Product Requirements Document (PRD) — Backend

Project: Backend for Mental-Wellbeing / Insights Web App

Owner: Diwakar Patel / Backend Team

Created: 2026-02-25

Purpose: This PRD defines requirements for a production-ready backend built with **Python 3.11+** + **FastAPI** designed to serve a mobile-first frontend (TypeScript + React + Vite). The backend secure user management, timed check-ins, internal AI inference, alerting, analytics, and admin functionality. It is optimized for developer velocity, observability, and privacy-sensitive contexts (e. healthcare/advisory prototypes).

---

1. Executive Summary

Build an asynchronous, scalable, and maintainable backend that: - Accepts user check-ins and raw inputs. - Runs internal AI pipelines to generate insights. - Produces alerts and sched summaries. - Exposes a versioned REST API consumed by the frontend. - Follows security bes for sensitive data.

Design tradeoffs prioritize simplicity for MVP (FastAPI background tasks) with a clear path to (Celery + Redis, separate AI service, Kubernetes).

---

2. Objectives & Success Metrics

Objectives

1. Reliable API with clear contracts for frontend integration.
2. Privacy-first handling of user data and AI outputs.
3. Scalable AI pipeline that can be run asynchronously and monitored.
4. Fast developer iteration with typed models and tests.

Success Metrics

- API uptime ≥ 99.9% (SLA target for production).
- Average API latency (p50) for read requests < 80ms.
- Background job success rate ≥ 99%.
- Mean time to detect < 5 minutes (Sentry alerts).
- Test coverage: 80% for core modules.

- Time from check-in to AI summary (MVP): < 30 s; with background processing acceptable up to 2 minutes.

---

3. User Personas

1. **End User (Student/Consumer)** — submits daily check-ins, reads insights and recommendations. Mobile-first.
2. **Coach/Teacher** — views aggregated trends for assigned users (requires privacy controls and consent).
3. **Admin / Ops** — manages users, monitors system health, views logs and audit trails.
4. **Data Scientist** — iterates on AI pipelines, needs access to raw inputs and feature logs.

---

4. Scope & Feature List

MVP (Must-have)

- REST API v1 with authentication (JWT + refresh) and endpoints:
- `POST /api/v1/auth/login`
- `POST /api/v1/auth/refresh`
- `POST /api/v1/auth/signup`
- `GET /api/v1/auth/me`
- `POST /api/v1/checkins`
- `GET /api/v1/checkins?range=30d`
- `GET /api/v1/dashboard`
- `GET /api/v1/insights/{id}`
- `GET /api/v1/admin/users`
- Internal AI inference via FastAPI background tasks (MVP) that compute `ai_analysis_results` and store them.
- PostgreSQL database with async SQLAlchemy 2.0 + Alembic migrations.
- Basic scheduler (APScheduler) for daily/weekly summary generation.
- Role-based access control (user, coach, admin).
- Logging (structured) and error tracking (Sentry).
- Dockerized service with Uvicorn ASGI server.

Post-MVP (Scale / Nice-to-have)

- Move heavy AI inference to Celery + Redis workers (GPU-enabled if required).
- Separate AI service (microservice) with gRPC/HTTP contract.
- Rate limiting, API quotas, and request throttling.
- Audit logs and export features for compliance.
- Full admin console and analytics dashboards.

---

5 . API Design & Contract Guidelines

- Base path: `/api/v1/`
- JSON RESTful responses with envelope:

```
{ "ok": true, "data": {...}, "error": null }
```

- Standard error format:

```
{ "ok": false, "error": {"code": "invalid_input", "message": "...",
"details": {...}}}
```

- Pagination: cursor-based for lists.
- Time fields always in UTC ISO `8 6 0 1` .

Example endpoints (brief)

- `POST /api/v1/checkins`
- Body: `{ user_id?, mood:int(1-10), sleep_hours:float, notes:string, timestamp?:ISO }`
- Response: immediate ack with `analysis_id` (if background) and `status` .
- `GET /api/v1/dashboard?range=30d`
- Response: aggregated metrics + recent insights.

The detailed OpenAPI spec should be generated from Pydantic models and included in the (FastAPI auto-docs).

---

6 . Authentication & Authorization

Strategy

- JWTs issued on login (short-lived access token) + long-lived refresh token.
- Tokens sent as HTTP-only, Secure cookies (Best practice) OR Authorization header if cookie not feasible.
- Refresh endpoint: `POST /api/v1/auth/refresh` rotates tokens.
- Logout invalidates refresh token server-side (store token identifier / jti in DB or Redis blacklist until expiry).

Libraries

- `python-jose` for JWT operations.
- `passlib[bcrypt]` for password hashing.
- FastAPI dependencies used to enforce `get_current_user` and role guards.

RBAC

- Roles: `user`, `coach`, `admin` with decorators/dependencies for route protection.

Security note: For HIPAA-like use cases, opt for server-side token revocation tracking and shor access tokens.

---

7. Database Schema (Conceptual & Example)

Primary DB: PostgreSQL. Use async SQLAlchemy `2.0` models with Alembic migrations.

Core tables (conceptual)

users - id: UUID PK - email: str (unique) - hashed_password: str - full_name: str - role: en updated_at - last_login - metadata JSONB (optional)

daily_checkins - id: UUID PK - user_id: FK -> users.id - mood: smallint - sleep_hours: nume notes: text - meta: jsonb (input modalities) - created_at: timestamp

ai_analysis_results - id: UUID PK - checkin_id: FK -> daily_checkins.id - user_id: FK - model_ve summary: text - labels: jsonb (e.g., stress_level, risk_score) - confidence: numeric - raw_features: created_at

alerts - id: UUID PK - user_id: FK - ai_result_id: FK (nullable) - type: enum (low_sleep, hig status: enum (open, acknowledged, closed) - payload: jsonb - created_at

user_settings - id: UUID PK - user_id FK - preferences jsonb (language, notifications, timezone)

audit_logs (optional but recommended) - id, actor_id, action, context jsonb, created_at

Design principles: - Separate raw inputs (daily_checkins) from derived outputs (ai_analysis_results). - JSONB for flexible feature storage and future model experimentation.

---

8. AI Integration Architecture

Philosophy

AI runs *internally* as a service or background job — **not** calling external third-party inference APIs for privacy and cost control.

MVP Flow (FastAPI Background Task)

1. User posts check-in → API stores `daily_checkins` and returns ack.
2. FastAPI background task triggers AI pipeline for that check-in.

3. Pipeline produces `ai_analysis_results` and possible `alerts`.
4. Frontend polls `GET /api/v1/insights/{id}` or receives realtime push (websocket / push) when ready.

Scale Flow (Recommended)

- Move heavy inference to Celery workers using Redis as broker & result backend.
- Workers run model code (PyTorch / Transformers) and write results to DB (or object store for large artifacts).
- Optionally expose an AI microservice with its own API and versioning.

Model Management

- Store `model_version` in results
- Keep feature logs for retraining
- Ensure deterministic pre-processing in production

Libraries

- `transformers`, `torch`, `scikit-learn`, `pandas`, `numpy` as required by models

Security: sandbox or containerize model execution to limit resource abuse.

---

9. Background Processing & Scheduling

MVP

- FastAPI `BackgroundTasks` for near-immediate small jobs (non-blocking) and APScheduler for scheduled jobs (daily summaries, trend scans).

Scale

- Celery + Redis for robust, retryable, distributed jobs.
- Use separate worker pools for CPU-bound preprocessing and optional GPU inference.

Examples

- Daily job: compute 24-hour trend summaries for users with activity.
- Weekly digest: aggregate insights and email (or WhatsApp) summaries.
- Alert scanner: run every 5–15 minutes to escalate critical alerts.

---

10. Notifications & Integrations

- In-app notifications via DB + WebSocket (or server-sent events).
- Email via transactional provider (SendGrid / Amazon SES).

• WhatsApp: use Twilio/MessageBird (if required) — treat as opt-in and log consents.
• Ensure user opt-in for notifications and record consent.

---

11. Observability & Monitoring

• Error tracking: Sentry (exceptions + performance traces).
• Structured logs: loguru or Python logging with JSON output.
• Metrics: Prometheus-compatible metrics exported (request latency, job queue length).
• Tracing: OpenTelemetry (optional) to trace across API and worker boundaries.
• Health endpoints: `/health` and `/metrics`.

---

12. Testing & QA

• Unit tests: `pytest` + `pytest-asyncio` for async endpoints.
• Integration tests: `httpx.AsyncClient` + test DB fixtures (Postgres in Docker / testcontainers).
• Mock AI models in tests or use lightweight deterministic model.
• End-to-end tests: Playwright hitting deployed preview or staging.
• CI: GitHub Actions to run lint, mypy (optional), tests, and build images.

---

13. Security & Compliance

Security Controls

• HTTPS required (TLS 1.2+)
• Passwords salted & hashed with bcrypt
• Short-lived access tokens + refresh tokens + rotation
• Rate limiting for auth endpoints
• Input validation with Pydantic v2
• Least-privilege DB roles

Compliance (guidance, not legal advice)

• If handling PHI (Protected Health Information), consult legal for HIPAA/GDPR requirements.
• Recommendations for compliance:
• Data minimization & encryption at rest
• Audit trails and access logs
• Business Associate Agreements (BAA) with cloud vendors where needed
• Data retention & deletion policy

---

14. Deployment & DevOps

- Containerize with Docker; multi-stage builds for small images.
- ASGI server: Uvicorn with Gunicorn or use Uvicorn workers (depending on env).
- Platforms: Render, Railway, or container-based deploy to AWS/GCP/Azure.
- Use managed Postgres (Supabase, Neon, RDS)
- CI/CD: GitHub Actions → build & push container → deploy to environment
- Secrets: store in provider secrets stores (Render secrets / GitHub secrets)

---

15. Roadmap & Milestones

M0 — Foundations (1–2 weeks) - Project scaffold, Dockerfile, basic auth, DB models, migrations, OpenAPI setup.

M1 — Core API (2–3 weeks) - Check-ins, AI background task MVP, dashboard endpoints, tests.

M2 — Scale & Reliability (2–4 weeks) Move to Celery workers, add Sentry, metrics, and monitoring.

M3 — Compliance & Production Hardening (2–3 weeks) Audit logging, retention policies, admin console, and performance tuning.

---

16. Risks & Mitigations

- **Risk:** AI model costs & latency.
- **Mitigation:** Use lightweight models, batch inference, and offline training.
- **Risk:** Data privacy & regulatory non-compliance.
- **Mitigation:** Keep data minimal, encrypt, and maintain audit logs.
- **Risk:** Long-running tasks causing request timeouts.
- **Mitigation:** Use background jobs & worker queues; always respond quickly to client.

---

17. Open Questions

- Will the app store any PHI-level data (medical diagnoses)? If yes, we must plan for full compliance.
- Expected scale: daily active users and check-ins/day (helps size infra).
- Need for realtime push vs polling for insights?
- Do you require message/WhatsApp integration at launch or later?

---

18. Appendix

Suggested Tech Stack (verbatim)

Backend Stack: Python 3.11, FastAPI, PostgreSQL, SQLAlchemy, JWT Authentication, Internal AI Services (Transformers + Scikit-learn), Background Tasks, Docker-based deployment.

Deliverables I can generate next (pick any)

- Exact DB schema (DDL + Alembic migrations)
- Complete OpenAPI spec (YAML/JSON)
- Sample FastAPI codebase scaffold (Docker-enabled)
- Celery + Redis worker setup for AI jobs
- Security/compliance checklist for PHI/GDPR

If you'd like, I can now generate the **exact DB schema** and **OpenAPI contract** for the endpoints I listed, or scaffold the full FastAPI project (with Docker, CI, and sample model). Which one should I create next?