Instrukcja - Frontend / Backend / Models / Tests

1 1) Frontend (React + TS + Vite)

Lokalizacja: `./frontend/`
UI: jedna strona `/` (**Analiza**) z 2 wykresami:

- BTC candlestick (Binance) + punkt predykcji z wybranego modelu
- wykres słupkowy dla wybranej metryki (dropdown)

Uruchomienie (lokalnie)

```
cd frontend
npm install
npm run dev
```

Wejdź na adres z konsoli (zwykle) `http://localhost:5173`.

Backend wymagany pod: `http://localhost:8000/marketdata`
Stała: `API_BASE` w `frontend/src/pages/Analysis.tsx`

Uruchomienie (Docker)

```
docker compose up --build
```

Budowanie/start: `frontend/Dockerfile`, start wspiera `frontend/docker/entrypoint.sh`.

Jak UI mapuje się na API

- Wejście na stronę:
  - `GET /marketdata/historical-data/?years_back=2`
- Kliknięcie **Uruchom model**:
  - `GET /marketdata/<endpoint-modelu>/`
  - wynik `close_btc` jest wyświetlany jako liczba + punkt predykcji na wykresie BTC
- Dropdown metryk (drugi wykres):
  - wybór metryki z danych historycznych (np. `close_btc`, `volume_eth`, `gdp`, `unrate`, ...)

Szybka mapa kodu

- `frontend/src/App.tsx` - routing + layout (nagłówek/stopka)
- `frontend/src/pages/Analysis.tsx` - fetch danych, wykresy, zoom-sync, uruchamianie modeli

- `frontend/src/main.tsx` - bootstrap React
- `frontend/src/index.css` + `tailwind.config.js` - Tailwind

Typowe problemy

- CORS / brak danych: backend musi działać na `:8000` i mieć ścieżki `/marketdata/...`
- Komunikat „Brak danych z endpointu historical-data.": backend zwraca pustą listę albo request nie dochodzi (sprawdź logi backendu i URL)

2 2) Backend (Django API: marketdata + inferencja)

Cel: API do danych rynkowych + inferencja modeli ML.
Brak "biznesowej"DB: `models.py` jest puste; aplikacja jest w praktyce plikowo-obliczeniowa.

Stack (minimum)

- Django 5.x + DRF (`APIView`, `Response`)
- `django-cors-headers`
- `requests`, `pandas`/`numpy`/`pyarrow`, `joblib`, `torch`, `darts` (+ Lightning dla TFT)

Uwaga praktyczna: dopisz `rest_framework` do `INSTALLED_APPS` dla przewidywalności środowiska testowego i DRF.

Uruchomienie (Docker)

```
docker compose up --build
```

Backend typowo wystawia: `0.0.0.0:8000`.

Uruchomienie (lokalnie)

```
pip install -r backend/requirements.txt
# wymagane dla FRED:
export FRED_API_KEY="..."
python backend/manage.py runserver 8000
```

Routing

- `/admin/`
- `/marketdata/` → `backend/marketdata/urls.py`

Endpointy (skrót)

Testy źródeł

- `GET /marketdata/binance-test/` - BTC ticker/price
- `GET /marketdata/coinmetrics-test/` - ostatni ReferenceRateUSD

- `GET /marketdata/fred-test/` - UNRATE + GDP (wymaga `FRED_API_KEY`)
- `GET /marketdata/stooq-test/` - SPX (CSV → lista rekordów)

Dane historyczne

1) `GET /marketdata/historical-data/` Query:

- `years_back` (domyślnie 10)
- `exclude` (wiele razy i/lub CSV w jednym)
- `parquet` = `1|true|yes|y` → generuje plik parquet zamiast zwracać dane
- `filename` (domyślnie `historical_data.parquet`)

Przykłady:

- generacja parquet: `/marketdata/historical-data/?years_back=5&parquet=true`
- zwrot danych: `/marketdata/historical-data/?years_back=2`
- wykluczenia: `/marketdata/historical-data/?years_back=2&exclude=volume_btc,volume_eth`

2) `GET /marketdata/get-historical-data/` Query:

- `metrics` (wymagane): `close_btc,volume_btc,...`
- `refresh`: `true|1|yes` → wymusza regenerację parquet
- `years_back` (przy regeneracji; domyślnie 10)

Zwraca: lista `{ open_time, ...requested_metrics }`
Przykład: `/marketdata/get-historical-data/?metrics=close_btc,volume_btc,low_btc,open_btc&year`

Modele ML (predykcja `close_btc`)

- `GET /marketdata/naive-model/` - `{ "close_btc": last_close }`
- `GET /marketdata/linear-regression-model/` - LR przewiduje return → `pred_close = last_close * (1 + ret_pred)`
- `GET /marketdata/random-forest-model/` - RF przewiduje log-return → `pred_close = last_close * exp(ret_pred)`
- `GET /marketdata/lstm-model/`
- `GET /marketdata/tft-model/`

Serwisy (odpowiedzialności)

- `HistoricalDataService.py`
  - pobiera i scala: Binance (OHLCV), Stooq (SPX), FRED (GDP/UNRATE)
  - zapis: `backend/data/historical_data.parquet`
- `DataReaderService.py`
  - szybki odczyt Parquet + auto-refresh:

    * freshness: `DATA_FRESHNESS_HOURS = 24`
    * coverage check względem `years_back`
  - czyta tylko wymagane kolumny (`pd.read_parquet(columns=...)`)
- `ModelService.py`
  - dba o świeżość danych + feature engineering + inferencja z artefaktów

Artefakty / zależności spójności

Źródło prawdy dla inferencji: `backend/data/historical_data.parquet` + artefakty modeli.
Zmiana kolumn/nazw w generatorze danych → ryzyko błędów `Missing columns...` w modelach.

„Co gdzie zmieniać"

- nowe źródło danych / aktywo: `HistoricalDataService.*` (symbole, `desired_cols`)
- polityka świeżości: `DataReaderService.DATA_FRESHNESS_HOURS`
- nowy endpoint modelu: `views.py` + `urls.py` + metoda w `ModelService`
- nowa metryka do frontu: musi istnieć w parquet i być uwzględniona w `desired_cols`

3 3) Models (trening + artefakty)

Lokalizacja: `./models/`
Wejście danych: `backend/data/historical_data.parquet`
Wyjście: artefakty do `backend/data/` wykorzystywane przez backend (`ModelService`).

Konwencja targetu

Większość modeli nie przewiduje ceny, tylko **zwrot na kolejny dzień** (`ret_btc_next`), potem backend mapuje to na cenę `close_btc`.

- log-return wariant: `pred_close = last_close * exp(pred_ret)`
- LR wariant (pct_change): `pred_close = last_close * (1 + pred_ret)`

Artefakty (lokalizacje)

- `backend/data/linear_regression_btc.pkl`
- `backend/data/random_forest_btc.pkl`
- `backend/data/lstm_btc.pt`
- `backend/data/tft_btc/` (katalog)

Skrypty (minimum co robią)

- `models/linear_regression_train.py`
  - pipeline: `StandardScaler -> (LinearRegression|Ridge|Lasso)`
  - target: pct_change + shift(-1)
  - split: chronologiczny
  - zapis: `linear_regression_btc.pkl` (model + features + target)
- `models/random_forest_train.py`
  - feature engineering: log-zwroty, EWM, rolling std, agregaty volume/trades, lagi makro
  - tryb: grid albo single
  - zapis: `random_forest_btc.pkl` (model + features + target)
- `models/lstm_train.py`
  - sekwencje `(N, lookback, n_features)`, skalery zapisane w artefakcie
  - zapis: `lstm_btc.pt` (cfg + scaler + state_dict + metryki)
- `models/tft_train.py`
  - Darts TFT: `target=ret_btc_next`, `past_covariates` + time features
  - zapis: `tft_btc/` (metadata + scalers + model)
- `models/shap_analysis.py`
  - SHAP dla RF, wykresy do `models/shap/plots/`

Wymóg krytyczny (utrzymanie)

Feature engineering w treningu musi być identyczny jak w inferencji.
Zmiana FE → retraining + aktualizacja artefaktów w `backend/data/`.

Dodanie nowego modelu (checklista)

1. `models/<new>_train.py`: load parquet → FE + `ret_btc_next` → trening → zapis do `backend/data/`
2. Backend: metoda w `ModelService` + view w `marketdata/views.py` + route w `marketdata/urls.py`
3. (Opcjonalnie) raport/interpretowalność (SHAP/permutation importance)

4 4) Tests (pytest)

Lokalizacja: `backend/marketdata/tests/`
Typy: unit (serwisy) + lekkie API (views). Integracje zewnętrzne izolowane przez mocki (`requests.get`, `pd.read_parquet`, itd.).

Struktura

- `test_binance_service.py`
- `test_coinmetrics_service.py`
- `test_stooq_service.py`
- `test_fred_service.py`
- `test_data_reader_service.py`
- `test_historical_data_service.py`
- `test_model_service.py`
- `test_views.py`

Uruchamianie (lokalnie)

```
cd backend
pip install -r requirements.txt
pytest -q
```

Uruchamianie (Docker)

```
docker compose exec <backend_service_name> pytest -q
```

Zakres (co jest sprawdzane)

Serwisy zewnętrzne

- sukces (200) → poprawne parsowanie
- błąd → propagacja przez `raise_for_status()` / `HTTPError`
- FRED: brak `FRED_API_KEY` → `ValueError`, puste obserwacje → `N/A`

Parquet / odświeżanie

- logika "starości"pliku (`_is_data_old`)
- auto-refresh gdy plik nieaktualny
- filtrowanie do `years_back`

Generator danych

- `_apply_exclusions` (bez naruszania `open_time`)
- mapowanie Stooq → spójne nazwy kolumn
- batch fetching (mock `_fetch`)
- zapis `.parquet` do `backend/data`

ModelService

- `naive()` zwraca ostatni `close_btc`
- zachowanie dla pustych danych → `ValueError`
- LR predykcja na ostatnim wierszu (najświeższy rekord)
- obecność krytycznych kolumn dla RF (`_rf_base_columns()`)

Views (API)

- `/marketdata/binance-test/` - 200 + payload przy mocku
- `/marketdata/historical-data/` - walidacja `years_back` (400 dla nieliczbowej)
- `/marketdata/get-historical-data/` - `metrics` wymagane (400 przy braku)
- `/marketdata/naive-model/` - 200 baseline
- `/marketdata/linear-regression-model/` - obsługa braku pliku modelu (500 + komunikat)

Techniki izolacji

- `mocker.patch("requests.get", ...)`
- `mocker.patch("pandas.read_parquet", ...)`
- `tmp_path` dla testów zapisu plików
- kontrola czasu/`mtime` (symulacja "starości"danych)

Wymagania środowiska testowego

- wymagane biblioteki: `djangorestframework`, `pytest`
- praktycznie: `rest_framework` w `INSTALLED_APPS`
- zapis/odczyt parquet: działający backend parquet (np. `pyarrow`)