CryptoPredict

AI-Driven Cryptocurrency Analysis & Prediction Platform

Developed for the Revolut Challenge — HackUPC 2025

Murad Hüseynov **Giorgia Barboni**

March 6, 2026

Abstract

CryptoPredict is a full-stack, AI-powered web application for real-time cryptocurrency market analysis, price prediction, and gamified user engagement. The platform fuses deep learning (LSTM networks) for time-series price forecasting with state-of-the-art NLP (FinBERT) for financial sentiment extraction from live news feeds. A gamified prediction quiz challenges users to estimate hidden live market prices, awarding points based on accuracy and ranking participants on a persistent competitive leaderboard. The system is built on a decoupled React/Flask architecture with live data integration from the Binance and NewsAPI ecosystems. This documentation provides an exhaustive technical reference covering the system architecture, data pipeline, machine learning methodology, application mechanics, API specification, security model, and deployment procedures. This project was developed as a submission for the Revolut challenge at the HackUPC 2025 Hackathon.

Contents

1 Introduction

The cryptocurrency ecosystem is characterized by extreme volatility, 24/7 market operation, and rapid information propagation through social media and news outlets. Traditional technical analysis—relying solely on historical price patterns—often fails to capture the emotional and macroeconomic factors that drive sudden price swings. Furthermore, retail investors frequently lack the quantitative tools available to institutional trading desks.

CryptoPredict bridges this gap by providing:

1. **Multi-modal Analysis:** Simultaneous processing of numerical OHLCV (Open-High-Low-Close-Volume) time-series data and unstructured textual news data.
2. **Deep Learning Forecasting:** An LSTM-based neural network trained on engineered features including sentiment scores, rolling returns, volatility metrics, and cyclical time encodings.
3. **Real-time Data Feeds:** Live integration with the Binance spot market API for price data and NewsAPI for global cryptocurrency headlines.
4. **Gamified Learning:** An interactive prediction quiz where users test their market intuition against live prices, with scored results and a public leaderboard.
5. **Premium User Experience:** A responsive, glassmorphic dark-themed interface built with React and Material UI, featuring interactive Recharts visualizations.

The five tracked assets are: Bitcoin (BTC), Ethereum (ETH), Binance Coin (BNB), Ripple (XRP), and Solana (SOL).

2 System Architecture

CryptoPredict employs a modern, decoupled client–server architecture. The frontend and backend communicate exclusively through a well-defined RESTful JSON API, enabling independent scaling and development.

2.1 High-Level Architecture

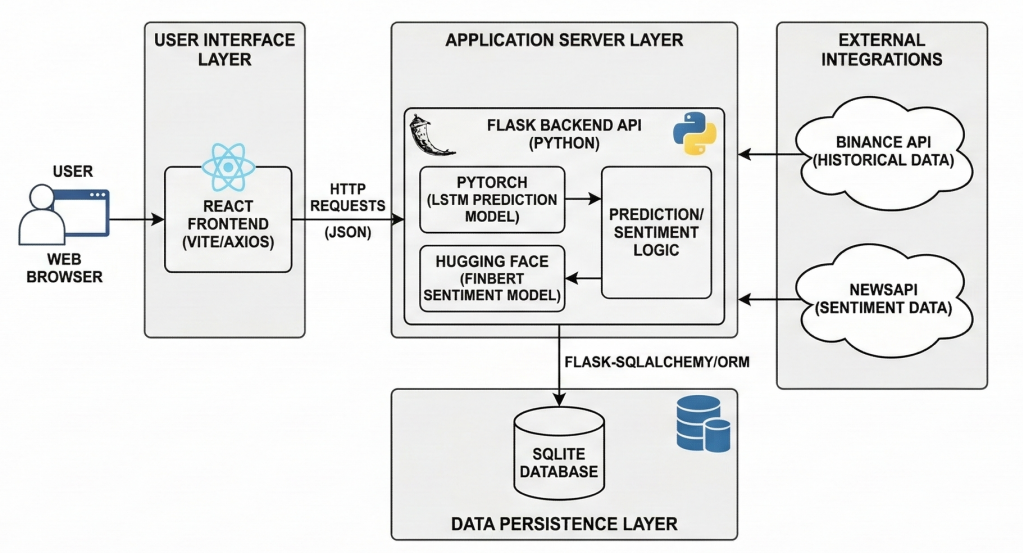

Figure 1: High-level system architecture showing the React frontend, Flask API, SQLite database, Binance API, and NewsAPI integrations.

2.2 Frontend Layer

- **Framework:** React 18 with functional components and Hooks (`useState`, `useEffect`, `useMemo`, `useCallback`, `useRef`, `useContext`).
- **UI Library:** Material UI v5 with a custom dark theme defined in `theme.js`. Component-level style overrides ensure visual consistency across Buttons, TextFields, Papers, and Cards.
- **Charting:** Recharts v2 for SVG-based rendering of line charts, area charts, bar charts, and custom reference elements (dots, lines).
- **Data Grid:** MUI X Data Grid for tabular OHLCV data display with sortable, styled columns.
- **Routing:** React Router v6 for client-side navigation between Home, Dashboard, Quiz, Leaderboard, Login, and Register pages.
- **State Management:** React Context API (`AuthContext`) for global authentication state, persisted to `localStorage`.
- **HTTP Client:** Axios with a proxy configuration (`"proxy": "http://localhost:5000"`) to forward API requests to the Flask backend during development.

2.3 Backend Layer

- **Framework:** Flask with Blueprints for modular route organization.
- **Database:** SQLite via Flask-SQLAlchemy ORM. Two models: `User` (id, username, password, points, created_at) and `Prediction` (id, user_id, crypto_symbol, predicted_price, actual_price, prediction_time, points_earned, created_at).

- **CORS:** Flask-CORS configured for `/api/*` endpoints.
- **Environment Management:** `python-dotenv` loads API keys from a `.env` file at startup via `load_dotenv()` in `app/__init__.py`.

2.4 Machine Learning Layer

- **PyTorch:** Powers the LSTM regression model (`LSTMRegressor`) defined in `LSTM_model.py`.
- **HuggingFace Transformers:** Utilizes the `yiyanghkust/finbert-tone` pre-trained model for financial sentiment classification.
- **Scikit-learn:** Gradient Boosting Regressor used during the data pipeline verification phase.

2.5 External API Integrations

Service	Purpose	Endpoints Used
Binance API	Live OHLCV data, current prices	`get_klines()`, `get_symbol_ticker()`, `get_ticker()`
NewsAPI	Global crypto news headlines	`get_everything()` with crypto keywords

Table 1: External API integrations and their roles within the platform.

3 Data Pipeline and Methodology

The machine learning subsystem operates through a multi-stage offline data pipeline, producing trained model artifacts that the live application loads at startup.

3.1 Stage 1: Data Ingestion (`scripts/build_dataset.py`)

The ingestion script orchestrates parallel data collection:

1. **Price Data:** For each of the five tracked coins, the script fetches hourly OHLCV candlestick data from the Binance `get_klines` endpoint. Each candle provides: open time, open price, high, low, close, and volume.
2. **News Data:** Cryptocurrency-related headlines are fetched from NewsAPI using a compound query: `"cryptocurrency OR bitcoin OR ethereum OR bnb OR xrp OR solana"`. Articles are collected for overlapping time windows and temporally aligned with the price data.
3. **Output:** A unified CSV file (`crypto_news_price_ohlc_dataset.csv`) with columns: `coin`, `timestamp`, `headline`, `open`, `high`, `low`, `close`, `volume`.

3.2 Stage 2: Sentiment Engineering (`scripts/sentiment_scores.py`)

Every news headline is processed through the pre-trained FinBERT model (`yiyanghkust/finbert-tone`), a BERT variant fine-tuned on financial text. The model outputs a three-class probability distribution: *positive*, *neutral*, *negative*.

The final numerical sentiment score is computed as:

$$S_{\text{final}} = \text{confidence} \times \delta(\text{label}) \tag{1}$$

where:

$$\delta(\text{label}) = \begin{cases} +1 & \text{if label} = \text{positive} \\ 0 & \text{if label} = \text{neutral} \\ -1 & \text{if label} = \text{negative} \end{cases}$$

This produces a continuous value $S_{\text{final}} \in [-1, +1]$ representing the directional financial sentiment at each hourly timestamp.

3.3 Stage 3: Feature Engineering (`scripts/csv_preprocess.py`)

The following engineered features are computed per coin, grouped by timestamp:

Feature	Symbol	Description
Sentiment Score	S_{final}	FinBERT-derived directional sentiment
5-min Return	`ret_5m`	Percentage price change over 5 periods
15-min Return	`ret_15m`	Percentage price change over 15 periods
30-min Return	`ret_30m`	Percentage price change over 30 periods
5-min Volatility	`vol_5m`	Rolling standard deviation of volume (5-window)
15-min Volatility	`vol_15m`	Rolling standard deviation of volume (15-window)
30-min Volatility	`vol_30m`	Rolling standard deviation of volume (30-window)
Sine Hour	$\sin(\frac{2\pi h}{24})$	Cyclical encoding of hour-of-day
Cosine Hour	$\cos(\frac{2\pi h}{24})$	Cyclical encoding of hour-of-day

Table 2: The 9-dimensional feature vector used as input to the LSTM model.

The target variable is the **60-minute forward percentage return**:

$$y_t = \frac{P_{t+60} - P_t}{P_t} \tag{2}$$

3.4 Stage 4: Model Training (`scripts/data_split.py` and `LSTM_model.py`)

3.4.1 LSTM Architecture

The `LSTMRegressor` is a PyTorch `nn.Module` with:

- **Input dimension:** 9 (the feature vector from Table 2)
- **Hidden dimension:** 64 LSTM units
- **Sequence length:** 20 consecutive hourly windows

- **Output:** A single scalar predicting the 60-minute forward return
- **Regularization:** Dropout layers to prevent overfitting
- **Optimizer:** Adam with default learning rate
- **Loss function:** Mean Squared Error (MSE)

The model is trained on a chronological train/validation split and checkpointed as `best_lstm_model.pt`.

3.4.2 Gradient Boosting Baseline

A Scikit-learn `GradientBoostingRegressor` is also trained on the same feature set as a traditional ML baseline. This model is serialized as `gbr_model.pkl` and used for pipeline validation.

3.4.3 Inference at Runtime

At Flask startup, the trained LSTM model weights are loaded:

```
model = LSTMRegressor(input_dim=9, hidden_dim=64)
model.load_state_dict(
    torch.load('best_lstm_model.pt',
               map_location=torch.device('cpu'))
)
model.eval()
```

The `make_predictions()` function in `app/utils.py` preprocesses incoming data, constructs sliding-window sequences, and runs batch inference without gradient computation (`torch.no_grad()`).

4 Platform Mechanics and Features

4.1 User Authentication System

The platform implements a full registration and login system backed by the SQLite database.

4.1.1 Registration Flow

1. User submits a username and password via the React `Register.js` form.
2. The frontend sends `POST /api/register` with `{username, password}`.
3. The backend checks for duplicate usernames, creates a new `User` row, and returns the `user_id` and `username`.
4. The frontend stores `{user_id, username}` in `AuthContext` and `localStorage` for session persistence.

4.1.2 Login Flow

1. User submits credentials via the React `Login.js` form.
2. `POST /api/login` validates against the database.
3. On success, returns `user_id` and `username`; on failure, returns HTTP 401.

4.2 Real-time Analytical Dashboard

The core analytical dashboard delivers instantaneous market overviews and high-resolution charting for all five tracked assets.

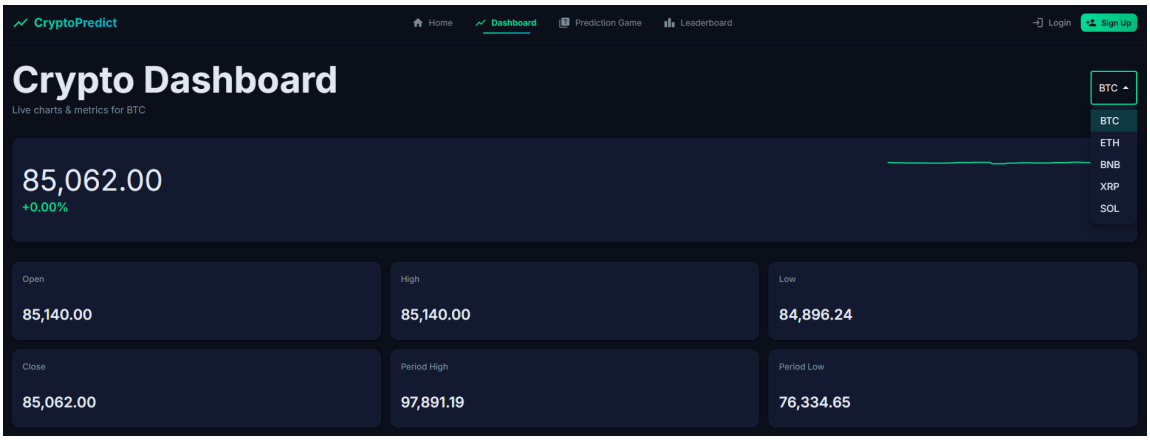

Figure 2: Interactive analytical dashboard (Part 1/5): Live price sparkline and OHLC metric cards.

Figure 2: Interactive analytical dashboard (Part 2/5): Gradient-filled price area chart.

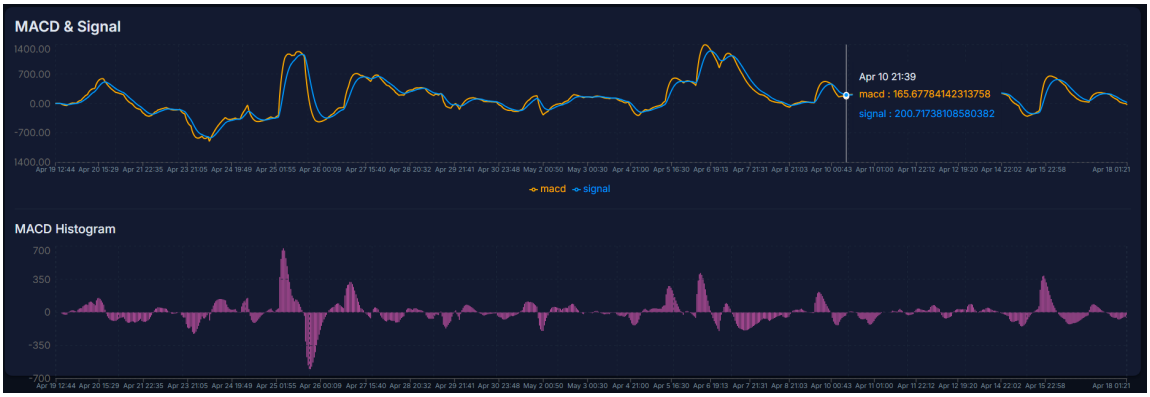

Figure 2: Interactive analytical dashboard (Part 3/5): MACD line chart and histogram.

Figure 2: Interactive analytical dashboard (Part 4/5): OHLCV data grid.

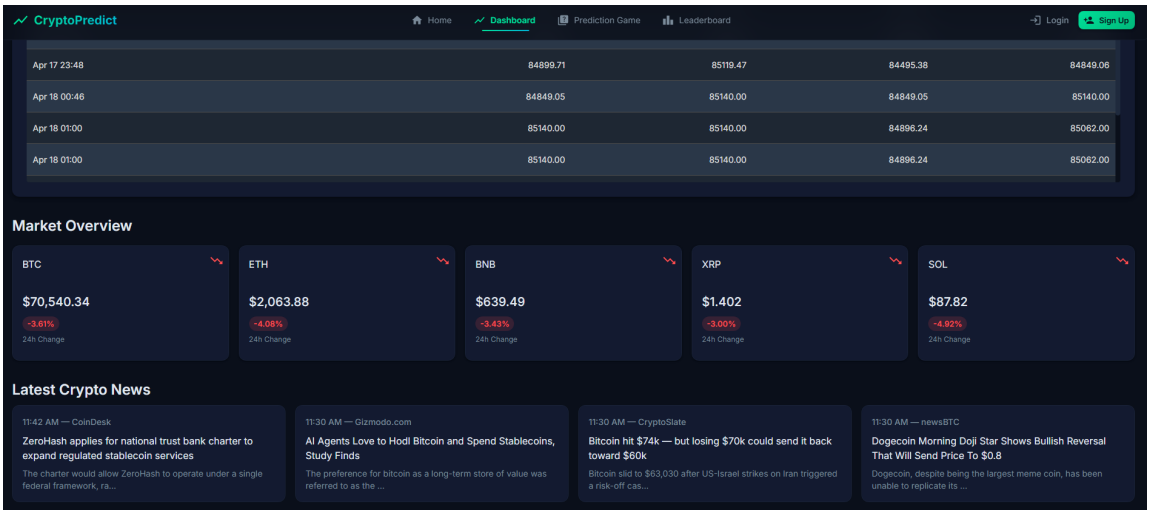

Figure 2: Interactive analytical dashboard (Part 5/5): Market overview cards and rotating news feed.

4.2.1 Dashboard Sections

1. **Price Sparkline & Summary:** Displays the latest close price, intra-period percentage change, and a compact 80-point line chart.

2. **Metric Cards:** Six cards showing Open, High, Low, Close, Period High, and Period Low.

3. **Price History Chart:** A full-width gradient-filled area chart (`AreaChart`) with formatted timestamps on the X-axis and locale-formatted prices on the Y-axis. Tooltips display hovering timestamps in `Mon DD HH:MM` format.
4. **MACD & Signal Chart:** A dual-line chart displaying the MACD line (12/26 EMA difference) and the 9-period Signal line.
5. **MACD Histogram:** A bar chart visualizing the divergence between the MACD and Signal lines, colored in magenta.
6. **Last 10 Data Points:** A styled `DataGrid` table showing recent OHLC values with formatted timestamps and precision-rounded prices.
7. **Market Overview:** Five cards (one per coin) fetched from `GET /api/market-overview`, each showing the live price and 24h percentage change with directional trend icons and color-coded chips.
8. **Latest Crypto News:** A rotating carousel of 4 news cards (auto-advancing every 7 seconds) fetched from `GET /api/news`. Each card displays the publication time, source name, headline, and description snippet.

4.2.2 Technical Indicator: MACD

The Moving Average Convergence Divergence is computed server-side in `app/utils.py`:

$$\text{EMA}_{12} = \text{close}.ewm(\text{span} = 12) \tag{3}$$
$$\text{EMA}_{26} = \text{close}.ewm(\text{span} = 26) \tag{4}$$
$$\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26} \tag{5}$$
$$\text{Signal} = \text{MACD}.ewm(\text{span} = 9) \tag{6}$$
$$\text{Histogram} = \text{MACD} - \text{Signal} \tag{7}$$

4.3 The Prediction Game

The prediction game is the platform's core interactive feature, challenging users to estimate the current live market price from a partially obscured historical chart.

Figure 3: The prediction game interface. The chart shows recent price history with the last 6 hours hidden. The user clicks in the prediction zone (right of the red cutoff line) to place their guess.

4.3.1 Game Flow — Step by Step

1. **Data Loading:** When the user enters the quiz page, the frontend sends `GET /api/quiz-data/<symbol>` to the Flask backend.

2. **Server-Side Data Preparation:** The backend calls `binance_client.get_klines()` to fetch the most recent 200 hourly candlesticks for the selected asset. It then **hides the last 6 candles** (representing approximately 6 hours of price history) and returns only the visible portion:

```
hidden_count = 6
visible = candles[:-hidden_count]
cutoff_ts = visible[-1]['timestamp']
```

3. **Chart Rendering:** The React component renders the visible candles as a line chart. The X-axis extends 15% beyond the last visible data point to create a blank "prediction zone." A vertical red dashed reference line marks the cutoff boundary. Timestamps are formatted as `Mon DD HH:MM`.

4. **Prediction Placement:** The user clicks anywhere in the prediction zone (to the right of the cutoff). The component captures the pixel coordinates relative to the chart's Cartesian plane and maps them to actual timestamp (X) and price (Y) values:

```
const ts = xMin + ((offsetX - marginLeft) / plotW)
           * (xMax - xMin);
const price = yMax - ((offsetY - marginTop) / plotH)
              * (yMax - yMin);
```

   A blue reference dot is rendered at the predicted coordinates.

5. **Submission:** The user clicks "Submit Prediction." The frontend sends:

```
POST /api/predict
{
    "user_id": <integer>,
    "crypto_symbol": "BTC",
    "predicted_price": 87432.50
}
```

6. **Server-Side Scoring:** The backend fetches the *current live* Binance price (`get_current_price()`) and computes the score:

$$\text{accuracy} = \frac{|P_{\text{predicted}} - P_{\text{actual}}|}{P_{\text{actual}}} \tag{8}$$

$$\text{Points} = \max(0,\ \min(100,\ \lfloor(1 - \text{accuracy}) \times 100\rfloor)) \tag{9}$$

   The prediction and points are persisted to the `Prediction` table, and the user's cumulative `points` column in the `User` table is incremented.

7. **Result Display:** The frontend receives the actual price and points earned. The chart is **locked** (click events are disabled), an orange horizontal reference line is drawn at the actual price level, and a detailed result alert is shown.

8. **New Round:** The user clicks "New Round" to fetch fresh data from Binance and play again.

4.3.2 Anti-Cheat Mechanisms

- The hidden data is **never sent to the client**—the backend truncates the candle array server-side.
- After submission, the chart wrapper rejects all click events (`if (locked) return`).
- The crypto selector is disabled while a round is locked.
- The "Submit" button is replaced by "New Round" after submission, preventing duplicate scoring.

4.4 Competitive Leaderboard

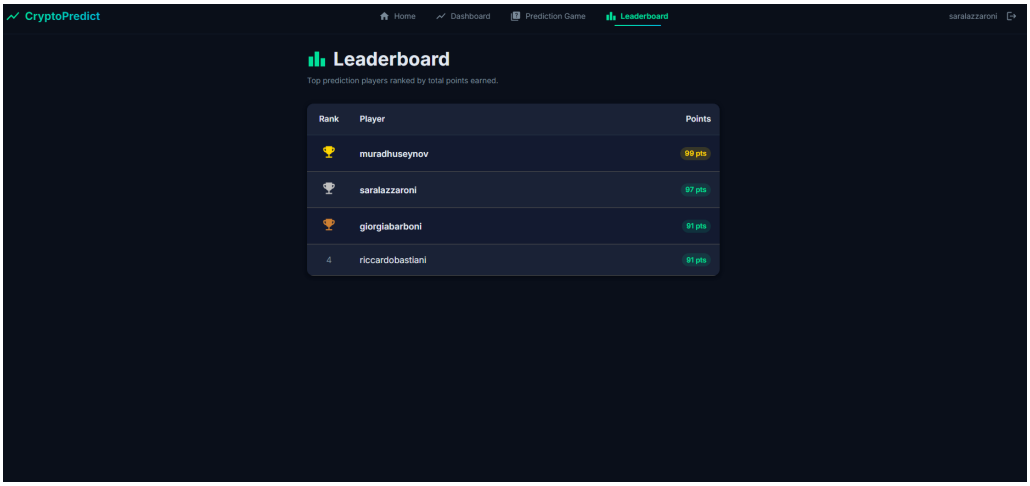

Figure 4: The leaderboard page displaying the top 10 players ranked by cumulative prediction points. The top 3 are distinguished with gold, silver, and bronze medal icons.

Mechanics: The `GET /api/leaderboard` endpoint queries the `User` table, orders by descending `points`, and returns the top 10 entries with rank, username, and cumulative points. The React component displays these in a styled table with medal icons (gold/silver/bronze) for the top three positions.

5 API Reference

The following table documents all RESTful API endpoints exposed by the Flask backend:

Method	Endpoint	Description
POST	`/api/register`	Register a new user. Body: `{username, password}`. Returns `user_id`.
POST	`/api/login`	Authenticate a user. Body: `{username, password}`. Returns `user_id`.
POST	`/api/predict`	Submit a prediction. Body: `{user_id, crypto_symbol, predicted_price}`. Returns actual price and points.
GET	`/api/leaderboard`	Top 10 users ranked by cumulative points.
GET	`/api/quiz-data/<sym>`	200 hourly candles from Binance with last 6h hidden.
GET	`/api/charts/<sym>`	Historical OHLCV + MACD data from the offline CSV dataset.
GET	`/api/market-overview`	24h price and percentage change for all 5 coins from Binance.
GET	`/api/news`	Latest 20 crypto headlines from NewsAPI (last 24h).

Table 3: Complete API endpoint reference.

6 Security Model

6.1 API Key Management

All external API credentials are stored in a `.env` file at the project root. The keys are loaded at Flask startup using `python-dotenv`:

```
# app/__init__.py
from dotenv import load_dotenv
load_dotenv()

# app/utils.py
binance_client = Client(
    os.getenv('BINANCE_API_KEY'),
    os.getenv('BINANCE_API_SECRET')
)
newsapi = NewsApiClient(
    api_key=os.getenv('NEWSAPI_KEY')
)
```

A `.env.example` template is provided for developer onboarding:

```
BINANCE_API_KEY=your_binance_api_key_here
BINANCE_API_SECRET=your_binance_api_secret_here
NEWSAPI_KEY=your_newsapi_key_here
COINLAYER_API_KEY=your_coinlayer_key_here
```

6.2 Git Security

The `.gitignore` file prevents accidental commits of:

- `.env` — API keys and secrets
- `instance/`, `*.db` — SQLite database files
- `*.pt`, `*.pkl` — trained model artifacts
- `*.csv` — raw and processed datasets
- `__pycache__/`, `node_modules/` — dependency caches
- `frontend/build/` — production build output

7 Project Structure

```
Crypto_Prediction_Application/
|-- app/
|   |-- __init__.py          # Flask app factory, dotenv loading
|   |-- models.py            # SQLAlchemy models (User, Prediction)
|   |-- routes.py            # All API endpoint handlers
|   |-- utils.py             # LSTM inference, MACD, Binance/News
   helpers
|-- frontend/
|   |-- public/index.html    # Entry HTML with SEO meta tags
|   |-- src/
```

```
|       |-- App.js           # Router + ThemeProvider wrapper
|       |-- theme.js         # Shared MUI dark theme configuration
|       |-- context/
|       |   |-- AuthContext.js  # Global auth state (React Context)
|       |-- components/
|           |-- Home.js          # Landing page with feature cards
|           |-- Navbar.js        # Navigation bar with glassmorphism
|           |-- Login.js         # Authentication form
|           |-- Register.js      # Registration form
|           |-- Charts.js        # Full analytical dashboard
|           |-- QuizPage.js      # Prediction game interface
|           |-- Leaderboard.js   # Ranked player table
|-- scripts/
|   |-- build_dataset.py    # Data ingestion (Binance + NewsAPI)
|   |-- csv_preprocess.py   # Feature engineering
|   |-- data_split.py       # Train/test split + GBR training
|   |-- sentiment_scores.py # FinBERT sentiment extraction
|-- LSTM_model.py           # LSTMRegressor class definition
|-- run.py                  # Flask entry point
|-- requirements.txt        # Python dependencies
|-- .env.example            # API key template
|-- .gitignore              # Security exclusions
|-- README.md               # GitHub-facing documentation
```

8 Setup and Execution

8.1 Prerequisites

- Python 3.8+ with `pip`
- Node.js v18+ with `npm`
- Valid API keys for Binance and NewsAPI (stored in `.env`)

8.2 Backend Setup

```
# 1. Create and activate virtual environment
$ python -m venv .venv
$ source .venv/bin/activate       # Linux/macOS
$ .venv\Scripts\activate          # Windows

# 2. Install Python dependencies
$ pip install -r requirements.txt

# 3. Configure API keys
$ cp .env.example .env
# Edit .env with your actual API keys

# 4. Start the Flask server
$ python run.py
# Server runs on http://localhost:5000
```

8.3 Frontend Setup

```
# 1. Install Node.js dependencies
$ cd frontend
$ npm install

# 2. Start the development server
$ npm start
# App opens at http://localhost:3000
```

8.4 Data Pipeline Execution (Optional)

To retrain the models from scratch:

```
$ python scripts/build_dataset.py      # Fetch data
$ python scripts/sentiment_scores.py   # Compute sentiments
$ python scripts/csv_preprocess.py     # Engineer features
$ python scripts/data_split.py         # Train GBR baseline
$ python LSTM_model.py                 # Train LSTM model
```

9 Technology Stack

Category	Technology	Role
Frontend	React 18	Component-based UI framework
	Material UI 5	Design system with dark theme
	Recharts 2	SVG-based data visualization
	MUI X Data Grid	Tabular data display
	Axios	HTTP client for API communication
	React Router 6	Client-side page routing
Backend	Flask	Lightweight Python web framework
	Flask-SQLAlchemy	ORM for SQLite database
	Flask-CORS	Cross-origin request handling
	python-dotenv	Environment variable management
ML / AI	PyTorch	Deep learning framework (LSTM)
	HuggingFace Transformers	FinBERT sentiment analysis
	Scikit-learn	Gradient Boosting baseline
	Pandas / NumPy	Data manipulation and computation
APIs	Binance API	Live OHLCV data and current prices
	NewsAPI	Global crypto news aggregation
Database	SQLite	Embedded relational database

Table 4: Complete technology stack.

10 Conclusion

CryptoPredict demonstrates an end-to-end integration of financial data engineering, deep learning, natural language processing, and modern web development. The platform delivers:

- A production-grade analytical dashboard with real-time MACD indicators, live market overviews, and a rotating news feed.
- A novel gamified prediction game that leverages live Binance data with server-side data hiding and anti-cheat protections.
- A persistent competitive ecosystem through the leaderboard, encouraging repeated engagement and analytical skill development.
- A secure, well-architected codebase following industry best practices for secret management, code organization, and separation of concerns.

For the Revolut HackUPC 2025 challenge, CryptoPredict delivers a sophisticated yet accessible platform that bridges the gap between institutional-grade market analysis tools and retail cryptocurrency enthusiasts.

Project Authors: Murad Hüseynov, Giorgia Barboni
Event: HackUPC 2025 (Barcelona, Spain) — Revolut Challenge
Repository: `Crypto_Prediction_Application`