Project "Iron Will" - Core Platform PRD (v1.1)

1. Executive Summary & Vision

Iron Will is a high-stakes, "Ruthless Accountability" web application designed to bridge the gap between intent and action. Unlike passive habit trackers that rely on user honesty, Iron Will functions as a strict, impartial adjudicator. It uses a **Hybrid Microservices Architecture**—combining the transactional stability of **Java Spring Boot** with the cognitive capabilities of **Python FastAPI**—to create a gamified system where an AI Agent validates proof of completion with zero bias.

This document serves as the **Master Specification** for the **System of Record (Spring Boot)**, the **Frontend Experience (Next.js)**, and the underlying **GCP Infrastructure**. It defines the "body" and "nervous system" of the application, treating the AI Agent as a specialized external organ. The primary goal of this phase is to establish a secure, timezone-aware, and scalable foundation that can support the high-concurrency demands of synchronous audit loops.

2. Technical Architecture Overview

The system is designed for **data integrity** and **auditability**. We use a clear separation of concerns:

- **Frontend Layer: Next.js (React) + Tailwind CSS**
  - **Hosting:** Google Cloud Run (Frontend container) or Vercel.
  - **Role:** Handles user interaction, image compression before upload, and real-time feedback display. It is a "dumb" client that trusts the backend for all logic.
  - **State Management:** React Query (TanStack Query) for handling server state and polling.
- **Backend Layer A (The Core): Java Spring Boot 3.x (JDK 17/21)**
  - **Hosting:** Google Cloud Run (Auto-scaling stateless container).
  - **Role:** The "Source of Truth." It manages authentication, the global state machine, scheduling, and transaction logs. It enforces the rules of the game (locking, scoring).
  - **Security:** Spring Security 6.0 with OAuth2 Client.
- **Backend Layer B (The Brain): Python FastAPI** *(Out of scope for this specific document, but noted for context)*
  - **Role:** Purely functional. It receives data, reasons about it, and returns a verdict. It is stateless.
- **Database: Google Cloud SQL (PostgreSQL 15)**
  - **Configuration:** High availability not required for MVP, but daily automated backups are mandatory.

    - **Extensions:** pg_trgm for text search (future proofing).
- **Storage: Google Cloud Storage (GCS)**
    - **Structure:** Organized by user_id/goal_id/ to ensure logical data segregation.
    - **Lifecycle:** Object Lifecycle Management policies to delete images older than 60 days to manage costs.
- **Communication Pattern:** Synchronous REST API.
    - **Reasoning:** While asynchronous flows (WebSockets) are more performant, a synchronous blocking call (Frontend -> Java -> Python) simplifies the MVP architecture by removing the need for a separate event bus or socket server. The UI will simply display a "Judgement in Progress" state.

3. Functional Modules

Module A: Identity & Profile Management

- **Authentication:**
    - Exclusively **Google OAuth2** via Spring Security. No password management on our side.
    - Token handling: Session-based or JWT (stateless), secured via HTTP-only cookies.
- **Timezone Synchronization (Critical):**
    - **Problem:** "11 PM" is relative. A user in Tokyo fails hours before a user in New York.
    - **Solution:** The Frontend **must** detect the browser's timezone (e.g., Asia/Kolkata, America/New_York) on every login and PUT it to the /api/user/timezone endpoint.
    - **Logic:** The Scheduler relies entirely on this field. If a user travels, the deadline shifts to their *new* local time effectively immediately upon their next login.
- **The Accountability Score:**
    - A floating-point value stored with high precision (DECIMAL(4,2)).
    - **Visuals:** Displayed as a "Health Bar" in the HUD. Green (7-10), Yellow (3-7), Red (0-3).

Module B: The Contract (Goal Management)

- **Goal Definition:**
    - Users do not create "habits"; they sign **contracts**.
    - **Attributes:**
        - Title: Distinct name.
        - Review Time: The daily deadline. Stored as **UTC** in the DB, but presented to the user in their Local Time.
        - Frequency: Initially strict "DAILY". Future support for "WEEKDAYS".
        - Criteria Config (JSONB): This is the prompt instruction for the AI.
            - *Example:* {"type": "ocr_match", "target": "Sleep Score", "operator": ">=", "value": 85}.
- **State Machine:**
    - ACTIVE: The contract is live. Proofs can be uploaded.

    - LOCKED: The "Penalty Box." Triggered when the Global Score drops below threshold. User is strictly blocked from interacting.
    - ARCHIVED: Voluntarily retired goals.
- **Hard Core Logic:**
    - The "Lock" is binary. You are either safe, or you are locked out. There is no middle ground.
    - Lockout applies to **ALL** active goals, not just the one you failed.

Module C: Audit Submission (The Loop)

- **The Workflow:**
    1. **User Action:** Uploads a screenshot via the Dashboard.
    2. **Frontend:** Validates file type (JPG/PNG) and size (<5MB).
    3. **Spring Boot:**
        - Checks User.Score > 3.0 and Goal.Status == ACTIVE.
        - Generates a unique filename: users/{uid}/{goal_id}/{date}_{hash}.jpg.
        - Uploads to GCS Bucket.
    4. **Spring Boot:** Constructs the payload for the AI Agent and blocks waiting for response (Read Timeout: 30s).
    5. **Spring Boot:** Receives verdict.
        - If PASS: Commit AuditLog as VERIFIED. Increment Score.
        - If FAIL: Commit AuditLog as REJECTED. Decrement Score.
    6. **Response:** Returns the structured result to the frontend.
- **Latency Handling:** The UI must implement a robust "Analyzing" skeleton screen or spinner that explicitly tells the user "The Agent is Reviewing your Proof..." to explain the 5-10s delay.

Module D: Scoring & Consequences

- **The Economy of Will:**
    - **Pass (+0.5):** Reward for consistency.
    - **Fail (-0.2):** Small penalty for trying but failing criteria.
    - **Missed Deadline (-1.0):** Massive penalty for ghosting the system. This is the "Ruthless" part.
- **The Threshold (Kill Switch):**
    - **Trigger:** Any score update (via Audit or Scheduler) checks: if (newScore < 3.0).
    - **Effect:**
        - Iterate all goals for user_id where status is ACTIVE.
        - Set status to LOCKED.
        - Set locked_until timestamp to NOW() + 24 Hours.
    - **Redemption:** After 24 hours, a scheduled job (or login check) unlocks the goals, but the score remains low, requiring immediate perfection to avoid re-locking.

Module E: Notification System

- **The "Nag" Engine:**
  - A background thread (Spring Scheduler) runs every 15 minutes (0 0/15 * * * ?).
  - **Query Logic:**
    - Select users where (Current_UTC_Time > Goal_Review_Time_UTC) AND (No AuditLog exists for Today).
    - *Refinement:* Ensure we don't nag users who are sleeping (e.g., don't nag at 3 AM local time).
  - **Action:** Create a row in the notifications table.
- **Delivery:**
  - The Frontend uses a useEffect hook to poll GET /api/notifications/unread every 60 seconds.
  - Visual: A red badge on the bell icon and a Toast popup.

4. Database Schema (PostgreSQL)

Refined schema with indexes and audit timestamps.

```
-- 1. USERS TABLE
-- Stores profile and global state
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    full_name VARCHAR(100),
    timezone VARCHAR(50) NOT NULL, -- Critical for scheduling
    accountability_score DECIMAL(4,2) DEFAULT 5.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Index for scheduler performance
CREATE INDEX idx_users_timezone ON users(timezone);



-- 2. GOALS TABLE
-- The Configuration / Contract
CREATE TABLE goals (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    review_time TIME NOT NULL, -- Stored as UTC. Converted to local in app logic.
    frequency_type VARCHAR(20) DEFAULT 'DAILY',
    criteria_config JSONB NOT NULL, -- The "Prompt" for the Agent
    status VARCHAR(20) DEFAULT 'ACTIVE', -- ACTIVE, LOCKED, ARCHIVED
```

```
    locked_until TIMESTAMP, -- Null unless in penalty box
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_goals_user_status ON goals(user_id, status);


-- 3. AUDIT LOGS TABLE
-- The Ledger of Truth
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY,
    goal_id UUID REFERENCES goals(id),
    audit_date DATE NOT NULL,
    proof_url TEXT,
    status VARCHAR(20) DEFAULT 'PENDING', -- PENDING, VERIFIED, REJECTED, MISSED
    agent_remarks TEXT, -- "I see you only slept 4 hours..."
    score_impact DECIMAL(4,2), -- Snapshot of points lost/gained
    submitted_at TIMESTAMP,
    UNIQUE(goal_id, audit_date) -- Constraint: One proof per day per goal
);

CREATE INDEX idx_audit_logs_date ON audit_logs(audit_date);


-- 4. NOTIFICATIONS TABLE
-- In-app messaging queue
CREATE TABLE notifications (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    message TEXT NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_notifications_user_unread ON notifications(user_id, is_read);
```

5. API Contract (Java -> Python Agent)

This defines the interface between the Core and the Brain.

Endpoint: POST /internal/agent/audit

Headers:

- X-Internal-Secret: Shared secret for basic service-to-service auth.

Request Payload:

```
{
  "request_id": "uuid-trace-id",
  "user_id": "uuid",
  "goal_context": {
    "title": "Deep Sleep Protocol",
    "description": "Ensure sleep score > 85",
    "criteria": {
        "type": "OCR",
        "keywords": ["Sleep Score", "Deep Sleep"]
    }
  },
  "proof_url": "gs://iron-will-proofs/users/123/goals/456/proof.jpg",
  "timezone": "Asia/Kolkata",
  "current_time_local": "2023-10-27T09:15:00"
}
```

Response Payload:

```
{
  "verdict": "PASS", // Or "FAIL"
  "remarks": "Excellent work. You achieved a score of 89, surpassing the target of 85.",
  "extracted_data": {
      "sleep_score": 89,
      "bed_time": "22:30"
  },
  "score_impact": 0.5
}
```

Error Handling:

- 500 Internal Server Error: Java assumes "Technical Difficulites", allows retry, does not penalize user.
- 400 Bad Request: Java rejects the proof immediately.

6. Implementation Task List

Phase 1: Infrastructure & Setup

- [ ] **Spring Boot Init:**
  - Setup Maven project with Dependencies: spring-boot-starter-web, spring-boot-starter-data-jpa, spring-boot-starter-security, spring-boot-starter-oauth2-client, spring-cloud-gcp-starter-storage, lombok.
  - Configure application.yml for multiple profiles (dev, prod).
- [ ] **Next.js Init:**
  - Scaffold with create-next-app.
  - Configure Tailwind tailwind.config.js.
  - Setup axios interceptors for API calls.
- [ ] **GCP Provisioning:**
  - Create Cloud SQL Instance (Postgres 15).
  - Create GCS Bucket iron-will-proofs with CORS configuration for direct browser access (if needed) or service-account access.
  - Create Service Account iron-will-backend-sa with Storage Admin and Cloud SQL Client roles.

Phase 2: Core Backend (Spring Boot)

- [ ] **Auth & Identity:**
  - Implement SecurityFilterChain allowing public access to /auth and securing /api/**.
  - Create CustomOAuth2UserService to map Google User info to our User entity.
  - Implement TimezoneController to handle client updates.
- [ ] **Domain Logic:**
  - Create Entities (User, Goal, AuditLog) with JPA annotations.
  - Implement GoalService: Logic for creating goals and validating timestamps.
  - Implement ScoreService: The logic for math (+0.5, -0.2) and triggering LockoutService.
  - Implement LockoutService: Logic to flip status to LOCKED and calculate timestamps.

Phase 3: The Audit Endpoint & GCS

- [ ] **GCS Integration:**
  - Create StorageService to handle MultipartFile upload.
  - Implement filename hashing strategy.
- [ ] **Agent Client:**
  - Use WebClient or RestClient to create a typed interface for the Python API.
  - Implement timeout handling (default 30s).
- [ ] **Orchestration Controller:**
  - Create POST /api/goals/{id}/audit.
  - Wire together: Auth Check -> Lock Check -> Upload -> AI Call -> DB Save -> Score Update.

Phase 4: Frontend Development

- [ ] **Auth Flow:** Login page with "Sign in with Google" button. Redirect logic.
- [ ] **Dashboard (HUD):**
  - Component: ScoreBar (Dynamic width/color based on score).
  - Component: GoalList (Cards showing status and "Upload" button).
- [ ] **Upload Experience:**
  - Modal with Dropzone.
  - State: Idle -> Uploading -> Analyzing (Spinner) -> Result (Success/Fail Animation).
- [ ] **Notification UI:**
  - Simple polling hook (useInterval) to fetch unread count.

Phase 5: The "Nag" Scheduler

- [ ] **Job Logic:**
  - Implement @Scheduled(cron = "...") method.
  - Write JPQL query to find users who missed deadlines in their specific timezone.
- [ ] **Notification Generation:**
  - Batch insert into notifications table.