Spec triển khai Social Media Well-being Predictor (kèm code mẫu)

Tài liệu này dùng để **bàn giao kỹ thuật đầy đủ** cho nhóm Backend và Frontend. Nội dung đã bao gồm **kiến trúc, phân công, và code mẫu có thể chạy được**.

---

0. Tổng quan luồng hệ thống

Frontend (Web Form) → gửi JSON input → **Backend API (FastAPI)** → Load model + validate input → Predict: stress, happiness, persona → Trả JSON kết quả về FE để hiển thị và gợi ý persona

Backend sẽ **load model một lần khi server start**, không load lại cho mỗi request.

---

1. Bàn giao cho Backend: các file model

Gửi cho Backend **1 file zip** chứa thư mục `models/` (giữ nguyên tên file):

```
models/
├── stress_pipeline.joblib
├── happiness_pipeline.joblib
├── persona_pipeline.joblib
├── features.json
└── persona_labels.json
```

---

2. Cấu trúc repo chuẩn (BE + FE)

```
project-root/
├── backend/
│   ├── app/
│   │   ├── main.py
│   │   ├── ml/
│   │   │   ├── loader.py
│   │   │   └── predictor.py
│   │   ├── schemas.py
│   │   └── utils.py
│   ├── models/
│   ├── requirements.txt
│   └── README.md
└── frontend/
    ├── src/
    │   ├── api/client.ts
    │   ├── pages/PredictorPage.tsx
    │   │
    │   ├── components/UserForm.tsx
    │   ├── components/ResultCard.tsx
    │   └── types.ts
    └── README.md
```

---

3. Backend (FastAPI)

3.1 backend/requirements.txt

```
fastapi==0.115.0
uvicorn[standard]==0.30.6
pydantic==2.8.2
pandas==2.2.2
numpy==2.0.2
scikit-learn
joblib
python-multipart==0.0.9
```

3.2 Load model – app/ml/loader.py

```python
import json
import os
from joblib import load

BASE_DIR = os.path.dirname(os.path.dirname(os.path.dirname(__file__)))
MODELS_DIR = os.path.join(BASE_DIR, "models")

class MLArtifacts:
    def __init__(self):
        self.stress_pipeline = None
        self.happiness_pipeline = None
        self.persona_pipeline = None
        self.features = None
        self.persona_labels = None


def load_artifacts() -> MLArtifacts:
    a = MLArtifacts()

    a.stress_pipeline = load(os.path.join(MODELS_DIR, "stress_pipeline.joblib"))
    a.happiness_pipeline = load(os.path.join(MODELS_DIR, "happiness_pipeline.joblib"))
    a.persona_pipeline = load(os.path.join(MODELS_DIR, "persona_pipeline.joblib"))

    with open(os.path.join(MODELS_DIR, "features.json"), encoding="utf-8") as f:
        a.features = json.load(f)

    with open(os.path.join(MODELS_DIR, "persona_labels.json"), encoding="utf-8") as f:
        raw = json.load(f)
        a.persona_labels = {int(k): v for k, v in raw.items()}

    return a
```

3.3 Predict – app/ml/predictor.py

```
import pandas as pd


def build_input_df(payload: dict, features: list[str]) -> pd.DataFrame:
    row = {f: payload.get(f, None) for f in features}
    return pd.DataFrame([row], columns=features)


def predict_all(artifacts, payload: dict) -> dict:
    X = build_input_df(payload, artifacts.features)

    stress = float(artifacts.stress_pipeline.predict(X)[0])
    happiness = float(artifacts.happiness_pipeline.predict(X)[0])

    pp = artifacts.persona_pipeline
    pf = pp["cluster_features"]

    cluster_row = {f: payload.get(f, None) for f in pf}
    cluster_df = pd.DataFrame([cluster_row], columns=pf)

    row_scaled = pp["scaler"].transform(cluster_df)
    persona_id = int(pp["kmeans"].predict(row_scaled)[0])
    persona_name = artifacts.persona_labels.get(persona_id, f"Cluster_{persona_id}")

    return {
        "stress": stress,
        "happiness": happiness,
        "persona_id": persona_id,
        "persona_name": persona_name,
    }
```

3.4 Schema – app/schemas.py

```
from pydantic import BaseModel
from typing import Dict, Any

class PredictRequest(BaseModel):
    data: Dict[str, Any]

class PredictResponse(BaseModel):
    stress: float
    happiness: float
    persona_id: int
    persona_name: str
```

3.5 Utils – app/utils.py

```
def validate_payload(payload: dict, features: list[str]) -> list[str]:
    return [f for f in features if f not in payload]
```

3.6 FastAPI entry – app/main.py

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware

from app.ml.loader import load_artifacts
from app.ml.predictor import predict_all
from app.schemas import PredictRequest, PredictResponse
from app.utils import validate_payload

app = FastAPI(title="Social Media Well-being Predictor")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

artifacts = load_artifacts()

@app.get("/health")
def health():
    return {"status": "ok"}

@app.get("/features")
def features():
    return {"features": artifacts.features}

@app.post("/predict", response_model=PredictResponse)
def predict(req: PredictRequest):
    payload = req.data
    try:
        return predict_all(artifacts, payload)
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

4. Frontend (React + TypeScript)

4.1 API client – src/api/client.ts

```
export const API_BASE = import.meta.env.VITE_API_BASE || "http://localhost:8000";

export async function getFeatures(): Promise<string[]> {
  const res = await fetch(`${API_BASE}/features`);
  const data = await res.json();
  return data.features;
}

export async function predict(payload: Record<string, any>) {
  const res = await fetch(`${API_BASE}/predict`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ data: payload })
  });
  return res.json();
}
```

4.2 Types – src/types.ts

```
export type PredictResult = {
  stress: number;
  happiness: number;
  persona_id: number;
  persona_name: string;
};
```

4.3 User Form – components/UserForm.tsx

```
export function UserForm({ features, values, onChange, onSubmit }: any) {
  const isNumber = (f: string) => /(_count|_hours|_minutes|_rate|age)/.test(f);

  return (
    <div>
      {features.map((f: string) => (
        <input
          key={f}
          placeholder={f}
          type={isNumber(f) ? "number" : "text"}
          value={values[f] ?? ""}
          onChange={(e) => onChange(f, isNumber(f) ? Number(e.target.value) : e.target.value)}
        />
      ))}
    <button onClick={onSubmit}>Predict</button>
  </div>
 );
}
```

4.4 Result Card – components/ResultCard.tsx

```
import { PredictResult } from "../types";

export function ResultCard({ result }: { result: PredictResult }) {
 return (
  <div>
    <div>Stress: {result.stress.toFixed(2)}</div>
    <div>Happiness: {result.happiness.toFixed(2)}</div>
    <div>Persona: {result.persona_name}</div>
  </div>
 );
}
```

---

5. Phân công

- **Người 1 (BE)**: FastAPI + model + API
- **Người 2 (FE)**: Form + call API + render kết quả

Chỉ cần làm đúng spec này là hệ thống chạy end-to-end.