Interface Requirements

The system includes the following user interfaces:

- User Login Page
- Admin Dashboard
- Insurance Coordinator Dashboard
- Claim Processing Officer Dashboard
- Medical Validator Dashboard
- Customer Dashboard
- Regional Head Dashboard

5. Project Execution Related Requirements

Development Environment

- Technologies: React with Material UI (version 22), MongoDB Database
- Software Tools: Enterprise Architect (EA) or EdrawMax for design, VS Code as the primary code editor

Coding Requirements

- Use functional React components with hooks.
- Implement Material UI theming.
- Secure API integration.
- Maintainable and testable code.
- Enforce role-based access control.
- Follow best practices for input validation, error handling, and performance optimization.

6. Resource Requirements

- Technologies: React (Frontend), Spring (Backend), MongoDB (Database)
- Software Tools: VS Code, Enterprise Architect (EA), JDeveloper

7. Module Allocation and Role Responsibilities

- Naveenkumar Kandala: Admin, Roles, Insurance Coordinator, Admin Login
- Roopesh Burra: Claims Processing Officer, Medical Validator
- Srigowri N: Policy Holder, Regional Head, Policy Holder Login

8. Training Requirements

End-user training will be conducted to equip claimants, claims officers, medical reviewers, approvers, and system administrators with the necessary skills. Training includes hands-on sessions, role-based walkthroughs, and comprehensive user manuals.

9. Testing Requirements

A comprehensive testing strategy includes:

- Unit Testing of individual components and modules such as claim submission forms, document uploads, validation logic, and approval workflows.
- Integration Testing to verify frontend and backend interactions.
- User Acceptance Testing (UAT) to ensure the system meets business needs and user expectations.

10. Software Acceptance Criteria

The system will be accepted upon:

- Successful completion of UAT.
- Implementation and testing of all documented functional requirements.
- Demonstrated system stability and performance within defined parameters.
- Formal approval by key stakeholders.

11. Installation and Setup

Prerequisites

- Node.js (version 16 or higher recommended) - **[node 22.15.1]**
- npm (comes with Node.js) - **[npm 10.9.2]**
- VS Code (recommended editor)

Frontend Setup (React + Vite)

Create a new React project using Vite: `npm create vite@latest my-app -- --template react`

Navigate to the project directory: `cd my-app`

Install project dependencies: `npm install`

Start the development server: `npm run dev`

React Router Installation

Install React Router v6: `npm install react-router-dom@6`

JSON Server Setup

Install JSON Server globally to mock backend APIs: `npm install -g json-server`
Start JSON Server with your mock data file (`db.json`): `json-server --watch db.json --port 9090`

Install npm commands

- React and ReactDOM: Core libraries
  `npm install react react-dom`
- Material UI (MUI) Core and Icons
  `npm install @mui/material @mui/icons-material`
  `@emotion/react`
  `@emotion/styled`
- React Router DOM (v6)
  `npm install react-router-dom@6`

- Proptypes - Used for type-checking React component props to catch bugs and document component APIs.
  `npm install prop-types`
- Toolpad Core - Background Theme
  `npm install @toolpad/core`
- Material UI Lab - Date picker
  `npm install @mui/lab`
- PDF - Viewer
  `npm install @react-pdf-viewer/core`

Plugin - Fast refresh

```
-[@vitejs/plugin-react](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react) uses [Babel](https://babeljs.io/) for Fast Refresh
```

12. Running the Application

- Ensure JSON Server is running on port 9090 to serve mock APIs.
- Run the frontend development server with Vite.
- Open your browser and navigate to `http://localhost:9090` (or the port Vite outputs).

13. Project Structure Overview

- `src/components` — Reusable UI components such as tabs, dialogs, and document previews.
- `src/pages` — Main pages including dashboards and claim management.
- `src/components/[rolename]` — Layout components for different user roles.
- `db.json` — Mock database file for JSON Server.
- `package.json` and `vite.config.js` — Project configuration files.

14. Contact - For questions or support, please contact:

- Naveenkumar Kandala
- Roopesh Burra
- Srigowri N

Document generated on Monday, July 07, 2025