Below is your **Claude-ready COMPREHENSIVE BACKEND TECH STACK LAYOUT** aligned perfectly with your React + Vite + TanStack Query frontend.

You can paste this into Claude to generate a full production backend.

🚀 BACKEND TECH STACK GUIDE (Aligned with Your Frontend)

🧠 Project Context

Build a **production-grade FastAPI backend** that supports:

- React + Vite frontend
- TanStack Query data fetching
- JWT cookie authentication
- Mobile-first India users
- AI inference pipeline
- Dashboard analytics
- Admin panel
- Future microservice scaling

1️⃣ Core Runtime

Language

Python 3.11

Requirements

- Async-first codebase
- Type hints everywhere
- No blocking DB calls
- Ruff/Black formatting

Framework

FastAPI (latest)

Must Enable

- Async endpoints
- Dependency injection
- OpenAPI auto docs
- BackgroundTasks
- Middleware support

ASGI Server

Uvicorn

Production mode:

uvicorn app.main:app --host 0.0.0.0 --port 8000

Optional production:

- Gunicorn + Uvicorn workers

2️⃣ API Architecture (Frontend-Compatible)

Base Path

/api/v1/

⚠️ Must match frontend environment variable:

VITE_API_BASE_URL

Response Envelope (MANDATORY)

Frontend TanStack Query expects consistent shape.

Success

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

Error

```
{
  "ok": false,
  "error": {
    "code": "string",
    "message": "human readable",
    "details": {}
  }
}
```

HTTP Rules

- JSON only
- UTF-8
- UTC timestamps
- snake_case in backend
- frontend may map to camelCase

3️⃣ Folder Structure (CRITICAL — Claude must follow)

```
backend/
├── app/
│   ├── core/
│   │   ├── config.py
│   │   ├── security.py
│   │   └── deps.py
│   │
│   ├── database/
│   │   ├── session.py
│   │   └── base.py
│   │
```

```
│   ├── models/
│   ├── schemas/
│   ├── services/
│   │
│   ├── auth/
│   ├── users/
│   ├── checkins/
│   ├── ai/
│   ├── alerts/
│   ├── insights/
│   │
│   ├── middleware/
│   └── main.py
│
├── alembic/
├── tests/
├── Dockerfile
├── docker-compose.yml
└── pyproject.toml
```

✅ Feature modular
✅ Scalable
✅ Industry standard

4️⃣ Database Layer

Primary DB

PostgreSQL

Hosting (MVP friendly)

- Supabase
- Neon
- Railway

ORM

SQLAlchemy 2.0 (async)

Requirements

- Async engine
- AsyncSession
- Declarative models
- Relationship loading optimized

Migrations

Alembic

Must support:

- autogenerate
- versioned migrations
- safe upgrades

Connection Pool

Use:

- asyncpg
- pool_pre_ping=True

5️⃣ Authentication (Frontend-Aligned 🔐)

Strategy (VERY IMPORTANT)

Your React app uses:

✅ HTTP-only cookies
✅ Silent refresh
✅ Protected routes

Backend MUST support this.

Token Design

Access Token

- JWT
- 15 min expiry
- HTTP-only cookie

Refresh Token

- JWT
- 7–30 days
- HTTP-only cookie
- rotation required

Required Libraries

- python-jose
- passlib[bcrypt]

Required Endpoints

POST /api/v1/auth/signup
POST /api/v1/auth/login
POST /api/v1/auth/refresh
POST /api/v1/auth/logout
GET /api/v1/auth/me

Cookie Settings (IMPORTANT)

Must set:

HttpOnly=True
Secure=True (prod)
SameSite=Lax

Frontend Compatibility Rule

TanStack Query will:

- send credentials: include
- expect 401 on expiry
- auto call refresh

Backend MUST:

- return proper 401
- not return 200 with error
- allow retry flow

6️⃣ Core Feature Modules

Users Module

Responsibilities:

- profile
- preferences
- roles

Checkins Module

Stores:

- mood
- sleep
- notes
- timestamp

Triggers AI pipeline.

AI Module

Internal service only.

MVP Flow

POST checkin
→ store DB
→ background task

→ run model
→ store results

Alerts Module

Generates:

- high stress
- low sleep
- risk signals

Insights Module

Provides dashboard data for:

- TanStack Query
- Recharts

7️⃣ Background Processing

MVP

Use:

FastAPI BackgroundTasks

For:

- AI inference
- alert creation

Scale Phase

Switch to:

Celery + Redis

When:

- jobs > 2 sec
- CPU heavy
- many users

8️⃣ Scheduler

Use:

APScheduler

Jobs:

- daily summary
- weekly insights
- alert scans

9️⃣ AI Stack

Internal only.

Libraries

- transformers
- torch
- scikit-learn
- numpy
- pandas

AI Performance Rules

- load model once (singleton)
- avoid reloading per request
- async-safe wrapper
- timeout protection

🔟 Validation Layer

Use:

Pydantic v2

Rules:

- strict mode
- response models everywhere
- no raw dict responses

1️⃣1️⃣ Logging & Monitoring

Logging

Use:

- loguru OR structlog
- JSON logs

Error Tracking

Sentry

Must capture:

- exceptions
- slow requests
- background failures

Health Endpoints

GET /health
GET /metrics

1️⃣2️⃣ Testing Stack

Required:

- pytest
- pytest-asyncio
- httpx

Coverage target:

≥ 80% core modules

1️⃣3️⃣ Security Requirements

Mandatory:

✅ bcrypt hashing
✅ JWT expiry
✅ input validation
✅ rate limit on auth
✅ CORS configured
✅ HTTPS only
✅ SQL injection safe (ORM)

1️⃣4️⃣ Docker Requirements

Backend must run with:

docker compose up --build

Container must include

- Python 3.11 slim
- non-root user
- healthcheck
- small image

1️⃣5️⃣ Performance Targets

Backend should achieve:

- p50 < 80ms
- p95 < 200ms
- async everywhere
- connection pooling
- no N+1 queries