Hour Tracker SaaS - Implementation Plan

Project Overview

Multitenant B2B time-tracking SaaS application with AI integration (MCP), deployed on Google Cloud Platform with cost optimization. There are 16 phases.

Tech Stack:

- Frontend: Next.js (App Router) + Tailwind CSS
- Backend: Node.js (NestJS)
- Database: PostgreSQL 16
- Caching: Redis
- Infrastructure: Google Cloud Platform (Cloud Run, Compute Engine)

PHASE 1: PROJECT FOUNDATION & INFRASTRUCTURE SETUP

1. Initialize Project Repository

- Create monorepo structure with appropriate folder hierarchy
- Set up Git repository with .gitignore for Node.js, Next.js, and environment files
- Create README.md with project overview and setup instructions

2. Configure Package Management

- Initialize package.json for root workspace
- Configure npm/yarn workspaces for monorepo structure
- Set up shared dependencies and workspace-specific dependencies

3. Set Up Development Environment Configuration

- Create .env.example template with all required environment variables
- Document environment variable purposes and formats
- Set up .env for local development (git-ignored)

4. Configure TypeScript

- Initialize TypeScript configuration for entire project
- Create tsconfig.json for backend with strict mode enabled
- Create tsconfig.json for frontend with Next.js settings
- Set up path aliases for clean imports

5. Set Up ESLint and Prettier

- Configure ESLint with TypeScript support
- Set up Prettier for consistent code formatting
- Create .eslintrc and .prettierrc configuration files
- Add lint-staged and husky for pre-commit hooks

6. Create Docker Configuration for Local Development

- Write Dockerfile for PostgreSQL 16 with custom configuration
- Write Dockerfile for Redis
- Create docker-compose.yml for local development environment
- Configure volume mounts for database persistence

7. Initialize Database Schema File

- Create initial migration directory structure
- Set up database migration tooling (e.g., TypeORM, Prisma, or Knex)
- Document migration workflow and commands

8. Set Up Terraform for GCP Infrastructure

- Create Terraform configuration for Google Cloud Project
- Define Compute Engine e2-micro instance for database
- Configure Cloud Run services for frontend and backend
- Set up networking and firewall rules

9. Create GCP Service Account and Permissions

- Define IAM roles and permissions
- Create service account JSON keys
- Document security best practices

10. Set Up Cloud Storage Bucket

- Configure Google Cloud Storage for database backups
- Set up lifecycle policies for cost optimization
- Create backup scripts

PHASE 2: DATABASE SCHEMA & MODELS

11. Create Tenants Table Schema

- Define tenants table with id, name, plan, created_at, updated_at
- Add indexes on frequently queried fields

- Create TypeScript interface/type for Tenant entity

12. Create Users Table Schema

- Define users table with id, tenant_id, email, password_hash, role, created_at, updated_at
- Add foreign key constraint to tenants table
- Add unique constraint on email within tenant
- Create indexes on tenant_id and email

13. Create Clients Table Schema

- Define clients table with id, tenant_id, name, deleted_at, created_at, updated_at
- Add foreign key constraint to tenants table
- Add index on tenant_id
- Implement soft delete pattern

14. Create Projects Table Schema

- Define projects table with id, tenant_id, client_id, name, is_billable, deleted_at, created_at, updated_at
- Add foreign key constraints to tenants and clients tables
- Add indexes on tenant_id and client_id
- Implement soft delete pattern

15. Create Tasks Table Schema

- Define tasks table with id, tenant_id, project_id, name, deleted_at, created_at, updated_at
- Add foreign key constraints to tenants and projects tables
- Add index on tenant_id and project_id
- Implement soft delete pattern

16. Create Time Entries Table Schema

- Define time_entries table with id, tenant_id, user_id, project_id, task_id, start_time, end_time, duration, description, deleted_at, created_at, updated_at
- Add foreign key constraints to all related tables
- Add indexes on tenant_id, user_id, project_id, start_time
- Implement soft delete pattern

17. Create Database Seed Data

- Write seed script for development environment
- Create sample tenants, users, clients, projects, and tasks
- Document how to run seed scripts

18. Implement Database Connection Module

- Create database connection configuration
- Set up connection pooling with appropriate limits
- Implement connection health checks
- Add error handling and retry logic

19. Create Database Repository Pattern

- Implement base repository class with common CRUD operations
- Add tenant isolation at repository level
- Implement soft delete filtering in base queries

20. Write Database Migration Scripts

- Create initial migration to create all tables
- Add migration for indexes
- Add migration for foreign key constraints
- Test rollback functionality

PHASE 3: AUTHENTICATION & AUTHORIZATION

21. Set Up NextAuth.js Configuration

- Install and configure NextAuth.js in Next.js app
- Define authentication providers (Email/Password)
- Configure session strategy and JWT settings

22. Create User Registration Endpoint

- Implement POST /api/auth/register endpoint
- Validate email format and password strength
- Hash passwords using bcrypt or argon2
- Create user record with tenant association

23. Create User Login Endpoint

- Implement POST /api/auth/login endpoint
- Verify credentials against database
- Generate JWT token with user and tenant information
- Set up session management

24. Implement Tenant Isolation Middleware

- Create middleware to extract tenant_id from JWT
- Add tenant_id to all database queries automatically
- Prevent cross-tenant data access

25. Create Role-Based Access Control (RBAC) Middleware

- Implement middleware to check user role from JWT
- Define role permissions (Admin, User)
- Protect admin-only routes and operations

26. Implement Password Reset Flow

- Create POST /api/auth/forgot-password endpoint
- Generate secure reset tokens with expiration
- Send password reset emails (setup email service later)
- Create POST /api/auth/reset-password endpoint

27. Add OAuth Provider Support (Optional)

- Configure Google OAuth provider
- Add OAuth callback handler
- Link OAuth accounts to existing users

28. Create Protected Route HOC/Component

- Implement client-side route protection
- Redirect unauthenticated users to login
- Show loading state during authentication check

PHASE 4: CORE ENTITY MANAGEMENT (BACKEND)

29. Create Clients API Endpoints

- Implement GET /api/clients (list with pagination)
- Implement GET /api/clients/:id (single client)
- Implement POST /api/clients (create)
- Implement PUT /api/clients/:id (update)
- Implement DELETE /api/clients/:id (soft delete)

30. Add Client Input Validation

- Validate required fields (name)
- Sanitize input to prevent XSS
- Return clear error messages

31. Create Projects API Endpoints

- Implement GET /api/projects (list with filtering by client)
- Implement GET /api/projects/:id (single project)

- Implement POST /api/projects (create with client association)
- Implement PUT /api/projects/:id (update)
- Implement DELETE /api/projects/:id (soft delete)

32. Add Project Input Validation

- Validate required fields (name, client_id, is_billable)
- Verify client exists and belongs to same tenant
- Return clear error messages

33. Create Tasks API Endpoints

- Implement GET /api/tasks (list with filtering by project)
- Implement GET /api/tasks/:id (single task)
- Implement POST /api/tasks (create with project association)
- Implement PUT /api/tasks/:id (update)
- Implement DELETE /api/tasks/:id (soft delete)

34. Add Task Input Validation

- Validate required fields (name, project_id)
- Verify project exists and belongs to same tenant
- Return clear error messages

35. Create People/Users Management Endpoints (Admin Only)

- Implement GET /api/users (list all users in tenant)
- Implement POST /api/users/invite (send invitation email)
- Implement PUT /api/users/:id (update user role)
- Implement DELETE /api/users/:id (deactivate user)

36. Add User Management Input Validation

- Validate email format for invitations
- Validate role values (admin, user)
- Prevent admin from removing their own admin role

PHASE 5: TIME ENTRIES (BACKEND)

37. Create Time Entries API Endpoints

- Implement GET /api/time-entries (list with filtering)
- Implement GET /api/time-entries/:id (single entry)
- Implement POST /api/time-entries (create)
- Implement PUT /api/time-entries/:id (update)

- Implement DELETE /api/time-entries/:id (soft delete)

38. Add Time Entry Input Validation

- Validate required fields (project_id, start_time, end_time or duration)
- Calculate duration from start_time and end_time if not provided
- Ensure end_time is after start_time
- Validate time entry doesn't overlap with existing entries

39. Implement Time Entry Filtering

- Add query parameters for date range filtering
- Add filtering by user_id (admins can filter by any user)
- Add filtering by project_id, client_id
- Add pagination for large result sets

40. Implement Time Entry Authorization

- Users can only create/edit/delete their own time entries
- Admins can create/edit/delete any time entry
- Implement these checks in middleware

41. Create Bulk Operations Endpoint

- Implement POST /api/time-entries/bulk (create multiple entries)
- Add transaction support to ensure all-or-nothing behavior
- Return detailed error messages for failed entries

42. Add Time Entry Duration Calculation Helper

- Create utility function to calculate duration from timestamps
- Format duration in hours and minutes
- Handle timezone conversions properly

PHASE 6: FRONTEND SETUP

43. Initialize Next.js Application

- Set up Next.js with App Router
- Configure TypeScript for frontend
- Set up folder structure (app, components, lib, types)

44. Install and Configure Tailwind CSS

- Install Tailwind CSS and dependencies

- Configure tailwind.config.js with custom theme
- Set up global CSS file
- Add common utility classes

45. Create Design System Components

- Button component with variants (primary, secondary, danger)
- Input component with validation states
- Select/Dropdown component
- Modal component
- Toast/Notification component
- Loading spinner component

46. Set Up API Client/Fetch Wrapper

- Create axios or fetch wrapper with base URL
- Add automatic JWT token attachment to requests
- Implement request/response interceptors
- Add error handling and retry logic

47. Create Layout Components

- Main layout with navigation sidebar
- Header with user menu and tenant switcher
- Empty state component for lists
- Error boundary component

48. Implement Authentication Pages

- Login page (/login)
- Registration page (/register)
- Forgot password page (/forgot-password)
- Reset password page (/reset-password)

PHASE 7: CALENDAR & TIME TRACKING UI

49. Install Calendar Library

- Choose and install calendar library (FullCalendar, React Big Calendar, or custom)
- Configure calendar with weekly view as default
- Set up calendar styling with Tailwind

50. Create Calendar Page Component

- Implement main calendar view at /dashboard/calendar

- Display week view with days as columns
- Show existing time entries as blocks on calendar

51. Implement Click-to-Add Time Entry

- Add click handler for empty calendar slots
- Open quick-entry modal on click
- Pre-fill date/time based on clicked slot

52. Create Time Entry Modal Component

- Build modal form for creating/editing time entries
- Include fields: project, task, date, start time, end time, description
- Add project/task dropdowns with data from API
- Implement form validation

53. Implement Time Entry Block Rendering

- Display time entries as colored blocks on calendar
- Show project name, task name, and duration on blocks
- Color-code blocks by project or client
- Show hover state with full details

54. Add Drag-and-Drop Functionality

- Make time entry blocks draggable
- Update entry date when dropped on different day
- Show visual feedback during drag
- Call API to update entry on drop

55. Implement Resize Functionality

- Add resize handle to bottom of time entry blocks
- Update duration when block is resized
- Show visual feedback during resize
- Call API to update entry on resize complete

56. Create Command Palette Component

- Implement keyboard shortcut (Cmd/Ctrl + K) to open palette
- Add search/filter functionality for quick entry
- Parse natural language input (e.g., "Dev 2h")
- Create time entry from palette input

57. Add Calendar View Switching

- Implement day view
- Implement week view (default)

- Implement month view
- Add navigation buttons (previous/next, today)

58. Implement Calendar Data Fetching

- Fetch time entries for current view date range
- Fetch projects and tasks for dropdowns
- Implement loading states
- Add error handling and retry

PHASE 8: ENTITY MANAGEMENT UI (ADMIN)

59. Create Clients Management Page

- Build list view at /dashboard/clients
- Display clients in table with search/filter
- Add "New Client" button
- Show edit/delete actions for each client

60. Create Client Form Component

- Build form for creating/editing clients
- Include validation
- Handle submission and API calls
- Show success/error messages

61. Create Projects Management Page

- Build list view at /dashboard/projects
- Display projects with associated client
- Add filtering by client
- Show billable status indicator

62. Create Project Form Component

- Build form for creating/editing projects
- Include client selection dropdown
- Add billable/non-billable toggle
- Handle submission and API calls

63. Create Tasks Management Page

- Build list view at /dashboard/tasks
- Display tasks with associated project
- Add filtering by project

- Show edit/delete actions

64. Create Task Form Component

- Build form for creating/editing tasks
- Include project selection dropdown
- Handle submission and API calls
- Show success/error messages

65. Create Users Management Page (Admin Only)

- Build list view at /dashboard/users
- Display users with roles
- Show invite button
- Show edit/deactivate actions

66. Create User Invitation Modal

- Build form for inviting new users
- Input email and role selection
- Send invitation via API
- Show confirmation message

PHASE 9: REPORTING & ANALYTICS

67. Create Dashboard Page

- Build main dashboard at /dashboard
- Add summary cards (total hours, billable hours, etc.)
- Display charts for time breakdown
- Show recent time entries

68. Install Chart Library

- Choose and install chart library (Recharts, Chart.js, etc.)
- Configure chart defaults and styling

69. Implement Hours Breakdown Chart

- Create pie chart for billable vs. non-billable hours
- Add time period selector (week, month, year)
- Fetch aggregated data from API
- Add chart interactions (hover, click)

70. Implement Time by Project Chart

- Create bar chart showing hours per project
- Add filtering by date range
- Sort projects by total hours
- Show top N projects with "Other" category

71. Implement Time by Client Chart

- Create chart showing hours per client
- Add filtering by date range
- Include billable vs. non-billable breakdown

72. Create Reports Page

- Build reports view at /dashboard/reports
- Add filters for date range, client, project, user
- Show summary statistics
- Display detailed time entries table

73. Create Report Generation API Endpoint

- Implement GET /api/reports/summary endpoint
- Accept query parameters for filtering
- Return aggregated data (hours by project, client, user)
- Include billable/non-billable breakdown

74. Implement CSV Export Functionality

- Add "Export to CSV" button on reports page
- Create POST /api/reports/export endpoint
- Generate CSV with filtered time entries
- Return CSV file for download

75. Implement PDF Export Functionality

- Install PDF generation library (React-PDF, PDFKit, etc.)
- Create PDF template for reports
- Add "Export to PDF" button
- Generate formatted PDF with charts and tables
- Return PDF file for download

PHASE 10: EMAIL FUNCTIONALITY

76. Set Up Email Service Configuration

- Choose email service (SendGrid, AWS SES, or Gmail SMTP)

- Configure email credentials in environment variables
- Create email service wrapper module

77. Create Email Templates

- Design HTML email template for invitation emails
- Create template for password reset emails
- Create template for weekly hour reports
- Ensure templates are mobile-responsive

78. Implement Send Invitation Email Function

- Create function to send user invitation emails
- Include invitation link with token
- Handle email sending errors gracefully

79. Implement Send Report Email Function

- Create function to send hour reports via email
- Attach PDF or include HTML report in email body
- Support multiple recipients

80. Create Manual Email Report Endpoint

- Implement POST /api/reports/send endpoint
- Accept recipient emails and report filters
- Generate and send report immediately
- Return success/error status

81. Set Up Cloud Scheduler for Automated Reports

- Create Cloud Scheduler job in GCP
- Configure weekly schedule (e.g., Friday afternoon)
- Set up job to call automated report endpoint
- Add authentication for scheduled endpoint

82. Create Automated Report Generation Endpoint

- Implement POST /api/cron/weekly-report endpoint
- Generate reports for all tenants
- Filter for missing hours or incomplete data
- Send emails to admins
- Add authentication via secret header

PHASE 11: MCP (MODEL CONTEXT PROTOCOL) INTEGRATION

83. Review MCP Specification

- Study Model Context Protocol documentation
- Understand required endpoint structure
- Document expected request/response formats

84. Create MCP Server Endpoint

- Implement POST /api/mcp endpoint
- Set up request routing based on MCP method
- Add authentication for MCP requests

85. Implement query_projects MCP Method

- Accept project query parameters
- Return list of projects for authenticated user's tenant
- Format response according to MCP spec

86. Implement log_time_entry MCP Method

- Accept time entry data from AI agent
- Parse natural language input if needed
- Validate and create time entry
- Return confirmation with entry details

87. Implement get_user_status MCP Method

- Return current user information
- Include summary of recent time entries
- Show current week's total hours

88. Add Additional MCP Methods

- Implement query_clients method
- Implement query_tasks method
- Implement get_time_entries method
- Implement update_time_entry method
- Implement delete_time_entry method

89. Create MCP Testing Tool

- Build simple testing interface for MCP endpoints
- Allow manual testing of MCP methods
- Display request/response for debugging

90. Document MCP Integration

- Write integration guide for AI agents
- Document available methods and parameters
- Provide example requests and responses
- Create setup instructions for Claude Desktop

PHASE 12: PERFORMANCE OPTIMIZATION

91. Implement Redis Caching Layer

- Set up Redis connection in application
- Create cache wrapper with get/set/delete methods
- Add TTL configuration for different data types

92. Add Caching to Frequently Accessed Data

- Cache project lists per tenant
- Cache client lists per tenant
- Cache user session data
- Set appropriate cache invalidation triggers

93. Implement Database Query Optimization

- Add missing indexes based on query patterns
- Optimize N+1 queries with proper joins
- Use database query explain plans to identify bottlenecks

94. Add API Response Compression

- Configure gzip compression for API responses
- Set appropriate compression thresholds

95. Implement Frontend Code Splitting

- Configure Next.js for optimal code splitting
- Lazy load heavy components (charts, calendar)
- Optimize bundle size

96. Add Frontend Data Caching

- Implement SWR or React Query for API data caching
- Configure stale-while-revalidate strategy
- Add optimistic updates for better UX

PHASE 13: TESTING

97. Set Up Testing Framework

- Install Jest and React Testing Library
- Configure test environment
- Set up test database for integration tests

98. Write Unit Tests for Utilities

- Test duration calculation functions
- Test date/time formatting functions
- Test validation functions

99. Write Unit Tests for API Endpoints

- Test authentication endpoints
- Test CRUD operations for all entities
- Test error handling and validation

100. Write Integration Tests

- Test complete user flows (registration, login, create entry)
- Test tenant isolation
- Test role-based access control

101. Write Frontend Component Tests

- Test form validation
- Test modal open/close behavior
- Test calendar interactions

102. Write End-to-End Tests

- Set up Playwright or Cypress
- Write tests for critical user journeys
- Test across different browsers

PHASE 14: SECURITY HARDENING

103. Implement Rate Limiting

- Add rate limiting middleware to API endpoints
- Configure limits per endpoint type (stricter for auth)
- Return appropriate 429 status codes

104. Add Input Sanitization

- Sanitize all user inputs to prevent XSS
- Validate and sanitize file uploads if added
- Use parameterized queries to prevent SQL injection

105. Implement CSRF Protection

- Add CSRF tokens to forms
- Validate CSRF tokens on state-changing requests

106. Add Security Headers

- Configure Content Security Policy (CSP)
- Add X-Frame-Options header
- Add X-Content-Type-Options header
- Configure CORS properly

107. Set Up Secrets Management

- Use Google Secret Manager for sensitive configuration
- Rotate API keys and database passwords
- Document secrets management process

108. Implement Audit Logging

- Log all authentication attempts
- Log data modification operations (create, update, delete)
- Include user, tenant, timestamp, and action in logs
- Store logs securely

PHASE 15: DEPLOYMENT & DEVOPS

109. Create Production Environment Variables

- Document all required environment variables
- Create separate configs for dev, staging, production
- Set up secrets in GCP Secret Manager

110. Build Docker Images

- Create optimized Dockerfile for Next.js frontend
- Create optimized Dockerfile for NestJS backend
- Use multi-stage builds to minimize image size
- Configure health check endpoints

111. Set Up Container Registry

- Push Docker images to Google Container Registry
- Tag images appropriately (version, environment)
- Set up automatic image scanning

112. Configure Cloud Run Services

- Deploy frontend container to Cloud Run
- Deploy backend container to Cloud Run
- Configure environment variables
- Set up custom domains

113. Set Up Database VM

- Launch Compute Engine e2-micro instance
- Install Docker and Docker Compose
- Deploy PostgreSQL and Redis containers
- Configure automatic backups

114. Configure Database Backups

- Set up daily automated backups to Cloud Storage
- Test backup restoration process
- Configure backup retention policy

115. Set Up CI/CD Pipeline

- Configure GitHub Actions or Cloud Build
- Add automated testing step
- Add Docker build and push step
- Add deployment step for passing builds

116. Configure Monitoring and Alerting

- Set up Google Cloud Monitoring
- Configure alerts for high error rates
- Monitor API response times
- Set up uptime checks

117. Implement Logging Strategy

- Configure structured logging

- Send logs to Google Cloud Logging
- Set up log retention and analysis

118. Create Deployment Documentation

- Document deployment process
- Create runbook for common issues
- Document rollback procedure

PHASE 16: POLISH & FINAL TOUCHES

119. Implement Loading States

- Add skeleton loaders for data fetching
- Show progress indicators for long operations
- Improve perceived performance

120. Add Error Boundaries and Error Pages

- Create custom 404 page
- Create custom 500 page
- Add error boundaries for component failures

121. Improve Form UX

- Add inline validation with immediate feedback
- Show success messages after operations
- Implement auto-save for drafts if applicable

122. Add Keyboard Shortcuts

- Document all keyboard shortcuts
- Add keyboard navigation for calendar
- Implement shortcuts for common actions

123. Implement Mobile Responsive Design

- Test and fix mobile layouts
- Optimize calendar for touch interactions
- Ensure forms work well on mobile

124. Add Data Export Options

- Allow users to export their own data
- Implement GDPR-compliant data export

125. Create Help Documentation

- Write user guide for all features
- Create video tutorials if applicable
- Add contextual help tooltips in UI

126. Implement User Preferences

- Allow users to set default view (day/week/month)
- Add time format preference (12h/24h)
- Save preferences to database

127. Add Accessibility Features

- Ensure WCAG 2.1 AA compliance
- Test with screen readers
- Add proper ARIA labels
- Ensure keyboard navigation works throughout

128. Performance Testing and Optimization

- Load test API endpoints
- Test with realistic data volumes
- Optimize slow queries
- Profile frontend performance

129. Create Admin Dashboard

- Build tenant overview page for super admins
- Show system health metrics
- Display usage statistics

130. Final Security Audit

- Review all authentication flows
- Test authorization on all endpoints
- Verify tenant isolation is watertight
- Check for common vulnerabilities (OWASP Top 10)

Implementation Notes

Key Principles

1. **Tenant Isolation**: Every database query must filter by tenant_id
2. **Soft Deletes**: Use deleted_at timestamps instead of hard deletes

3. **Authorization**: Check user role before every admin operation
4. **Input Validation**: Validate all inputs on both client and server
5. **Error Handling**: Return clear, actionable error messages
6. **Performance**: Cache aggressively, index properly
7. **Security**: Treat all user input as untrusted

Testing Strategy

- Write tests alongside implementation, not after
- Aim for 80%+ code coverage
- Focus integration tests on critical paths
- Test tenant isolation thoroughly

Git Workflow

- Use feature branches for each step or group of steps
- Write clear commit messages referencing step numbers
- Require code review before merging to main
- Keep main branch deployable at all times

Documentation Requirements

- Document all API endpoints with request/response examples
- Keep README updated with setup instructions
- Document environment variables
- Create architecture diagrams
- Maintain changelog

Cost Optimization Checklist

- Verify Cloud Run scales to zero
- Confirm e2-micro instance is sufficient
- Monitor egress costs
- Use Cloud Storage lifecycle policies
- Review costs weekly during development

Success Criteria

The implementation is complete when:

1. ✅ All 130 implementation steps are executed
2. ✅ All tests pass with >80% coverage
3. ✅ Application deployed and accessible via custom domain
4. ✅ Users can register, login, and manage time entries
5. ✅ Admins can manage clients, projects, tasks, and users

6. ✅ Calendar view supports drag-drop and resize
7. ✅ Reports can be generated and exported (PDF/CSV)
8. ✅ Automated weekly emails are sent via Cloud Scheduler
9. ✅ MCP integration works with Claude Desktop
10. ✅ Application runs within GCP free tier for <100 users
11. ✅ API response times <200ms
12. ✅ Tenant isolation verified through security testing
13. ✅ All documentation is complete and up-to-date

Document Version: 1.0
Last Updated: Generated from PRD
Total Implementation Steps: 130