Hour tracking application

The Goal

Be able to efficiently keep track of worked hours on projects
Inserting hours should be fast and easy
To be able to review the hours and send them to accounting
Analyse the data

Features:

1. Multitenant application, with user management.
2. Create account for a company
3. Roles: admin, user
4. Add People, clients>projects> tasks
5. Add, edit, delete(soft delete?) time entries on tasks for a timeframe(day)
6. Calendar view, timeframes day, week, month (year)
7. View entries by Client, Project, People in a time frame
8. View hour report
9. Send hour report to emails (manual, scheduled)
10. Access the data via API/MCP server (Add, delete, query, add clients, people, projects, tasks)

This Product Requirements Document (PRD) outlines the specifications for the **Hour Tracking Application**, tailored for a cost-optimized deployment on **Google Cloud Platform (GCP)**.

Product Requirements Document (PRD)

Project Name: Hour Tracker SaaS

Target Infrastructure: Google Cloud Platform (Cost-Optimized)

Version: 1.0

Status: Draft

---

1. Executive Summary

The goal is to build a multitenant, B2B time-tracking SaaS that minimizes friction for users entering hours and maximizes trust for accountants reviewing them. The application must be "Fast and Easy" to use, support AI integration via MCP, and run on a near-zero cost infrastructure using Google Cloud's free tier capabilities.

2. User Personas

- **The Consultant (User):** Needs to log hours quickly without navigating complex forms. Wants to see "how much I worked this week" instantly.
- **The Manager (Admin):** Needs to onboard new employees, define billable projects, and ensure all hours are logged by Friday.
- **The Accountant (Admin):** Needs accurate, immutable data exports (PDF/CSV) to generate invoices.
- **The AI Agent (MCP Client):** An LLM (e.g., Claude) acting on behalf of a user to log time via natural language.

---

3. Functional Requirements

3.1 Authentication & Tenancy

- **Multitenancy:** Strict data isolation using `tenant_id` on all tables.
- **Auth:** Secure login (Email/Password or OAuth).
- **Roles:**
    - `Admin`: Full access to Settings, Billing, Users, and all Time Entries.
    - `User`: Read/Write access only to their own Time Entries. Read-only access to assigned Projects.

3.2 Core Entities Management (Admin Only)

- **Clients:** Create, Update, Soft Delete clients (e.g., "Google").
- **Projects:** Linked to Clients (e.g., "Google Maps Revamp").
- **Tasks:** Standardized activities linked to Projects (e.g., "Backend Dev", "Meeting").
- **People:** Invite users via email; assign roles.

3.3 Time Tracking (The "Fast" Experience)

- **Calendar View:** A weekly grid view.
    - **Click-to-Add:** Click an empty slot to open a quick-entry modal.
    - **Drag-and-Drop:** Move time blocks between days.
    - **Resize:** Drag bottom handle to extend duration.
- **Command Palette:** Keyboard-first entry (e.g., `cmd+k` -> type "Dev 2h").
- **Soft Deletes:** Deleting an entry marks it as `deleted_at` rather than removing the row.

3.4 Reporting & Analysis

- **Dashboard:** Visual breakdown of hours (Billable vs. Non-billable).
- **Exports:** Generate PDF/CSV reports filtered by Client/Project/User for a specific date range.
- **Scheduled Emails:** Automated weekly summary sent to Admins (e.g., "Missing Hours Report").

3.5 API & MCP Integration

- **Public API:** RESTful endpoints for all core resources.
- **MCP Server:** A dedicated endpoint adhering to the Model Context Protocol to allow AI agents to:
    - `query_projects()`
    - `log_time_entry()`
    - `get_user_status()`

---

4. Technical Architecture (Google Cloud Cost-Optimized)

To meet the "cheap as possible" constraint, we will leverage GCP's Always Free tier and scaling-to-zero capabilities.

4.1 Infrastructure Diagram

1. **Frontend & API:** Google Cloud Run (Serverless Container).
2. **Database:** Google Compute Engine (`e2-micro` instance).
3. **Storage:** Local Disk on VM (backed up to Cloud Storage).
4. **Networking:** Cloud Load Balancing (optional, or direct mapping).

4.2 Component Stack

Component	Technology	GCP Service	Configuration
Frontend	Next.js (App Router)	**Cloud Run**	Min instances: 0. Memory: 512MB.
Backend	Node.js (NestJS)	**Cloud Run**	Deployed as a monorepo with Frontend or separate service.

Database	PostgreSQL 16	**Compute Engine**	e2-micro (2 vCPU, 1GB RAM). **Dockerized Postgres.**
Caching	Redis	**Compute Engine**	Run as a Docker container alongside Postgres on the same VM.
Cron Jobs	Cloud Scheduler	**Cloud Scheduler**	Triggers API endpoints for email reports.

4.3 Database Schema (Key Tables)

- tenants (id, name, plan, created_at)
- users (id, tenant_id, email, password_hash, role)
- clients (id, tenant_id, name)
- projects (id, tenant_id, client_id, name, is_billable)
- time_entries (id, tenant_id, user_id, project_id, start_time, end_time, duration, description, deleted_at)

---

5. Non-Functional Requirements

- **Cost Efficiency:** Architecture must fit within the GCP Free Tier for low volume (approx. <$1/month).
- **Performance:** API response time <200ms.
- **Scalability:** Validated support for up to 10 tenants (approx. 100 users) on the e2-micro database before vertical scaling is required.
- **Security:**
    - All data in transit encrypted via TLS.
    - Application-level Row Level Security (RLS) enforcement.

---

6. Development Roadmap

Phase 1: The Foundation (Weeks 1-2)

- Setup GCP Project and e2-micro VM with Docker/Postgres.
- Implement Next.js repo with Tailwind CSS.
- Build Auth system (NextAuth.js) and Tenant isolation middleware.

- Create CRUD APIs for Clients/Projects.

Phase 2: The Tracker (Weeks 3-4)

- Build the Calendar UI (React Big Calendar or FullCalendar).
- Implement Drag-and-Drop logic.
- Connect API to Database for creating/editing time entries.

Phase 3: The Intelligence (Week 5)

- Implement MCP Server endpoints (`/api/mcp`).
- Test integration with Claude Desktop/LLMs.

Phase 4: Reports & Polish (Week 6)

- Build the Dashboard charts (Recharts).
- Implement PDF generation (React-PDF).
- Setup Cloud Scheduler for weekly emails.

---

7. Next Step

Would you like me to generate the **Terraform script** to automatically provision this specific Google Cloud infrastructure (Cloud Run + e2-micro VM), or would you prefer the **SQL initialization script** for the database?