## Strategic Goal Management System (SGMS) Development Plan

This plan outlines a 4-week timeline (3 weeks development + 1 week testing) for a two-person team. The backend (Node.js/Express + PostgreSQL) and frontend (React + Tailwind) tasks are paired each week to deliver all core features: authentication, RBAC, goal/task/activity modules, reporting, notifications, i18n, and settings. Practical DevOps and version-control tips are included.

### Week 1: Project Setup, Authentication & RBAC

**Backend (Node.js/Express + PostgreSQL):** - Initialize project and Git repo - Set up basic folder structure (/models, /controllers, /routes) - Install Express, node-postgres or ORM (Sequelize, TypeORM) - Create PostgreSQL schema and tables: users, roles, etc. - Implement JWT-based authentication with bcrypt password hashing - Design RBAC system: roles (Admin, Manager, User) with middleware to protect routes

**Frontend (React + Tailwind):** - Initialize app (Create React App or Vite) with Tailwind CSS - Set up Git branches (backend-setup, frontend-setup) - Build Login and Registration pages - Implement global state for JWT and user role - Create header/menu with role-based navigation visibility - Stub out empty pages for Goals, Tasks, Reports, etc.

**Milestones:** - Authentication fully working (login, token handling) - Backend role-based middleware functional - Frontend correctly stores and responds to user roles - Git repo clean with separate feature branches

### Week 2: Core Modules (Goals, Tasks, Activities), Notifications & i18n

**Backend:** - Create database tables: goals, tasks, activities (linked by foreign keys) - Implement CRUD APIs: POST/GET/PUT/DELETE for each entity - Add RBAC enforcement on API routes - Create notifications table and routes for creating/fetching messages - Integrate Nodemailer for email alerts - Add i18next or similar for multi-language support (load translation JSONs) - Add settings module: store user language preferences, notification toggles

**Frontend:** - Build pages and components for Goals, Tasks, Activities - Use Tailwind to style forms/tables/lists - Connect forms and displays to backend via Fetch/Axios - Add notification dropdown with unread indicators - Allow marking notifications as read - Add language toggle (i18n setup with react-i18next or similar) - Create user settings page

**Milestones:** - Full CRUD operations for core modules work from UI - Notifications work and can be seen/read on frontend - Multi-language toggle works end-to-end - Merge week’s feature branches

### Week 3: Reports, SMS Notifications, Finalization

**Backend:** - Create reports table and endpoints (GET, POST, possibly PDF generation) - Implement logic for aggregating goal/task/activity data - Finalize RBAC across all routes - Add SMS gateway integration (e.g. Twilio or SMS.to) - Register account, get API keys - Create sendSms(to, message) helper function - Use +251 format for Ethio Telecom numbers - Trigger SMS alerts for critical events (e.g. deadlines, assignments) - Finalize email alerts - Document environment variables and deployment scripts - Optimize PostgreSQL (max\_connections ~100), and prepare for deployment

**Frontend:** - Complete Reports page (tables or charts) - Polish all pages with responsive layout and consistent design - Finalize validation on all forms - Ensure role-based visibility and routing is enforced - Finish all translation and localization work - Run production build and prepare for deployment

**Milestones:** - Reporting module works end-to-end - Email + SMS notifications are triggered properly - Frontend styling complete and validated - App ready for production deployment

### Week 4: Deployment & Testing

**Deployment (Ubuntu 4GB PC):** - Update Ubuntu, install Node.js (LTS), PostgreSQL, nginx - Create app database and user - Clone repo, configure .env file - Install dependencies with npm install - Build frontend, serve with nginx or Express static middleware - Use PM2 to run backend and auto-restart - Configure firewall (only allow ports 22, 80, 443) - Optional: Set up HTTPS with Let’s Encrypt

**User Capacity Estimation:** - With 4GB RAM: handle ~50–100 concurrent users with moderate usage - Light usage (reading) may support several hundred - Heavy DB operations will bottleneck earlier - Monitor with htop, PM2 dashboard

**Final Testing:** - Simulate real usage with multiple roles/users - Test all API routes with Postman or curl - Validate email and SMS delivery - Do load testing if possible - Have frontend teammate review usability and catch edge cases

### Integration & Process Tips

**Version Control:** - Use branches (feature/auth, feature/goals, etc.) - Clear commit messages - Perform basic code reviews - Tag releases before deployment

**Workflow Tips:** - Weekly API contract alignment - Daily syncs or async check-ins - Frontend/backend develop in parallel

**Security:** - Validate and sanitize all inputs - Use HTTPS - Rate-limit sensitive routes (e.g. login) - Hash passwords with bcrypt

**Monitoring:** - Use PM2 logging - Consider uptime monitoring tools

By following this structured plan, you can deliver a stable, feature-rich SGMS in 3 weeks, reserving the 4th for final testing and deployment. This approach ensures you ship a product with authentication, access control, multi-language support, real-time alerts (email/SMS), and core reporting functionality—all within a tight timeline.