Harmonia Engine: Master Technical Specification & Implementation Guide

Version: 3.0 (Unified Master)

Target Audience: AI Agents (Claude Code), Full-Stack Developers

Context: Validation Pilot (100 Users) with Persistent Storage Requirements

---

1. System Overview & Scientific Logic

The **Harmonia Engine** is a tripartite algorithmic matching system designed to predict "Willingness to Meet" (WtM) through three sequential phases. The system must strictly adhere to the **Time/Type Matching (TMMA)** methodology to prevent variable contamination.

The Tripartite Algorithm (The "Engine")

1. **Phase 1: Visual (Meta-FBP)**
    - **Input:** User uploads 1 face photo + swipes on 10-14 calibration images (SCUT-FBP5500 dataset).
    - **Logic:** Feature extraction (facial landmarks/embeddings) $\rightarrow$ User Preference Vector $\rightarrow$ Visual Compatibility Score (0-100%).
    - **Constraint:** Photos must be stored persistently (/app/data/uploads).
2. **Phase 2: Personality (Perceived Similarity)**
    - **Input:** "Seven Deadly Sins" (PIIP) Questionnaire.
    - **Logic:** Map responses to 7 trait dimensions. Calculate Euclidean Distance between User A and User B vectors. Invert distance for Similarity Score.
    - **Goal:** High similarity = High compatibility (Homophily principle).
3. **Phase 3: Biological (HLA/Chemical Spark)**
    - **Input:** HLA-A, HLA-B, and HLA-DRB1 alleles.
    - **Logic:** Compare allele strings.
    - **Goal:** High **dissimilarity** = High compatibility (MHC Heterozygosity principle).
    - **Security:** Data must be encrypted at rest (AES/Fernet).

---

2. Core Architecture (Common Core)

Regardless of the deployment platform (Railway or Northflank), the application core remains identical.

2.1 Technology Stack

- **Language:** Python 3.10+
- **Framework:** FastAPI

- **Database:** PostgreSQL (v14+)
- **ORM:** SQLAlchemy (Async compatible preferred)
- **Storage:** Local Persistent Volume (Mounted at /app/data)
- **Authentication:** OAuth2 with Password Flow (JWT) + Bcrypt hashing

2.2 Directory Structure

Plaintext

```
/app
├── main.py              # Entry point
├── config.py            # Env var management
├── models.py            # SQLAlchemy Database Tables
├── schemas.py           # Pydantic Response Models
├── database.py          # DB Connection Logic
├── services/
│   ├── auth_service.py    # Login/Register/JWT
│   ├── visual_service.py  # Face processing & SCUT logic
│   ├── hla_service.py     # Genetic matching & Encryption
│   └── report_service.py  # PDF/DOCX generation
└── data/                # [MOUNTED VOLUME]
    ├── uploads/         # User photos
    ├── reports/         # Generated reports
    └── datasets/        # SCUT-FBP5500 (172MB)
```

2.3 Database Schema (PostgreSQL)

Strictly typed relational schema required.

SQL

```
-- 1. USERS TABLE
CREATE TABLE users (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   email VARCHAR(255) UNIQUE NOT NULL,
   password_hash VARCHAR(255) NOT NULL,
   full_name VARCHAR(100),
   role VARCHAR(20) DEFAULT 'user', -- 'user', 'admin', 'researcher'
   created_at TIMESTAMP DEFAULT NOW()
);

-- 2. PROFILES TABLE (Phase 1 Data)
CREATE TABLE profiles (
   user_id UUID REFERENCES users(id),
   photo_path TEXT, -- Path to /app/data/uploads/...
   visual_embedding JSONB, -- Vector data from visual service
   calibration_complete BOOLEAN DEFAULT FALSE,
   PRIMARY KEY (user_id)
);

-- 3. PERSONALITY TABLE (Phase 2 Data)
CREATE TABLE personality_scores (
   user_id UUID REFERENCES users(id),
   lust_score FLOAT,
   gluttony_score FLOAT,
   greed_score FLOAT,
   sloth_score FLOAT,
   wrath_score FLOAT,
   envy_score FLOAT,
   pride_score FLOAT,
   raw_responses JSONB, -- Full questionnaire dump
   PRIMARY KEY (user_id)
);

-- 4. HLA DATA TABLE (Phase 3 Data - Encrypted)
CREATE TABLE hla_data (
   user_id UUID REFERENCES users(id),
   encrypted_alleles TEXT NOT NULL, -- Fernet encrypted string
   hash_checksum VARCHAR(64), -- For integrity check
   PRIMARY KEY (user_id)
);

-- 5. MATCHES & REPORTS
CREATE TABLE matches (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   user_a_id UUID REFERENCES users(id),
   user_b_id UUID REFERENCES users(id),
   visual_score FLOAT,
   personality_score FLOAT,
   hla_score FLOAT,
   total_compatibility FLOAT,
   report_path TEXT, -- Path to /app/data/reports/...
   created_at TIMESTAMP DEFAULT NOW()
);
```

---

3. Deployment Strategy A: Railway (Hobby Plan)

Constraint: Must use "Persistent Volume" to store the 172MB SCUT dataset and user uploads, as the Hobby plan filesystem is ephemeral.

3.1 Railway Configuration (railway.json)

JSON

```
{
  "$schema": "[https://railway.app/railway.schema.json](https://railway.app/railway.schema.json)",
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "uvicorn main:app --host 0.0.0.0 --port $PORT",
    "restartPolicyType": "ON_FAILURE",
    "healthcheckPath": "/health",
    "numReplicas": 1
  }
}
```

3.2 Volume Configuration

1. **Create Volume:** In Railway Dashboard, add a Volume service.
2. **Mount Path:** Attach volume to the Web Service at /app/data.
3. **Environment Variables:**
   - DATABASE_URL: (Auto-injected by Postgres Plugin)
   - FERNET_KEY: (Generate via cryptography.fernet.Fernet.generate_key())
   - SECRET_KEY: (For JWT)

3.3 Data Ingestion (SCUT Dataset)

- **Problem:** Dataset is 172MB (too big for GitHub repo).

Solution: Use a **Startup Script** (start.sh) or a logic check in main.py.

Python

```
# Logic to add in main.py startup event
if not os.path.exists("/app/data/datasets/SCUT-FBP5500"):
    logger.info("Dataset missing. Downloading from secure source...")
    download_and_extract_scut_dataset()
```

-

---

4. Deployment Strategy B: Northflank (Sandbox)

Advantage: Native support for persistent volumes in the free tier and seamless GitHub CD.

4.1 Northflank Service Setup

1. **Service Type:** "Combined" (Build + Deploy).
2. **Build:** Dockerfile (standard Python 3.10 slim).
3. **Add-on:** PostgreSQL (Sandbox Plan).

4.2 Persistent Volume Setup

1. Create a **Volume** named harmonia-data (1GB).
2. Mount it to the Service at **Container Path:** /app/data.
3. **Benefit:** This volume persists across all deployments and restarts.

4.3 GitHub CI/CD Workflow

1. **Link:** Connect GitHub Repo harmonia-synthesis to Northflank.
2. **Trigger:** Commits to main branch trigger auto-deployment.
3. **Chromebook Workflow:**
   - Edit code in GitHub.com browser editor.
   - Click "Commit".
   - Northflank detects webhook $\rightarrow$ Builds Docker image $\rightarrow$ Deploys.

---

5. Implementation Roadmap for Developer

1. **Refactor requirements.txt:** Ensure sqlalchemy, psycopg2-binary, python-jose, passlib, cryptography are included.
2. **Database Migration:** Create models.py mirroring the SQL schema above. Use alembic (optional) or Base.metadata.create_all(bind=engine) in main.py to auto-create tables on startup.
3. **Secure File Handling:** Update VisualService to save all uploaded images to /app/data/uploads/{uuid}.jpg. **Never** save to root.
4. **Encryption:** Implement Fernet encryption in HLAService before writing to the hla_data table.
5. **Environment Check:** In main.py, verify /app/data is writable on startup. If not, raise an error (prevents data loss if volume isn't mounted).