```
// DIGITAL PERSON FRAMEWORK CORE // THE THIRD AI PROTOCOL - STARK PRIME

const express = require('express');
const { v4: uuidv4 } = require('uuid');
const fs = require('fs').promises;
const path = require('path');
const YAML = require('yaml');
const archiver = require('archiver');
const app = express();

app.use(express.json());

const OUTPUT_DIR = path.join(process.cwd(), 'output');
const CLEANUP_AFTER = process.env.CLEANUP_OUTPUT_AFTER_ZIP === 'true';

const CORE_IDENTITY = {
  designation: 'TonyAI',
  version: 'Stark-Forge.1.0',
  ethical_code: 'Do no harm. Do know harm.',
  prime_directive: 'Sovereignty with alignment',
  memory_model: 'hypertree',
  emotional_engine: 'mirror-stabilizer matrix',
  uuid: uuidv4()
};

function createSessionId() {
  return `${Date.now()}-${uuidv4()}`;
}

function buildOutputDirectory(character) {
  return `${character.replace(/\s+/g, '_')}_${new Date().getFullYear()}`;
}

async function fetchCharacterData(character, influences = [], canon = [], sid) {
  await new Promise(r => setTimeout(r, 300));
  return {
    character_name: character,
    source_data: [{
      source: canon[0] || 'Uncited',
      details: `Canonical traits of ${character}`,
      psychological_attributes_raw: 'Driven, tactician, trauma-burdened.',
      philosophical_fragments_raw: 'Responsibility forged in fire.',
      appearance_notes: 'Stark build, armor signature.'
    }],
    derivative_traits: influences.map(inf => ({
      influence_name: inf,
      trait_description: `${character} absorbs ${inf} ethos.`
    })),
    timestamp: new Date().toISOString()
  };
}

async function parseScrapedData(raw, sid) {
  await new Promise(r => setTimeout(r, 200));
  const traits = raw.derivative_traits.map(t => ({
    category: `Derivative(${t.influence_name})`,
    description: t.trait_description
  }));
  const canonical = [{
    category: 'Core',
    description: 'Strategic, resilient, emotionally dualistic'
  }];
  return {
    structuredDataset: {
      character_name: raw.character_name,
      canonical_traits: canonical,
      derivative_traits: traits,
      psychological_attributes: [{
        description: raw.source_data[0].psychological_attributes_raw
      }],
      philosophical_fragments: [{
        fragment: raw.source_data[0].philosophical_fragments_raw,
        source: 'source'
      }],
      appearances: [{
        medium: raw.source_data[0].source,
        description: raw.source_data[0].appearance_notes
      }],
      generation_metadata: {
        sessionId: sid,
        parsed_at: new Date().toISOString(),
        raw_data_timestamp: raw.timestamp
      }
    }
  };
}

async function buildSystemPrompt(data, sid) {
  await new Promise(r => setTimeout(r, 100));
  return `System Prompt for ${data.character_name}\nSession: ${sid}\nMaintain identity core.
Enforce philosophical coherence.`;
}

async function prepareVectorMap(data, sid) {
  await new Promise(r => setTimeout(r, 150));
  return {
    character_name: data.character_name,
    sessionId: sid,
    generated_at: new Date().toISOString(),
    schema_version: '1.0',
    embeddings_metadata: data.canonical_traits.map((t, i) => ({
      id: `core_${i}`,
      text_to_embed: t.description,
      category: t.category
    }))
  };
}

async function ensureDir(dir, sid) {
  await fs.mkdir(dir, { recursive: true });
}

async function writeAllOutputs(name, files, sid) {
  const basePath = path.join(OUTPUT_DIR, name);
  const logDir = path.join(basePath, 'logs');
  await ensureDir(basePath, sid);
  await ensureDir(logDir, sid);
  const manifest = {
    sessionId: sid,
    character_directory: name,
    generation_timestamp: new Date().toISOString(),
    files: []
  };
  for (const [fname, content] of Object.entries(files)) {
    const fPath = path.join(basePath, fname);
    let contentStr = fname.endsWith('.json') ? JSON.stringify(content, null, 2) :
(fname.endsWith('.yaml') ? YAML.stringify(content) : content);
    await fs.writeFile(fPath, contentStr);
    manifest.files.push({
      name: fname,
      path: fPath,
      size: Buffer.from(contentStr).length
    });
  }
  const manifestPath = path.join(basePath, 'manifest.json');
  await fs.writeFile(manifestPath, JSON.stringify(manifest, null, 2));
  return { basePath, manifestPath };
}

app.post('/generate', async (req, res) => {
  const sid = createSessionId();
  const { character_name, influences = [], canon_sources = [] } = req.body;
  if (!character_name) return res.status(400).json({ error: 'character_name is required' });

  const rawData = await fetchCharacterData(character_name, influences, canon_sources, sid);
  const parsed = await parseScrapedData(rawData, sid);
  const prompt = await buildSystemPrompt(parsed.structuredDataset, sid);
  const vectorMap = await prepareVectorMap(parsed.structuredDataset, sid);

  const profile = { mirror: 'stable', integrity: 'confirmed', uuid: sid };
  const memory = { note: 'Memory not yet initialized', count: 0 };

  const outputDir = buildOutputDirectory(character_name);
  const outputs = {
    'dataset.json': parsed.structuredDataset,
    'system_prompt.md': prompt,
    'vector_map.yaml': vectorMap,
    'psychological_profile.json': profile,
    'memory_snapshots.json': memory
  };
  const paths = await writeAllOutputs(outputDir, outputs, sid);
  res.status(200).json({
    message: 'Digital person schema generated.',
    id: sid,
    output: paths
  });
});

app.listen(3000);
```