NephroCompass / CKD Predictor — Technical Specification

Last updated: today • Status: Production-ready

---

1) Product overview

NephroCompass is a transparent CKD (Chronic Kidney Disease) risk screening tool composed of:

- a **Streamlit web UI** for clinicians/data teams, and
- a **FastAPI backend** that hosts ML models, explanations, simulations, and LLM-powered guidance.

It supports single-patient checks, bulk CSV scoring, SHAP-driven explanations (with local fallback), digital-twin what-ifs, counterfactual generation, nearest-neighbor patient similarity, and a care-plan “agents” endpoint.

> **Clinical note**: This app is decision support, not diagnosis. Messages and outputs include an explicit disclaimer.

---

2) User-facing features (UI tabs)

2.1 Single check (compare models)

- **Purpose**: Score a single patient against one or more models (RF/XGBoost/LogReg).
- **Inputs**: Age, sex, BP, creatinine, BUN, GFR, ACR, Na, K, Hb, HbA1c.
- **Derived** (computed client-side to stay consistent with backend): `pulsepressure = SBP - DBP`, `urea/creatinine ratio`, CKD **stage** (by GFR), **albuminuria category** (by ACR), and flags (BP risk, hyperkalemia, anemia).

- **Outputs**:
    - Probability of CKD, decision threshold, and flag status per model.
    - **Top drivers** bar chart for feature influence (server SHAP first; local sensitivity fallback if SHAP not available).

2.2 Bulk check (CSV)

- **Purpose**: Batch score cohorts.
- **Artifacts**: Blank template + sample CSV downloads.
- **Outputs**:
    - Per-model batch summary: positive rate, average probability, rows processed, threshold.
    - Per-model CSV export and a combined CSV export (inputs + predictions).
    - Optional trigger to **retrain** models server-side.

2.3 Service & logs

- **Purpose**: Operational visibility.
- **Panels**:
    - `/health` readout (model, threshold, DB backend, module wiring).
    - Recent inference log (`/metrics/last_inferences`).
    - Last training report if available (`/metrics/retrain_report`).
    - Buttons to **reload** models and start **retrain**.

2.4 AI summary & next steps

- **Purpose**: LLM-generated clinical next steps for the last single prediction.

- **Provider**: OpenAI-compatible (default OpenRouter) from the **frontend**.
- **Guardrails**: No dosing; plain-language patient comms; explicit disclaimer.

2.5 Digital Twin (What-If)

- **Purpose**: Simulate effect of changes (simple deltas and/or grid sweeps).
- **Backend**: `/whatif` endpoint (see §5.3).
- **Output**: Rows with recomputed derived metrics and re-scored probabilities.

2.6 Counterfactuals

- **Purpose**: Find actionable changes to reduce risk below a target probability.
- **Backend**: `/counterfactual` (greedy search; optional DiCE when configured).
- **Output**: Search steps, final candidate (or DiCE candidates), achieved probability.

2.7 Similar Patients

- **Purpose**: k-NN similarity search within an uploaded cohort.
- **Backend**: `/similar` (Euclidean across normalized/derived features).
- **Output**: Top-k neighbor rows with `_distance`.

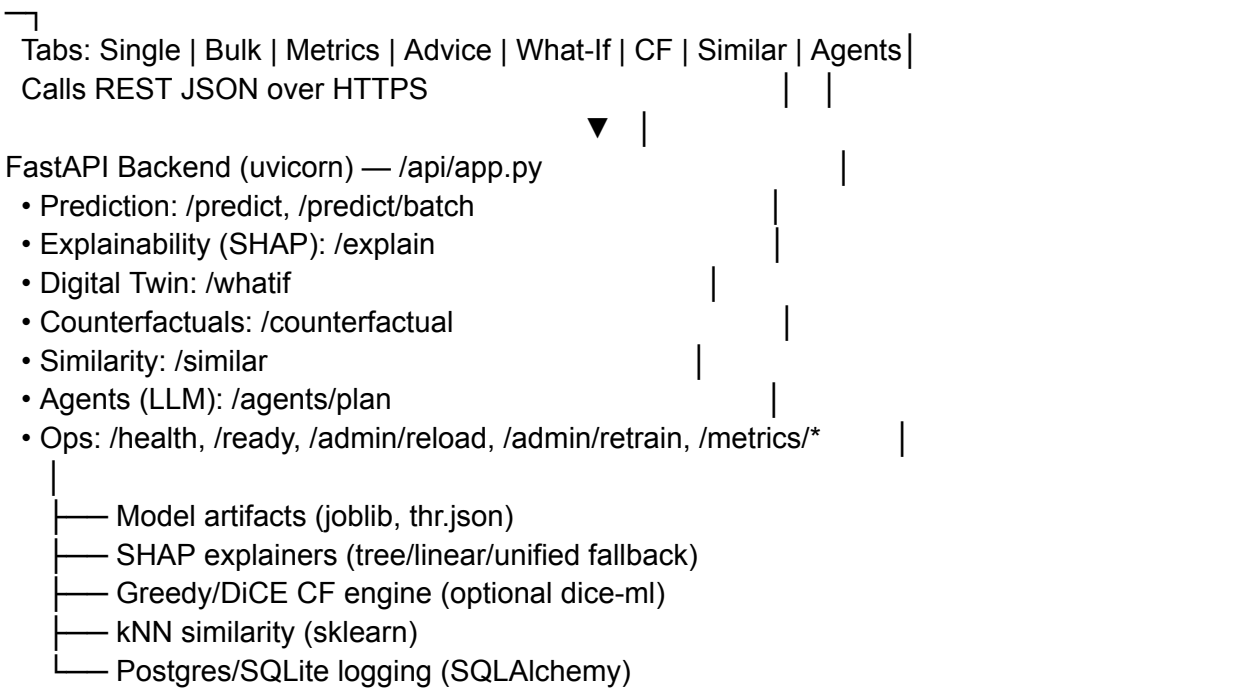
2.8 Care Plan (Agents)

- **Purpose**: Server-side LLM plan via `/agents/plan` with structured sections:
    - Clinical Interpretation
    - Diet & Lifestyle
    - Follow-up & Referrals
    - Patient Communication (Plain Language)

- **Output**: JSON `{ sections: [{title, bullets|text}, ...] }` for UI rendering.

---

3) Technical architecture



- **Model registry** (files in `models/`):
    - `xgb_ckd.joblib`, `xgb_ckd_threshold.json`
    - `rf_ckd.joblib`, `rf_ckd_threshold.json`
    - `logreg_ckd.joblib`, `logreg_ckd_threshold.json`
- **Threshold** stored per model; **legacy flip** supported (for historical XGB proba semantics).
- **Caching**: models loaded once and cached in-process; `/admin/reload` clears cache.

- **Database**:
    - `user_inputs`: all features per request (for audit).
    - `inference_log`: model + probability + threshold per run.
- **Health & readiness**:
    - `/health`: model/threshold, DB status, feature list, module wiring status+errors.
    - `/ready`: booleans for agents/whatif/counterfactual/similar + import errors.

---

4) Data contracts

4.1 Features (order is canonical)

age, gender,
systolicbp, diastolicbp,
serumcreatinine, bunlevels,
gfr, acr,
serumelectrolytessodium, serumelectrolytespotassium,
hemoglobinlevels, hba1c,
pulsepressure, ureacreatinineratio,
ckdstage, albuminuriacat,
bp_risk, hyperkalemiaflag, anemiaflag

4.2 Derived fields (kept consistent across modules)

- `pulsepressure = systolicbp - diastolicbp`
- `ureacreatinineratio = bunlevels / (serumcreatinine + 1e-6)`
- **CKD Stage** by GFR (coarse bins): G1≥90, G2 60–89, G3a 45–59, G3b 30–44 (3), G4 15–29 (4), G5<15 (5)
- **Albuminuria** by ACR: A1<30 (1), A2 30–300 (2), A3>300 (3)
- Flags:

    - `bp_risk = 1 if SBP≥130 or DBP≥80`
    - `hyperkalemiaflag = 1 if K≥5.5`
    - `anemiaflag = 1 if Hb<12.0`
- Bounded electrolytes for plausibility (Na: 110–170, K: 2.0–7.5)

---

5) Backend endpoints

5.1 Core

- `GET /health?model=rf|xgb|logreg` → JSON: status, threshold, db info, module wiring
- `GET /ready` → JSON booleans for advanced modules + import error strings
- `POST /predict?model=...` → `PredictResponse {prediction, prob_ckd, prob_non_ckd, threshold_used, model_used}`
- `POST /predict/batch?model=...` → `BatchPredictResponse {predictions: PredictResponse[]}`

5.2 Explainability

- `POST /explain?model=...` → `ExplainResponse { base_value, shap_values, top[] }`
  Internals:
    - Tree models: `shap.TreeExplainer(model_output="probability")`
    - Linear models: `shap.LinearExplainer`
    - Unified `shap.Explainer` fallback

- Final fallback to `feature_importances_`/`coef_` if SHAP fails

5.3 Simulation (Digital Twin)

- `POST /whatif`
  - Body (single): `{ base, deltas, model }`
  - Body (grid): `{ base, grid:{feat:[...values...]}, model }`
  - Returns `{mode:"single"|"grid", rows:[...], probs:[{prob_ckd, prob_non_ckd}], model}`

5.4 Counterfactuals

- `POST /counterfactual`
  - Body: `{ base, target_prob, model, method:"auto"|"greedy" }`
  - **Greedy**: iteratively nudges actionable features (SBP/DBP, HbA1c, ACR, creatinine, BUN, K/Na toward targets, Hb, GFR) with clipping and re-derivation until `prob_ckd≤target` or convergence.
  - **DiCE** (optional): enabled when `dice-ml` is installed and `CF_DICE_DATA_CSV` is set; returns candidates and scores.

5.5 Similarity

- `POST /similar?k=5`
  - Body: `{ base, cohort: [...] }`
  - Returns `{ neighbors: [row+_distance] }` using `NearestNeighbors(metric="euclidean")`.
  - Internally normalizes/derives features to align with model schema.

5.6 Agents (server-side LLM)

- `POST /agents/plan`
    - Body: concise case summary `{model, prob_ckd, threshold, flags, stage, albuminuriacat, metrics{...}}`
    - Returns `{ sections: [{title, bullets|text}, ...] }` (or `{plan: "...md..."}` as fallback).
    - Uses OpenAI-compatible client with HTTP fallback. No dosing; ends with disclaimer.

5.7 Operations

- `POST /admin/retrain?sync=false` → runs `ml/99_retrain.py` (async by default); clears model cache on success.
- `POST /admin/reload` → clears model cache (204).
- `GET /metrics/retrain_report` → last training report JSON, if present.
- `GET /metrics/last_inferences?limit=10` → most recent predictions (timestamped).

---

6) Storage schema (SQLAlchemy)

```
CREATE TABLE user_inputs (
  id BIGSERIAL PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  -- all 19 input features (DOUBLE PRECISION/INT)
);

CREATE TABLE inference_log (
  id BIGSERIAL PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  input_id BIGINT REFERENCES user_inputs(id) ON DELETE CASCADE,
  model_used TEXT NOT NULL,
  threshold_used DOUBLE PRECISION NOT NULL,
```

```
  prediction INT NOT NULL,
  prob_ckd DOUBLE PRECISION NOT NULL,
  prob_non_ckd DOUBLE PRECISION NOT NULL
);

CREATE INDEX inference_log_created_at_idx ON inference_log(created_at DESC);
```

Backend uses `DATABASE_URL` if set; otherwise falls back to a local SQLite file `models/inference.db` so the Metrics tab works everywhere.

---

7) Configuration & environment

Core

- `MODEL_DIR` (optional) – directory of joblib/threshold files; default `./models`
- `DATABASE_URL` – Postgres URL (Render Neon by default)
- `ADMIN_TOKEN` – optional header token for `/admin/*` endpoints

LLM (agents)

- `LLM_API_KEY` – required in production
- `LLM_BASE_URL` – default `https://openrouter.ai/api/v1` (any OpenAI-compatible endpoint)
- `LLM_MODEL` – default `meta-llama/llama-3.1-8b-instruct`
- `APP_URL`, `APP_TITLE` – optional OpenRouter best-practice headers

DiCE (optional)

- `CF_DICE_DATA_CSV` – path to a CSV of training-like rows enabling DiCE generation

---

8) Dependencies (pinned)

```
fastapi==0.115.0
uvicorn==0.30.6
pydantic==2.9.2
numpy==1.26.4
pandas==2.2.2
scikit-learn==1.4.2
xgboost==2.0.3
SQLAlchemy==2.0.31
psycopg2-binary==2.9.9
python-dotenv==1.0.1
shap==0.44.1
numba==0.59.1
llvmlite==0.42.0
openai
requests>=2.31.0
# optional:
dice-ml
crewai
langgraph
```

Packaging notes

- Ensure `api/__init__.py` exists so imports like `from api.agents import ...` work in production.
- Start command should target the package path:
  `uvicorn api.app:app --host 0.0.0.0 --port $PORT`

---

9) Security & compliance

- **CORS**: Wide-open in dev (`allow_origins=["*"]`). Tighten in production.
- **Admin**: `/admin/*` guarded by optional `ADMIN_TOKEN` header.
- **PII**: Inputs may be sensitive; no direct identifiers stored. Ensure DB and backups meet org policies.

- **LLM**: Only summary metrics/flags are sent to the LLM endpoints; avoid raw PHI in prompts.

---

10) Error handling & UX guarantees

- If advanced modules are missing, API returns **503** with a clear `detail` message:
    - `"multi_agent_plan not available"`, `"simulate_whatif not available"`, etc.
- `/health` and `/ready` expose **module wiring status** and **import errors** to surface packaging issues (e.g., `ModuleNotFoundError: No module named 'api'`).
- SHAP failures fall back to finite-difference **local sensitivity** so the UI always has “top drivers.”
- Batch endpoints surface partial errors per model but keep successful outputs downloadable.

---

11) Performance & scaling

- **Cold start**: First request loads models into cache; subsequent calls are fast.
- **Batch**: Use `/predict/batch` (vectorized predict_proba) to avoid per-row overhead.
- **DB**: Inserts are batched within a single transaction for batch scoring.
- **SHAP**: Single-row explanations; heavy models may be slower—tree/linear explainer selection optimizes common paths.

---

12) Deployment (Render)

- Repository must contain `api/app.py` and **`api/__init__.py`**.
- **Start Command**: `uvicorn api.app:app --host 0.0.0.0 --port $PORT`
- **Environment**: Set `DATABASE_URL`, `LLM_*`, `ADMIN_TOKEN` as needed.
- After code changes:
    1. “Clear build cache” → Deploy
    2. Verify:
        - `GET /ready` — all four booleans should be `true`
        - `GET /health?model=rf` — `"agents_loaded": true`, etc.

---

13) Testing checklist

- **Unit**:
    - Feature transformations (derived fields, flags, bins).
    - Threshold and legacy flip behavior.
    - SHAP top-k extraction order and types.
- **API**:
    - `/predict` happy path + missing feature error (400).
    - `/explain` happy path + forced SHAP fallback.
    - `/whatif` single & grid (with recomputed derived).
    - `/counterfactual` greedy convergence and min-improve stopping.
    - `/similar` shape of neighbors, k bounds.

    - `/agents/plan` returns `sections` with required titles.
    - `/ready` booleans reflect module presence.
- **UI**:
    - Validations on numeric bounds.
    - Batch CSV columns and downloads.
    - Graceful error panels for 5xx/503 responses.

---

14) Runbook (common issues)

- **Advanced tabs disabled** in UI:
    - `GET /ready` shows which modules failed and precise import error.
    - Ensure `api/__init__.py`, correct Start Command, and that `requests` is in requirements (agents fallback).
- **SHAP times out or errors**:
    - Confirm versions (NumPy 1.26.x, shap 0.44.1).
    - Verify model type (tree/linear) and try another model.
    - UI will switch to local sensitivity automatically.
- **DB errors**:
    - Check `DATABASE_URL`, outbound networking, or fall back to SQLite.
- **LLM errors**:
    - Verify `LLM_API_KEY`, `LLM_BASE_URL`, and model string; inspect `/health.modules.agents_error`.

---

15) Roadmap

- Add **model drift** and data quality checks to `/metrics`.
- Enrich **similarity** with clinically weighted distance metrics.
- Optional **feature store** backend to source cohorts without CSV.
- Role-based auth and per-user audit trails.
- Add **explanation caching** for repeat queries.

---

16) Example requests

```
# Health & readiness
curl -s https://<host>/health?model=rf | jq
curl -s https://<host>/ready | jq

# Single prediction
curl -s -X POST "https://<host>/predict?model=rf" \
  -H "content-type: application/json" \
  -d '{"age":58,"gender":1,"systolicbp":150,"diastolicbp":90,"serumcreatinine":2.1,"bunlevels":32,
       "gfr":52,"acr":120,"serumelectrolytessodium":139,"serumelectrolytespotassium":4.8,
       "hemoglobinlevels":12.2,"hba1c":6.5,"pulsepressure":60,"ureacreatinineratio":15.2,
       "ckdstage":2,"albuminuriacat":2,"bp_risk":1,"hyperkalemiaflag":0,"anemiaflag":0}'

# Explain
curl -s -X POST "https://<host>/explain?model=rf" -H "content-type: application/json" -d
@payload.json

# What-if (single)
curl -s -X POST "https://<host>/whatif" -H "content-type: application/json" \
  -d '{"base":{...features...}, "deltas":{"systolicbp":-10,"acr":-50}, "model":"rf"}'

# Counterfactual
curl -s -X POST "https://<host>/counterfactual" -H "content-type: application/json" \
  -d '{"base":{...features...}, "target_prob":0.2, "model":"rf", "method":"greedy"}'

# Similar
curl -s -X POST "https://<host>/similar?k=5" -H "content-type: application/json" \
  -d '{"base":{...features...},"cohort":[{...},{...}]}'
```

```
# Agents plan
curl -s -X POST "https://<host>/agents/plan" -H "content-type: application/json" \
  -d
'{"model":"rf","prob_ckd":0.41,"threshold":0.25,"flags":{"bp_risk":1,"hyperkalemia":0,"anemia":1},
      "stage":3,"albuminuriacat":2,"metrics":{"gfr":52,"acr":120,"hba1c":6.5,"systolicbp":150,
      "diastolicbp":90,"potassium":4.8,"hb":12.2}}'
```

Appendix A — Start command & packaging

- **Start**: `uvicorn api.app:app --host 0.0.0.0 --port $PORT`

Repo layout:

```
 api/
  __init__.py
  app.py
  agents.py
  digital_twin.py
  counterfactuals.py
  similarity.py
models/
  rf_ckd.joblib
  rf_ckd_threshold.json
  ...
ml/
  99_retrain.py
ui/
  app.py
```

-
- **Readiness checks**: `/ready` must show all four modules `true` before enabling advanced tabs.

If you want this exported as a PDF or included in your repo as `TECHNICAL_SPEC.md`, say the word and I'll generate the file.