

# Technical Documentation - Cooperative Management System (React Version)

## 1. Introduction

### Application Description

The **Cooperative Management System** is a comprehensive web and mobile-first platform for managing home-based social and healthcare services. The application automates service hour tracking, budget management, invoice generation, and monthly closing processes with tax integration via Fatture in Cloud. It is designed with responsive user experience in mind and is built using modern frontend technologies.

### Purpose and Key Features

- **Advanced Budget Management:** Multi-level system for planning and monitoring care budgets
- **Hours Tracking:** Detailed work hour logging with automatic cost calculation
- **Home Budget Planner:** Intelligent tool for optimal resource distribution
- **Staff Management:** Full collaborator management with differentiated rates
- **Client Management:** Complete client directory with service history
- **Tax Integration:** Automated connection with Fatture in Cloud for document generation
- **Reporting and Analytics:** Detailed dashboards and reports for performance analysis
- **Email Notifications:** Transactional emails and reminders using external email provider
- **Firebase Integration:** Push notifications, analytics, and authentication support
- **Claude Integration:** Intelligent assistant and automation for planning and documentation
- **Language Toggle (EN/IT):** Seamless switch between English and Italian using i18n

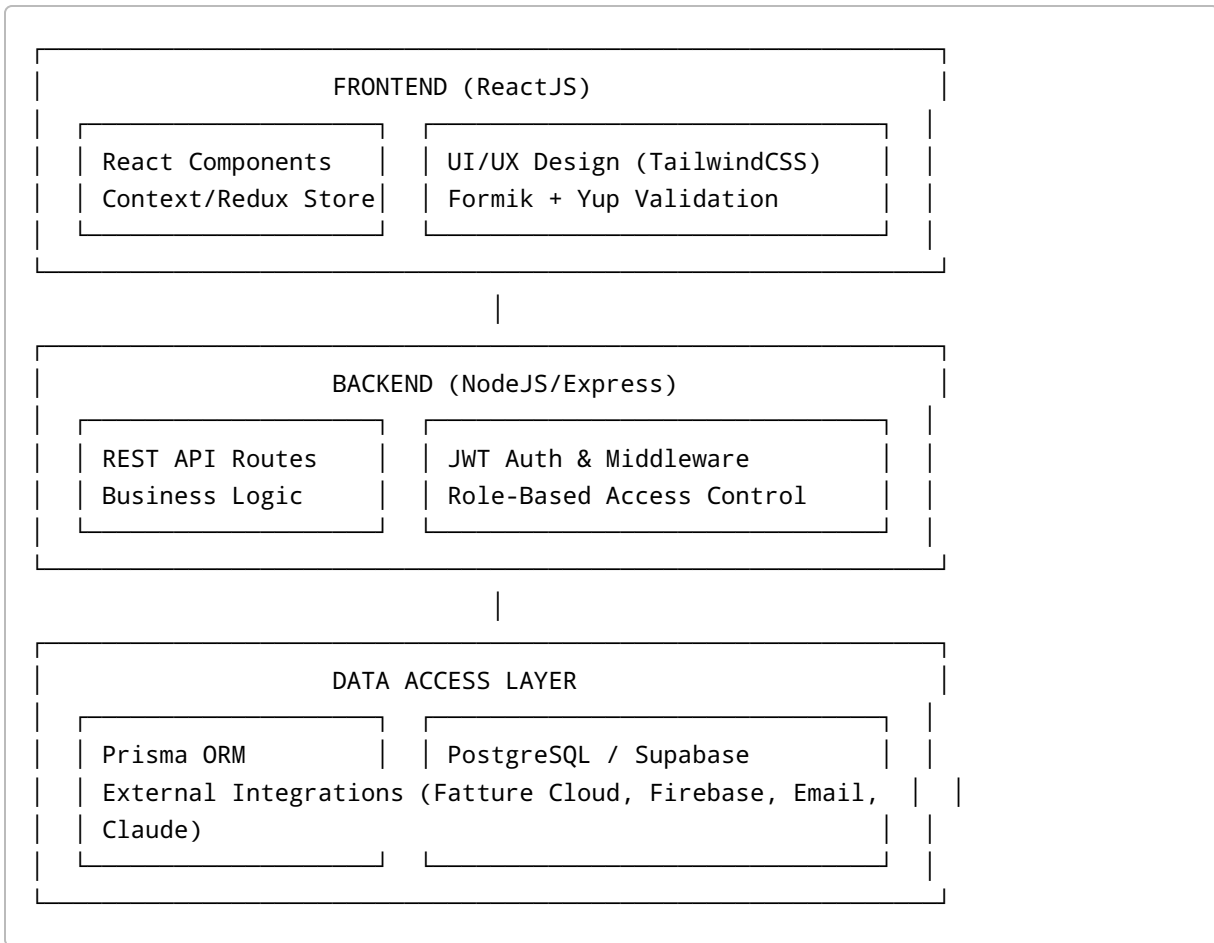
### Distinctive Features

- **GDPR Compliance:** Full privacy management and data tracking system
- **Intelligent Planning:** Advanced algorithms for optimized budget allocation
- **Multi-Budget Management:** Support for combined budgets and complex planning
- **Integrated Calendar:** Assistance planning via calendar view
- **Audit Trail:** Full tracking of all critical changes
- **Mobile-First UI:** Optimized experience for mobile and tablet users
- **Language Toggle (English/Italian):** UI toggle for multilingual interface using i18next

## 2. Updated System Architecture

### Architectural Overview

The application follows a **microservices architecture** with a **React frontend**, a **RESTful API backend (Node.js/Express or Flask)**, and **modular service layers**. The mobile-first design supports PWA and integration with React Native for Android/iOS applications.



## Main Technologies

- **Frontend:** ReactJS (with Hooks), TailwindCSS, React Router, Formik, Yup, i18next
- **Mobile App:** React Native (shared business logic)
- **State Management:** Context API / Redux Toolkit
- **Backend:** Node.js (Express) or Flask (Python)
- **Database:** PostgreSQL (optionally Supabase)
- **Auth:** JWT-based authentication
- **APIs:** RESTful APIs, OAuth2.0 (Fatture in Cloud, Claude)
- **Email:** Integration with SendGrid/Mailgun/SMTP
- **Firebase:** Notifications, auth, and analytics
- **Claude:** AI assistant integration for document and workflow automation
- **Internationalization (i18n):** Language support via react-i18next
- **Deployment:** Vercel/Netlify (frontend), Railway/Render (backend), Docker (optional)

## 3. Goals & Best Practices

### Mobile-First & Progressive Web App

- Start from mobile screen size and scale up

- Use React Responsive or Tailwind breakpoints
- Implement PWA features for offline support

## Design & UX Principles

- Minimal, accessible, and responsive design
- Intuitive navigation and clear user feedback
- Use Material Design or custom UI kit (accessible WCAG 2.1)
- Use wireframes and UI prototypes (e.g., Figma)



## Dev Best Practices

- Modular code structure
- DRY & reusable components
- Form validation with Formik + Yup
- Test coverage: Unit (Jest), Integration (Cypress)
- CI/CD setup for deployment pipelines
- ESLint + Prettier for code formatting

# 4. System Requirements

## Functional Requirements

Code	Requirement	Priority	Status
FR-001	The system must support user authentication with persistent session.	High	✓ Completed
FR-002	Users must be able to record monthly service hours with automatic cost calculation.	High	✓ Completed
FR-003	Each client must have 10 mandatory budget categories assigned.	High	✓ Completed
FR-004	The system must sync receipts with Fatture in Cloud.	High	✓ Completed
FR-005	The system must include a planner for automatic budget distribution.	High	✓ Completed
FR-006	Daily automatic backups must be supported.	Medium	✓ Completed
FR-007	API errors must be logged and display appropriate user alerts.	Medium	🔄 In Progress
FR-008	The user must be able to export reports as PDF and Excel files.	Medium	✗ Pending
FR-009	The user must be able to toggle interface language (IT/EN).	Medium	✗ Pending
FR-010	The app must support push notifications using Firebase.	Low	✗ Pending

Code	Requirement	Priority	Status
FR-011	Admin can manage collaborators and clients using dedicated CRUD modules.	High	 Completed
FR-012	Mobile users must experience a responsive UI across all screens.	High	 In Progress

## Non-Functional Requirements

Code	Requirement	Description
NFR-001	Security	Must use HTTPS, input validation, token-based auth (JWT), and CSRF protection
NFR-002	Performance	API response time < 200ms, page load < 2 seconds
NFR-003	Backup	Automatic daily backups with 30-day retention
NFR-004	Availability	99.9% uptime with system health check endpoints
NFR-005	Compatibility	Works on all modern browsers and mobile devices
NFR-006	Scalability	Modular frontend with REST API backend for future microservice migration
NFR-007	Localization	Interface supports English and Italian via i18n (react-i18next)
NFR-008	Maintainability	Code must follow industry standards (ESLint, Prettier, folder structure)
NFR-009	Accessibility	Meets WCAG 2.1 AA standards for screen readers and keyboard navigation

## Notes for React Ecosystem

- **Authentication:** Use JWT with role-based UI rendering (React Context/Redux)
- **Internationalization:** `react-i18next` for EN/IT toggle
- **PDF/Excel Export:** Use `jspdf`, `xlsx`, or `react-pdf`
- **Notifications:** Use Firebase Cloud Messaging (FCM)
- **Responsive Design:** Tailwind CSS with mobile-first layout
- **Error Monitoring:** Use custom toast alerts or integrate with Sentry

## 4. Additional Features to Add

### Phase 1: Core Functionality

- Import and parse Excel file for budgets and planning (High Priority)
- User Auth (Signup/Login/Forgot Password)
- Dashboard (Summaries of budgets, hours, clients)
- Collaborators & Clients CRUD
- Budget Planning Wizard

- Invoice Syncing with Fatture in Cloud
- Role-Based Access (Admin, Manager, Staff)
- Language Toggle (EN ↔ IT) via i18n in frontend

## **Phase 2: Advanced Tools**

- Budget Forecasting & Warnings
- Integrated Calendar with Drag-and-Drop Planning
- Exportable Reports (PDF, Excel)
- Data Visualization (Chart.js or Recharts)
- Notifications & Alerts System
- Session Logs and Audit Trail Viewer
- Email Notification System
- Firebase Push Notifications & Analytics

## **Phase 3: Mobile App Integration**

- Build React Native version using same backend
- Push notifications (Firebase)
- Offline mode sync for mobile
- Geolocation and route planning (for staff)
- Claude integration for planning automation, AI assistance, and summaries

Let me know if you'd like the codebase folder structure, backend API planning, or React component architecture outlined next.