HealthAI — Full Implementation Plan (Complete Project, Not Placeholders)

This document replaces the previous scaffold and now contains a **complete, actionable plan** and a prioritized TO-DO list you can share with Cursor (or implement manually). The goal is to **deliver a working, end-to-end HealthAI project**: data ingestion → cleaning → EDA → classical ML → DL → time-series → NLP → model serving → Streamlit UI, all orchestrated with Docker.

📌High-level Deliverables (what "complete" means)

1. Clean repository structure, no duplicates, no dead files.
2. Working Docker Compose setup to run FastAPI (8000) and Streamlit (8501).
3. Fully implemented backend services (no placeholders):
4. Data ingestion & cleaning pipeline
5. EDA service with visualizations and serializable results
6. Classical ML pipeline (train/test/serialize): logistic regression, linear regression, KMeans, association rules
7. Deep Learning pipelines: tabular NN, CNN for medical images, evaluation and model saving
8. Time series pipelines: RNN/LSTM forecasting
9. NLP pipelines: tokenization, BioBERT embeddings, sentiment
10. Model evaluation + clinical-interpretation translator
11. AI tools endpoints (summarize, classify, sentiment)
12. Streamlit UI with complete pages that call API endpoints and render interactive visualizations and tables.
13. Notebooks and scripts that reproduce preprocessing, EDA, training, and evaluation.
14. Unit tests for core functions and smoke tests for endpoints.
15. README, API docs, and minimal developer onboarding instructions.

🧹Revised Folder Structure (same as earlier but with implementation notes)

```
project/
├── docker-compose.yml        # two services: api, ui
├── requirements.txt
├── README.md
├── api/
│   ├── main.py
│   ├── routes/
│   │   ├── data.py
│   │   ├── eda.py
│   │   ├── models.py
│   │   ├── ai_tools.py
│   │   └── predictions.py
│   ├── core/
```

```
│   │   ├── config.py        # paths, constants, logging
│   │   └── utils.py         # I/O helpers
│   ├── services/
│   │   ├── data_service.py          # ingestion, cleaning, save/load
│   │   ├── eda_service.py           # run_eda(df)->dict
(stats,plots,matrices)
│   │   ├── classical_ml_service.py  # train_logistic, train_linear,
clustering, feature_select
│   │   ├── dl_service.py            # tabular_nn, cnn_image
│   │   ├── time_series_service.py   # train_lstm, forecast
│   │   └── nlp_service.py           # tokenize, embed, sentiment
│   └── ai_tools/
│       ├── summarizer.py
│       ├── classifier.py
│       └── sentiment.py
├── ui/
│   ├── dashboard.py
│   ├── pages/
│   │   ├── home.py
│   │   ├── data_view.py
│   │   ├── eda_overview.py
│   │   ├── ml_models.py
│   │   ├── dl_models.py
│   │   ├── nlp_analysis.py
│   │   ├── time_series.py
│   │   └── ai_tools.py
│   └── components/
│       ├── sidebar.py
│       └── viz_helpers.py
├── data/
│   ├── raw/                 # place CSV/PNG/JSON here
│   └── cleaned/             # cleaned CSVs, small sample datasets
├── notebooks/
│   ├── data_cleaning.ipynb
│   ├── eda.ipynb
│   ├── classical_ml.ipynb
│   ├── deep_learning.ipynb
│   ├── nlp.ipynb
│   └── time_series.ipynb
├── scripts/
│   ├── clean_data.py
│   ├── run_eda.py
│   ├── train_classical_ml.py
│   ├── train_dl.py
│   ├── run_nlp.py
│   └── run_time_series.py
├── models/                  # serialized models (.pkl, .pt)
├── tests/                   # unit + integration tests
└── docker/
```

```
├── api/Dockerfile
└── ui/Dockerfile
```

🔗Prioritized, Actionable TODOs (suitable to paste into Cursor)

Phase 0 — Prework (Repo hygiene)

- [ ] **Remove duplicates & dead files**: identify files with zero references (scripts, old notebooks) and delete or archive to `archive/`.
- [ ] **Add** `.env.example` with all env var names; ensure `api` and `ui` Dockerfiles read from env.
- [ ] **Pin dependencies** in `requirements.txt` (use exact minimal set: fastapi, uvicorn, pandas, numpy, scikit-learn, matplotlib, seaborn, plotly, streamlit, torch, transformers, datasets, mlxtend, mlflow (optional)).

Phase 1 — Data layer (foundation)

- [ ] `api/services/data_service.py`
- Implement `list_raw()`, `load_raw(name)`, `clean_and_save(raw_name, cleaned_name)`
- Add robust parsing for CSV/JSON, handle date columns and missing values
- Add small sample dataset(s) in `data/raw/` (e.g., `patients_sample.csv`, `images_sample.zip`)
- [ ] `scripts/clean_data.py` that runs `data_service.clean_and_save` from CLI
- [ ] Store cleaned outputs to `data/cleaned/`

Phase 2 — EDA (complete)

- [ ] `api/services/eda_service.py` — fully implemented functions:
- `basic_stats(df)` → returns mean, median, std, min, max, count for numeric columns
- `distribution_plots(df, columns)` → generate histograms/density plots and save PNGs or return base64 images
- `correlation_matrix(df)` → returns correlation DataFrame and heatmap image
- `boxplot_by_category(df, feature, category)` → boxplot image and summary stats
- `compare_groups(df, group_col, features)` → t-test or Mann-Whitney where appropriate
- Expose a single function `run_full_eda(df, config)` that returns a JSON with: stats, base64 plots, correlation table
- [ ] `api/routes/eda.py` — add endpoints:
- `POST /eda/run` (input: cleaned dataset name + options) → kicks off `run_full_eda` and returns results
- `GET /eda/{dataset}/plots/{plotname}` → return images
- [ ] `notebooks/eda.ipynb` and `scripts/run_eda.py` that call same service methods

Phase 3 — Classical ML (complete)

Goal: fully implement training, evaluation, serialization for the requested models.

- [ ] `api/services/classical_ml_service.py` functions:

- `prepare_features(df, target, feature_selection=None)` — handle categorical encoding, scaling
- `train_logistic_regression(df, target)` → train, return model metrics (accuracy, AUC, precision, recall), persist model under `models/classification/diabetes_lr.pkl`
- `feature_selection(df)` → correlation-based and SelectKBest (return selected feature list)
- `train_linear_regression(df, target)` → train, persist, return RMSE
- `train_kmeans(df, n_clusters)` → fit, return cluster labels, save model
- `association_rules(df, transactions_col)` → Apriori rule mining (mlxtend), return top rules
- [] `api/routes/models.py` endpoints:
- `POST /model/train/classification` (payload: dataset, target, hyperparams) → returns metrics and model_id
- `POST /model/train/regression` → LOS prediction
- `POST /model/train/clustering` → returns clusters
- `GET /model/{model_id}/predict` → apply model to new data
- [] `scripts/train_classical_ml.py` CLI wrappers

Phase 4 — Deep Learning (complete)

- [] `api/services/dl_service.py` implementations:
- `train_tabular_nn(df, target, model_config)` → PyTorch model, early stopping, save `.pt`
- `train_cnn_image(train_dir, val_dir, model_config)` → basic CNN (Conv->Pool->FC), binary classification flow
- `evaluate_model(model, data)` → compute accuracy, AUC, RMSE (as appropriate)
- `clinical_interpretation(metrics)` → Map metrics to clinical significance paragraph
- [] `api/routes/predictions.py` to expose DL prediction endpoints
- [] `notebooks/deep_learning.ipynb` demonstrating training on small sample

Phase 5 — Time Series (complete)

- [] `api/services/time_series_service.py`:
- `train_lstm(series, lookback, params)` → training loop, save model
- `forecast(model, steps)` → return predicted points + confidence
- [] `ui/pages/time_series.py` to show forecasts and input params

Phase 6 — NLP (complete)

- [] `api/services/nlp_service.py`:
- `tokenize_clinical(text)` → specialized tokenizer (handle lab values, tokens)
- `embed_with_biobert(text)` → use `transformers` pipeline + `dmis-lab/biobert` (or note local weights)
- `sentiment_analysis(text)` → rule-based + model fallback
- [] `ui/pages/nlp_analysis.py` to show tokenization, embeddings sample, and sentiment

Phase 7 — Model Evaluation & Clinical Interpretation

- [] `api/services/eval_utils.py`:
- `calc_metrics_classification(y_true, y_pred, y_scores)` → accuracy, AUC, precision, recall, F1
- `calc_metrics_regression(y_true, y_pred)` → MAE, RMSE, R2

- `format_for_clinical(metrics)` → human-readable paragraph mapping numbers to clinical risk
- [ ] Expose `GET /model/{id}/metrics` and `GET /model/{id}/interpretation`

Phase 8 — Streamlit UI (complete)

- [ ] `ui/components/sidebar.py`: full nav with icons and badges for model status
- [ ] `ui/pages/eda_overview.py`: call `/eda/run` and render the returned base64 plots + tables
- [ ] `ui/pages/ml_models.py`: form to run training jobs, show training status, metrics, confusion matrices
- [ ] `ui/pages/dl_models.py` & `ui/pages/time_series.py` & `ui/pages/nlp_analysis.py`: similar patterns
- [ ] `ui/pages/data_view.py`: table viewer with filter/search and option to download cleaned CSV
- [ ] `ui/pages/ai_tools.py`: text box to test summarizer/classifier endpoints

Phase 9 — Tests, CI, and Validation

- [ ] Unit tests for each service function; small sample data fixtures stored in `tests/fixtures/`
- [ ] Integration tests for routes using `httpx` and TestClient
- [ ] Add `pre-commit` with flake8/isort/black
- [ ] Optional: Add GitHub Actions pipeline to run tests and `docker-compose build`

Phase 10 — Finalize Docs and Handoff

- [ ] Complete `README.md` with run instructions and sample API calls
- [ ] Add `CHANGELOG.md` and `DEVELOPMENT.md` with coding standards and contribution guide
- [ ] Create a short demo script `demo.sh` to populate sample data, run training for one model, and start UI

🕐Implementation Details & Standards (for Cursor to follow)

1. **APIs return JSON** with clear schemas; for heavy assets (plots, models) return URLs or base64.
2. **Model artifacts** saved under `models/<type>/<name>.<pkl|pt>`; include a `metadata.json` for each model containing training params, dataset name, feature list, and timestamp.
3. **Determinism**: set random seeds for sklearn/numpy/torch during training for reproducibility.
4. **Small sample datasets**: ship minimal synthetic datasets (100–200 rows) to enable dev/test without large data.
5. **Logging**: use Python `logging` with sensible log levels; errors must return helpful messages but not crash server.
6. **Security**: do not allow arbitrary file writes or code execution via endpoints; validate inputs.

🛠️Example Cursor Prompt (paste this to perform the full conversion)

```
Refactor and fully implement the HealthAI project to the specifications
below. Keep existing working FastAPI and Streamlit behavior intact. Perform
the following actions:

1. Apply the folder structure exactly as shown in this document.
2. Remove duplicate/obsolete files and move any unreferenced notebooks into
`notebooks/archive/`.
3. Create and implement the following new files (complete functional
implementations, not placeholders):
   - `api/services/data_service.py` (ingest, clean_and_save, list/load)
   - `api/services/eda_service.py` (basic_stats, distribution_plots,
correlation_matrix, boxplot_by_category, run_full_eda)
   - `api/services/classical_ml_service.py` (train_logistic_regression with
feature selection, train_linear_regression, train_kmeans, association_rules)
   - `api/services/dl_service.py` (train_tabular_nn, train_cnn_image,
evaluate_model, clinical_interpretation)
   - `api/services/time_series_service.py` (train_lstm, forecast)
   - `api/services/nlp_service.py` (tokenize_clinical, embed_with_biobert,
sentiment_analysis)
   - `api/routes/eda.py`, `api/routes/models.py`, `api/routes/
predictions.py`, `api/routes/data.py` — fully wired to services
   - `ui/pages/*.py` — fully functional Streamlit pages calling the API and
rendering results
   - `scripts/*.py` — CLI thin wrappers to call services
4. Add small sample datasets under `data/raw/` and ensure `scripts/
clean_data.py` produces `data/cleaned/sample_cleaned.csv`.
5. Add unit tests for key service functions and integration tests for API
routes.
6. Update `README.md` with exact run instructions and example API calls.
7. Ensure `docker-compose up` runs both services and that all endpoints
return healthy responses.

Make reasonable design choices where unspecified but minimize third-party
dependencies. After changes, provide a summary of moved/created/deleted files
and a final folder tree.
```

📋Deliverable Checklist (copyable)

- [ ] Repo cleaned and reorganized
- [ ] Sample data in `data/raw/` and `data/cleaned/`
- [ ] EDA service + endpoints implemented
- [ ] Classical ML implemented + persisted models
- [ ] DL pipelines implemented (tabular + CNN)
- [ ] Time series pipelines implemented (LSTM)

- [ ] NLP service implemented (tokenize + embed + sentiment)
- [ ] Streamlit UI pages for each area
- [ ] Scripts and notebooks that reproduce steps
- [ ] Unit + integration tests
- [ ] README and demo script

If you want, I can now: 1. Break down **Phase 2 (EDA)** into exact code files with function signatures and short implementation snippets. OR 2. Create the **Cursor job** that runs the refactor and returns a change summary (requires Cursor to have repo access).

Which of these two would you like me to produce next? If neither, tell me which phase you want fully written out into file-by-file implementation details and I'll expand that immediately.