Trading App v5 Documentation

1. Project Overview
This document describes the full architecture and implementation of the Trading App v5. It covers data ingestion, model training, prediction, strategy execution, and front-end integration.

2. Data Utilities
- **Location**: `backend/data_utils.py`
- **Function**: `load_stock_data(symbol)`
  Reads `backend/data/{symbol}.csv` and returns a list of close prices.

3. CNN-LSTM Model
- **Location**: `backend/models/cnn_lstm_model.py`
- **Class**: `CNNLSTMModel`
  - `__init__(self, day, symbol)`
  - `build_model()`: Conv2D -> Reshape -> LSTM -> Dense
  - `load_data(self, symbol)`: CSV read, drop Date/Adj Close, MinMaxScaler, sliding windows, train_test_split, reshape to (N, day, 1, 5)
  - `train(self, symbol, epochs, batch_size)`: loads data, checkpoint, fit
  - `predict(self, X)`: model.predict
  - `get_status(self)`: computes MAE, RMSE, R^2
  - `save_model()`, `load_model()`

```python
from sklearn.preprocessing import MinMaxScaler
...
df = pd.read_csv(csv_path)
scaled = scaler.fit_transform(df[['Open','High','Low','Close','Volume']])
...
```

4. Backend Core
- **File**: `backend/core.py`
- **Endpoint**: `/train_model`
```python
def train_model(params):
    symbol = params['stock']
    time_window = params['time_window']
    cnn_model = CNNLSTMModel(day=time_window, symbol=symbol)
    cnn_model.train(symbol, epochs, batch_size)
    return {"status": cnn_model.get_status()}
```

- **Endpoint**: `/run_simulation`
  - Loads historical data, calls model predictions, applies strategy, returns JSON & saves to `backend/results/`.

5. Strategy Execution
- **Function**: `simulate_series(prices, initial_money, strategy_name)`
  - Uses `MyCustomAgent` to generate buy/sell indices.

  - Tracks cash, inventory, portfolio value.
  - Calculates metrics: final_balance, profit_pct, Sharpe ratio, max drawdown, win rate.
  - Returns trades list and portfolio_values.

6. Frontend (Streamlit)
- **File**: `app.py`
- **Pages**:
  - Trading Dashboard
  - Training & Forecast
  - Strategy Execution
  - Performance Analytics
- **Strategy Execution** calls `/run_simulation` and displays metrics:

```python
if st.button("Run Simulation"):
    res = requests.post(f"{API_URL}/run_simulation", json=payload).json()
    hist = res['historical']
    pred = res['predicted']
    # display charts and tables
```

7. Usage Instructions
1. `pip install -r requirements.txt`
2. Place CSV files under `backend/data/` named `{symbol}.csv`
3. Run backend: `python backend/core.py`
4. Run UI: `streamlit run app.py`