FAIN

Intelligent Trading Assistant for the Tunisian Stock Exchange (BVMT)

Technical Report

IHEC CODELAB 2.0 Hackathon

Team:	Makarouna Kadheba
Date:	February 2026
Repository:	GitHub
Live Demo:	App
License:	MIT

This document provides a comprehensive technical overview of the FAIN platform, covering its microservice architecture, AI/ML pipelines, API specifications, and deployment procedures.

Contents

Chapter 1

Introduction and Project Overview

1.1 Project Context

FAIN is an intelligent trading assistant developed for the **IHEC CODELAB 2.0 Hackathon**. The platform targets the **Bourse des Valeurs Mobilières de Tunis (BVMT)** — the Tunisian Stock Exchange — and provides an integrated suite of AI-powered tools for traders and investors.

The system combines five core AI/ML capabilities:

1. **Price Prediction** — 5-day forecasting using XGBoost and RandomForest models
2. **Sentiment Analysis** — Multilingual (French, Arabic, Tunisian dialect) news and social media analysis
3. **Anomaly Detection** — Statistical and ML-based detection of market irregularities
4. **Portfolio Management** — Reinforcement Learning (PPO) portfolio optimization with explainability
5. **RAG Chatbot** — Context-grounded Q&A assistant powered by ChromaDB and Llama 3.3 70B

1.2 High-Level Architecture

The platform follows a **microservice architecture** pattern with **10 independently deployable services**, each functional domain encapsulated in its own service. The key architectural decisions are:

Table 1.1: Technology Stack Overview

Layer	Technology
Frontend	Next.js 16 with TypeScript and Tailwind CSS
Backend Core	NestJS 11 (TypeScript) — API orchestration + JWT auth
ML Services	FastAPI (Python) — Forecasting, Anomaly Detection, Sentiment, Portfolio
Chatbot	FastAPI + ChromaDB + Llama 3.3 70B (RAG pipeline)
Database	PostgreSQL 16 with TimescaleDB extension
Vector Store	ChromaDB (all-MiniLM-L6-v2 embeddings)
Cache	Redis (optional)
Deployment	Docker Compose

1.3 Project Structure

The repository is organized as a monorepo with clear separation of concerns:

```
.
|-- backend/
|   |-- core/                # NestJS API orchestration layer
|   |   |-- src/
|   |       |-- auth/        # JWT authentication (login, signup, guards)
|   |       |-- market/      # Market data aggregation
|   |       |-- forecast/    # Price prediction proxy
|   |       |-- anomaly/     # Anomaly detection proxy
|   |       |-- portfolio/   # Portfolio optimization proxy
|   |-- services/             # Python microservices
|   |   |-- api_gateway/             # FastAPI gateway (port 8000)
|   |   |-- stock_service/           # Stock data CRUD (port 8001)
|   |   |-- market_service/          # Market analytics (port 8002)
|   |   |-- notification_service/    # Email & alerts (port 8003)
|   |   |-- anomaly_detection/       # Anomaly detection (port 8004)
|   |   |-- sentiment-analysis/      # Sentiment + Tunizi NLP (port 8005)
|   |   |-- chatbot/                 # RAG chatbot (port 8004)
|   |   |-- forecasting/             # Price forecasting (XGBoost)
|   |   |-- portfolio_management_service/ # RL portfolio optimizer (PPO)
|   |   |-- jobs_service/            # Background job scheduler
|   |-- shared/              # Shared Python modules
|   |-- scripts/             # Utility scripts
|-- ui/                      # Next.js 16 frontend
|   |-- app/                 # App Router pages
|   |   |-- analyse/         # Market analysis view
|   |   |-- prevision/       # Price prediction view
|   |   |-- sentiment/       # Sentiment heatmap & social media
|   |   |-- surveillance/    # Anomaly monitoring view
|   |   |-- portefeuille/    # Portfolio management view
|   |   |-- login/           # Authentication
|   |   |-- signup/          # Registration
|   |-- components/          # React components
|   |-- lib/                 # API client, auth context, config
|-- data/                    # Historical BVMT data (2016--2026)
|   |-- data engineering/    # ETL scripts (CSV/XLS -> PostgreSQL)
|-- database/                # SQL initialization scripts
|-- notebooks/               # Jupyter notebooks
|-- docs/                    # Documentation
```

Listing 1.1: Top-Level Directory Structure

1.4 Prerequisites and Quick Start

System Requirements

- Docker and Docker Compose
- 8 GB RAM minimum
- 20 GB free disk space

```
# Clone the repository
git clone
   https://github.com/Moetez-Fradi/IHEC-Code-Lab-Makarouna-Kadheba.git
cd IHEC-Code-Lab-Makarouna-Kadheba

# Configure environment
cp backend/.env.example backend/.env

# Start all services
docker-compose up -d

# Load historical data
docker-compose exec backend python scripts/load_data.py
```

Listing 1.2: Quick Start Commands

1.5 Access Points

Table 1.2: Service Access Points

Service	URL	Credentials
Frontend Dashboard	`http://localhost:3000`	—
Backend API (NestJS)	`http://localhost:8000/api`	JWT required
PostgreSQL	`localhost:5432`	postgres / postgres
Chatbot Service	`http://localhost:8004`	—

1.6 Data Flow

The system follows a layered communication pattern:

```
User -> Next.js Frontend (:3000)
          -> NestJS Backend Core (:8000/api)
               -> Python Microservices (:8001--:8007)
                    -> PostgreSQL / External APIs / LLMs
```

Listing 1.3: Service Communication Flow

Chapter 2

Backend Core — NestJS Orchestration Layer

2.1 Overview

The backend core is built with **NestJS 11**, a progressive Node.js framework for building efficient and scalable server-side applications. It serves as the **API orchestration layer**, routing incoming requests from the frontend to the appropriate Python microservices.

2.2 Technology Stack

- **Runtime:** Node.js
- **Framework:** NestJS (TypeScript)
- **Package Manager:** pnpm
- **Build System:** Nest CLI
- **Testing:** Jest (unit and e2e)

2.3 Development Commands

```
cd backend/core

# Install dependencies
pnpm install

# Development (watch mode)
pnpm run start:dev

# Production build
pnpm run start:prod

# Unit tests
pnpm run test

# End-to-end tests
pnpm run test:e2e

# Test coverage
pnpm run test:cov
```

Listing 2.1: NestJS Development Commands

2.4 Module Architecture

The NestJS core includes the following functional modules:

- **Auth Module** — JWT-based authentication and authorization
- **Market Module** — Market data aggregation and routing
- **Forecast Module** — Price prediction proxy
- **Anomaly Module** — Anomaly detection proxy
- **Portfolio Module** — Portfolio optimization proxy

Each module follows the standard NestJS pattern with controllers, services, and DTOs, acting as a typed proxy to the underlying Python microservices.

2.4.1 Authentication Module

The Auth Module provides a complete **JWT-based authentication system**:

- **User Entity** — TypeORM entity stored in PostgreSQL (`user.entity.ts`)
- **JWT Strategy** — Passport.js JWT strategy for token validation (`jwt.strategy.ts`)
- **Auth Guard** — Route protection via `JwtAuthGuard` (`jwt-auth.guard.ts`)
- **DTOs** — Input validation with class-validator (`auth.dto.ts`)

The module exposes `/api/auth/login`, `/api/auth/signup`, and `/api/auth/me` endpoints, returning JWTs that protect all other routes in the system.

2.4.2 Portfolio Module

The Portfolio Module proxies requests to the Portfolio Management Python microservice, exposing endpoints for:

- `/api/portfolio/recommend` — Portfolio allocation recommendations
- `/api/portfolio/simulate` — Historical simulation with user capital
- `/api/portfolio/stress-test` — Stress testing scenarios
- `/api/portfolio/snapshot` — Current portfolio state
- `/api/portfolio/macro` — Macroeconomic data
- `/api/portfolio/train` — RL agent training trigger

Chapter 3

API Gateway

3.1 Overview

The API Gateway serves as the **main entry point** for all client API requests. It is a lightweight FastAPI application that routes incoming requests to the appropriate downstream microservices.

3.2 API Endpoints

3.2.1 Health Check

Method	Endpoint	Description
GET	`/health`	Service health check

3.2.2 Authentication Endpoints

All routes (except auth) require a JWT token via `Authorization: Bearer <token>`.

Method	Endpoint	Description
POST	`/api/auth/login`	Login → JWT token
POST	`/api/auth/signup`	Register new user
GET	`/api/auth/me`	Get current user profile

3.2.3 Market Data Endpoints

Method	Endpoint	Description
GET	`/api/market/overview`	Market overview with TUNINDEX
GET	`/api/market/stocks`	List all BVMT stocks
GET	`/api/market/history/{code}`	Historical prices for a stock
GET	`/api/market/latest`	Latest market data

3.2.4 Forecasting Endpoints

Method	Endpoint	Description
GET	`/api/forecast?code={ISIN}`	5-day price forecast

3.2.5 Anomaly Detection Endpoints

Method	Endpoint	Description
GET	`/api/anomalies?code={code}&start={date}&end={date}`	Detect anomalies in date range

3.2.6 Sentiment Analysis Endpoints

Method	Endpoint	Description
GET	`/api/sentiment/sentiments/daily`	Daily sentiment scores per ticker
GET	`/api/sentiment/articles`	List analysed articles (filterable)
POST	`/api/sentiment/scrape`	Trigger scraping pipeline
POST	`/api/sentiment/analyze-tunizi`	**K.O.:** Tunizi text analysis
POST	`/api/sentiment/search-social-media`	**K.O.:** Social media search

3.2.7 Portfolio Management Endpoints

Method	Endpoint	Description
POST	`/api/portfolio/recommend`	Portfolio allocation + explanation
POST	`/api/portfolio/simulate`	Historical simulation with capital
POST	`/api/portfolio/stress-test`	Stress test scenarios
GET	`/api/portfolio/snapshot`	Current portfolio state
GET	`/api/portfolio/macro`	Macroeconomic indicators
POST	`/api/portfolio/train`	Train RL agent

3.2.8 Chatbot Endpoints

Method	Endpoint	Description
POST	`/chat`	Send message → AI response (port 8004)
DELETE	`/chat/{session_id}`	Clear conversation history

3.3 Setup and Execution

```
cd backend/services/api_gateway
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Run the gateway
uvicorn app:app --reload --port 8000
```

Listing 3.1: API Gateway Setup

Alternatively, all Python microservices can be started at once:

```
cd backend/services
bash start_all.sh
```

Listing 3.2: Start All Services

Chapter 4

Forecasting Service

4.1 Overview

The BVMT Forecasting Service provides **5-day price, volume, and liquidity forecasts** for securities listed on the Bourse de Tunis. It uses an **on-the-fly training approach**, ensuring that models are always trained on the most up-to-date historical data.

4.2 Key Features

- **Price Forecasting** — Predicts closing prices for the next 5 business days
- **Confidence Intervals** — Provides upper and lower bounds for predictions
- **Volume Forecasting** — Predicts expected trading volume per day
- **Liquidity Classification** — Classifies expected liquidity as "high" or "low" with probability
- **On-the-Fly Training** — Models are trained in real-time using the latest available data

4.3 Technology Stack

Component	Technology
Backend	Python, FastAPI
ML Models	XGBoost, Scikit-learn
Data	Pandas, NumPy
Database	PostgreSQL (via `asyncpg`)
Server	Uvicorn

4.4 Machine Learning Models

4.4.1 Price and Volume Forecasting

The service uses `xgboost.XGBRegressor` to predict closing prices and volumes for each of the next 5 trading days. A **separate model is trained for each day** in the forecast horizon, ensuring day-specific accuracy.

4.4.2 Liquidity Classification

A `sklearn.ensemble.RandomForestClassifier` predicts the probability of high or low liquidity. Liquidity is defined as "high" if the daily volume exceeds its 20-day rolling median.

4.4.3 Feature Engineering

A rich set of over **40 features** is generated from raw OHLCV data:

- Lagged prices and returns
- Moving averages (MA, EMA)
- Technical indicators (MACD, RSI, Bollinger Bands)
- Volatility measures
- Calendar features (day of week, month)

4.5 API Endpoints

Table 4.1: Forecasting Service Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
GET	`/forecast`	Generate 5-day forecast for a stock
GET	`/config`	Current service configuration

4.5.1 Forecast Endpoint Parameters

Parameter	Type	Required	Description
`code`	string	Yes	Stock ISIN code (e.g., `TN0001100254`)
`lookback`	integer	No	Historical days for training (default: 500)

4.6 Configuration

Table 4.2: Environment Variables

Variable	Default	Description
PORT	8002	Service port
HOST	0.0.0.0	Bind host
DATABASE_URL	—	PostgreSQL connection string
FORECAST_HORIZON	5	Number of days to forecast
MIN_HISTORY_DAYS	100	Minimum historical data required
DEFAULT_LOOKBACK_DAYS	500	Default training window
XGB_N_ESTIMATORS	100	XGBoost ensemble size
XGB_MAX_DEPTH	4	XGBoost tree depth

Chapter 5

Anomaly Detection Service

5.1 Overview

The BVMT Anomaly Detection Service identifies **unusual trading patterns** in stock market data. It employs multiple statistical and machine learning techniques to flag anomalous trading days based on volume spikes, abnormal price changes, and irregular transaction patterns.

5.2 Key Features

- **Multi-method Anomaly Detection** — Analyzes volume, price changes, and transaction patterns
- **Volume Spike Detection** — Flags days with volume exceeding 3σ from the mean
- **Price Manipulation Identification** — Detects suspicious price movements
- **Isolation Forest** — ML-based pattern anomaly detection for irregular trading behaviour
- **CMF Regulatory Compliance** — Alerts for potential regulatory violations against Conseil du Marché Financier rules
- **Configurable Algorithms** — Thresholds, rolling windows, and model parameters are fully tunable

5.3 Technology Stack

Component	Technology
Backend	Python, FastAPI
Database	PostgreSQL (via `asyncpg`)
Data Science	Pandas, NumPy, Scikit-learn
Server	Uvicorn

5.4 API Endpoints

Table 5.1: Anomaly Detection Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
GET	`/anomalies`	Run anomaly detection pipeline
GET	`/config`	Current algorithm configuration

5.4.1 Anomalies Endpoint Parameters

Parameter	Type	Required	Description
`code`	string	Yes	Stock code (e.g., `PX1`)
`start`	string	Yes	Start date (`YYYY-MM-DD`)
`end`	string	Yes	End date (`YYYY-MM-DD`)

5.5 Configuration

Variable	Default	Description
`DATABASE_URL`	Neon.tech URL	PostgreSQL connection string
`HOST`	0.0.0.0	Bind host
`PORT`	8001	Service port

Algorithm-specific parameters (thresholds, rolling windows, etc.) are tunable in `config.py`.

5.6 Example Usage

```
curl
    "http://localhost:8001/anomalies?code=BIAT&start=2023-01-01&end=2023-03
```

Listing 5.1: Querying the Anomaly Detection Endpoint

The response includes per-day anomaly details and summary statistics.

Chapter 6

Sentiment Analysis Service

6.1 Overview

The BVMT Sentiment Analysis Module is a **multilingual sentiment analysis system** for the Tunisian Stock Exchange, built with FastAPI and integrated with **OpenRouter LLM** (DeepSeek R1 Chimera) for intelligent text analysis.

6.2 Key Features

- **Multilingual Support** — Handles French, Arabic, and Tunisian dialect (Arabizi) seamlessly
- **Real-time Scraping** — Monitors 3 Tunisian financial news sources
- **LLM-Powered Analysis** — Uses DeepSeek R1 Chimera via OpenRouter (free tier)
- **Ticker Detection** — Automatically identifies mentioned Tunisian companies
- **Daily Aggregation** — Computes sentiment scores per ticker
- **Markdown Reports** — Generates downloadable per-company sentiment reports
- **SQLite Storage** — Lightweight async database

6.3 K.O. Features — Competitive Differentiators

6.3.1 Tunizi/Arabizi NLP Engine

The platform includes a purpose-built **Tunisian dialect (Tunizi) sentiment processor** — a competitive differentiator that no other team has. This module (`tunizi.py`) provides:

- **Arabizi Normalization** — Converts Tunisian number-letter encoding to Latin equivalents (e.g., `3`→`aa`, `7`→`h`, `9`→`q`)
- **Financial Slang Dictionary** — 30+ Tunizi terms with sentiment weights (e.g., `tla3`="rising" → +0.7, `ti7`="drop" → −0.8)
- **Code-Switching Detection** — Identifies Arabic/French/Tunizi mixed-language posts
- **Entity Mapping** — Maps informal company nicknames to BVMT tickers (e.g., "la bière" → SFBT)

- **Hybrid Scoring** — Combines Tunizi dictionary scores (60% weight) with LLM scores (40% weight) for enhanced accuracy

Example: Tunizi Analysis

Input: `"SFBT bech ti7 2main"` → Bearish: "SFBT will drop tomorrow" (score: −0.70)
Input: `"La bière tla3et behi"` → Bullish: "The beer [SFBT] went up nicely" (score: +0.65)

6.3.2 Social Media Search via Perplexity

The platform uses **Perplexity AI's Sonar model** (`perplexity_search.py`) for real-time social media search, capturing retail investor sentiment from:

- **Twitter/X** — Tunisian finance community discussions
- **Reddit r/tunisia** — Investment and economic discussions
- **Facebook** — Tunisian finance groups (public posts)
- **Tunisia-Sat forums** — BVMT stock discussions

This complements official news sources by capturing grassroots sentiment, with automatic **comparison between social media and official news sentiment**, flagging divergences (e.g., "Social media is MUCH MORE BULLISH than news").

6.4 News Sources

Table 6.1: Monitored News Sources

Source	Language	Coverage
IlBoursa.com	French	Financial news and market updates
Tustex.com	French	Stock market analysis
TunisieNumerique.com	Arabic	General economy coverage

6.5 Supported BVMT Tickers

Table 6.2: Recognized BVMT Companies

Ticker	Company Name
SFBT	Société Frigorifique et Brasserie de Tunis
BIAT	Banque Internationale Arabe de Tunisie
BNA	Banque Nationale Agricole
SAH	Société d'Articles Hygiéniques
CARTHAGE	Ciments de Carthage
POULINA	Poulina Group Holding
DELICE	Délice Holding
EURO-CYCLES	Euro-Cycles
TELNET	Telnet Holding
TUNISAIR	Tunisair

6.6 Architecture

```
/app
|-- main.py              # FastAPI entry point
|-- config.py            # Settings (OpenRouter Key, Ticker
    List)
|-- database.py          # SQLite setup & ORM models
|-- services/
|   |-- scraper.py       # News scraping from 3 Tunisian sources
|   |-- llm.py           # OpenRouter/DeepSeek integration
|   |-- aggregator.py    # Daily sentiment score calculation
|   |-- tunizi.py        # K.O.: Tunizi/Arabizi NLP engine
|   |-- perplexity_search.py  # K.O.: Social media search
|-- routers/
    |-- sentiment.py     # API endpoints
```

Listing 6.1: Sentiment Analysis Module Structure

6.7 API Endpoints

Table 6.3: Sentiment Analysis Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
POST	`/trigger-scrape`	Trigger scraping and analysis pipeline
GET	`/sentiments/daily`	Get daily sentiment scores per ticker
GET	`/articles`	List recent articles (filterable)
GET	`/report/{company}`	Download company sentiment report (.md)
POST	`/analyze-tunizi`	**K.O.:** Analyze Tunizi/Arabizi text
POST	`/search-social-media`	**K.O.:** Search social media for a ticker
POST	`/search-social-media-batch`	**K.O.:** Batch search multiple tickers

6.7.1 Article Filtering Parameters

Parameter	Type	Required	Description
`limit`	integer	No	Maximum number of articles to return
`source`	string	No	Filter by news source (e.g., `IlBoursa`)
`ticker`	string	No	Filter by ticker symbol

6.8 Sample Response — Daily Sentiment

```
{
  "date": "2026-02-08",
  "tickers": [
    {
      "ticker": "BIAT",
      "avg_score": 0.65,
      "article_count": 3
    },
    {
      "ticker": "SFBT",
      "avg_score": -0.2,
      "article_count": 1
    }
  ]
}
```

Listing 6.2: Daily Sentiment Response Example

6.9 Performance Considerations

- **Rate Limiting** — OpenRouter free tier allows approximately 20 requests/minute

- **Background Processing** — Scraping runs asynchronously to avoid blocking
- **Deduplication** — Articles are deduplicated by title to avoid re-processing
- **Error Resilience** — If one news source fails, others continue working

Chapter 7

Portfolio Management Service

7.1 Overview

The Portfolio Management Service implements an **automated portfolio advisor** using cutting-edge AI techniques:

- **Reinforcement Learning (PPO)** — Proximal Policy Optimization for portfolio weight optimization
- **Real Macroeconomic Data** — World Bank, IMF, and BCT (Banque Centrale de Tunisie) indicators
- **Explainability (SHAP + LLM)** — AI-generated explanations in French
- **Risk Profiles** — Conservative, moderate, and aggressive strategies
- **Historical Simulation** — ROI, Sharpe, Sortino, and Max Drawdown calculations

7.2 Technology Stack

Component	Technology
Language	Python 3.12 with `uv` environment manager
API Framework	FastAPI
Demo UI	Streamlit
RL Agent	Stable-Baselines3 (PPO)
Explainability	SHAP (KernelExplainer)
LLM	OpenRouter (Gemma 3 4B)
Market Data	yfinance
HTTP Client	httpx (async)

7.3 Managed Tunisian Assets

The service manages a portfolio of **8 Tunisian bank stocks**:

`BIAT` `BH` `ATB` `STB` `SFBT` `UIB` `BNA` `ATTIJARI`

7.4 Service Architecture

```
portfolio_management_service/
|-- main.py                    # FastAPI entry point
|-- streamlit_app.py           # Demo UI
|-- app/
|   |-- api/
|   |   |-- endpoints.py     # 7 REST endpoint handlers
|   |   |-- routes.py        # Route configuration
|   |   |-- schemas.py       # Pydantic models
|   |-- core/
|   |   |-- config.py        # Centralized settings
|   |   |-- types.py         # Enums (RiskProfile, Signal,
    StressType)
|   |-- data/
|   |   |-- stock_loader.py # BVMT data via yfinance
|   |   |-- features.py      # Feature engineering (RSI, SMA,
    MACD)
|   |   |-- macro.py         # Macroeconomic data aggregation
|   |   |-- preprocessor.py # Normalization
|   |   |-- providers/
|   |       |-- world_bank.py  # 7 World Bank indicators
|   |       |-- imf.py         # 5 IMF indicators
|   |       |-- bct.py         # BCT scraping (rates)
|   |-- portfolio/
|   |   |-- tracker.py       # Portfolio state management
|   |   |-- metrics.py       # Sharpe, Sortino, Max Drawdown
|   |   |-- profile.py       # Risk profile adjustment
|   |   |-- simulator.py     # Historical simulation
|   |-- rl/
|   |   |-- environment.py   # Gymnasium PortfolioEnv
|   |   |-- rewards.py       # Sharpe-adjusted reward function
|   |   |-- agents/
|   |       |-- optimizer.py    # PPO agent
|   |       |-- adversary.py    # Adversarial training agent
|   |       |-- trainer.py      # Training pipelines
|   |-- explainability/
|       |-- shap_explain.py     # SHAP feature importance
|       |-- interpreter.py      # LLM explanation generation
|-- tests/                     # 49 unit tests
|-- models/                    # Saved RL models
```

Listing 7.1: Portfolio Management Service Structure

7.5 Design Principles

- **Modularity** — Each file $\leq$ 100 lines
- **Centralized Configuration** — All parameters in `config.py` and `.env`
- **No Hardcoded Values** — Everything is parameterizable
- **Type Safety** — Pydantic for input/output validation

- **Comprehensive Testing** — 49 tests covering all modules

7.6 API Endpoints

Table 7.1: Portfolio Management Endpoints

Method	Endpoint	Description
GET	`/api/v1/health`	Health check (status + version)
GET	`/api/v1/macro`	Aggregated macroeconomic data
GET	`/api/v1/portfolio`	Current portfolio state
POST	`/api/v1/train`	Train the RL agent
POST	`/api/v1/recommend`	**Primary endpoint:** profile → allocation + explanation
POST	`/api/v1/simulate`	Simulation with user capital
POST	`/api/v1/stress-test`	Stress test (scenario + intensity)

7.6.1 Recommend Endpoint

This is the **key integration endpoint**. It accepts a risk profile and returns a complete portfolio recommendation including weights, metrics, and an AI-generated explanation.

Request:

```
{
  "profile": "modere"
}
```

Accepted profiles: `"conservateur"`, `"modere"`, `"agressif"`.

Response:

```
{
  "profile": "modere",
  "weights": {
    "BIAT": 0.185, "BH": 0.142, "ATB": 0.098,
    "STB": 0.173, "SFBT": 0.067, "UIB": 0.145,
    "BNA": 0.089, "ATTIJARI": 0.101
  },
  "metrics": {
    "sharpe": 0.847,
    "sortino": 1.123,
    "max_drawdown": -12.34,
    "total_return": 0.0456,
    "annual_volatility": 0.1823
  },
  "explanation": "Pour votre profil modere, nous
     recommandons..."
}
```

Listing 7.2: Recommend Endpoint Response

7.6.2 Simulate Endpoint

Projects portfolio performance with the user's actual capital.

Request:

```
{
  "profile": "modere",
  "capital": 5000.0,
  "days": null
}
```

Response:

```
{
  "profile": "modere",
  "initial_capital": 5000.0,
  "final_value": 5748.32,
  "roi": 14.97,
  "sharpe": 0.821,
  "sortino": 1.098,
  "max_drawdown": -8.45,
  "volatility": 18.23,
  "n_days": 252,
  "daily_values": [5000.0, 5012.3, "...", 5748.32]
}
```

Listing 7.3: Simulate Endpoint Response

7.7 Risk Profile Constraints

Table 7.2: Risk Profile Configuration

Constraint	Conservative	Moderate	Aggressive
Max weight per asset	15%	25%	50%
Volatility cap	20%	35%	None
Minimum cash reserve	10%	5%	None

7.8 Data Pipeline

The service aggregates data from four distinct sources:

Table 7.3: Data Sources

Source	Type	Indicators
World Bank	REST API	GDP, inflation, unemployment, exchange rate, reserves, current account, public debt
IMF	REST API	GDP growth, debt/GDP, current account/GDP
BCT	Web scraping (XLS)	Policy rate, TMM, savings rate, EUR/TND, USD/TND
BVMT	yfinance API	Historical prices for 8 Tunisian bank stocks

7.9 Reinforcement Learning Pipeline

The RL pipeline follows these stages:

1. **Feature Engineering** — RSI, SMA, MACD, rolling volatility computed from OHLCV data
2. **Environment** — Custom Gymnasium `PortfolioEnv` with observation space combining prices, features, and macro indicators
3. **Agent Training** — PPO agent from Stable-Baselines3 with configurable hyperparameters:
   - Learning rate: 3×10^{-4}
   - Discount factor (γ): 0.99
   - GAE lambda: 0.95
   - Clip range: 0.2
4. **Reward Function** — Sharpe ratio adjusted with drawdown and stress penalties
5. **Profile Adjustment** — Post-training weight adjustment based on user risk profile
6. **Explainability** — SHAP analysis identifies feature importance; LLM generates French explanation

7.10 Testing

The service includes **49 unit tests** organized by module:

Test File	Coverage
`test_api.py`	REST endpoint tests
`test_data.py`	Data provider tests (WB, IMF, BCT, yfinance)
`test_explainability.py`	SHAP and LLM tests
`test_portfolio.py`	Metrics and simulator tests
`test_rl.py`	RL environment and agent tests

```
# Run all tests
pytest tests/ -v

# With coverage report
pytest tests/ --cov=app --cov-report=html
```

Listing 7.4: Running Tests

Chapter 8

Supporting Microservices

8.1 Stock Service

The Stock Service is a dedicated microservice for **stock data operations**, providing CRUD access to BVMT stock information.

Table 8.1: Stock Service Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
GET	`/stocks`	List all stocks
GET	`/stocks/{ticker}`	Get stock by ticker
GET	`/stocks/{ticker}/history`	Get historical prices

Port: 8001

8.2 Market Service

The Market Service provides **market-level statistics and analytics**, including the TUNINDEX overview and top movers.

Table 8.2: Market Service Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
GET	`/overview`	Market overview with TUNINDEX
GET	`/gainers`	Top gaining stocks
GET	`/losers`	Top losing stocks

Port: 8002

8.3 Notification Service

The Notification Service handles **email dispatch and alert management** for the platform.

Table 8.3: Notification Service Endpoints

Method	Endpoint	Description
GET	`/health`	Service health check
POST	`/email/send`	Send an email
POST	`/alert/anomaly`	Send an anomaly alert
GET	`/test`	Test email configuration

Port: 8003

8.4 Jobs Service

The Jobs Service manages **background job scheduling and processing** for automated platform tasks.

Table 8.4: Scheduled Jobs

Job	Frequency	Description
Market Pulse	Every 15 minutes	Fetch and analyze market news
Anomaly Detection	Hourly	Detect price and volume anomalies
Daily Report	18:00 daily	Generate daily market summary

8.5 Chatbot Service (RAG)

The Chatbot Service is a **Retrieval-Augmented Generation (RAG)** chatbot that allows users to ask natural-language questions about the FAIN platform. It combines a local knowledge base with LLM-powered responses for accurate, context-grounded answers.

8.5.1 Technology Stack

Component	Technology
Framework	FastAPI
Vector Database	ChromaDB (persistent, on-disk)
Embeddings	all-MiniLM-L6-v2 (via ChromaDB default)
LLM	Meta Llama 3.3 70B Instruct (via OpenRouter, free tier)
Knowledge Base	Custom `knowledge_base.txt` (476 lines, 10 topic areas)

8.5.2 Architecture

The service implements a classic RAG pipeline:

1. **Chunking** — The knowledge base is split into overlapping chunks (500 chars, 100-char overlap)

2. **Embedding & Indexing** — Chunks are embedded and stored in ChromaDB on first startup
3. **Retrieval** — User queries are embedded and matched against the top-3 most relevant chunks
4. **Generation** — The retrieved context + conversation history are sent to the LLM
5. **Memory** — Per-session conversation history (up to 10 messages) enables multi-turn dialogue

8.5.3 API Endpoints

Table 8.5: Chatbot Service Endpoints

Method	Endpoint	Description
POST	`/chat`	Send a message and receive an AI response
DELETE	`/chat/{session_id}`	Clear conversation history for a session
GET	`/health`	Service health check

Port: 8004

8.5.4 Frontend Integration

The chatbot is accessible from every page via a **floating bubble button** (`chatbot-bubble.tsx`) in the bottom-right corner of the UI. The bubble opens a chat panel supporting real-time messaging, conversation history, and session clearing.

Chapter 9

Frontend — Next.js Application

9.1 Overview

The frontend is a **Next.js 16 application** built with TypeScript and Tailwind CSS. It provides the user-facing dashboard for interacting with all backend services.

9.2 Technology Stack

- **Framework:** Next.js 16 (App Router)
- **Language:** TypeScript
- **Styling:** Tailwind CSS
- **Font:** Geist (optimized via `next/font`)
- **Package Manager:** pnpm

9.3 Application Pages

Table 9.1: Frontend Routes

Route	Description
`/`	Main dashboard
`/login`	User authentication
`/signup`	User registration
`/analyse`	Market analysis view
`/prevision`	Price prediction view
`/sentiment`	Sentiment analysis heatmap and social media view
`/surveillance`	Anomaly monitoring view
`/portefeuille`	Portfolio management view

9.4 Key Components

- `client-shell.tsx` — Application shell with layout
- `sidebar.tsx` — Navigation sidebar
- `mobile-nav.tsx` — Responsive mobile navigation

- `require-auth.tsx` — Authentication guard component
- `chatbot-bubble.tsx` — Floating RAG chatbot widget (available on all pages)
- `sentiment-heatmap.tsx` — Color-coded sentiment heatmap visualization
- `social-media-sentiment.tsx` — Social media sentiment search and comparison
- `ui/` — Reusable UI component library

9.5 Development

```
cd ui

# Install dependencies
pnpm install

# Start development server
pnpm run dev

# Production build
pnpm run build
pnpm start
```

Listing 9.1: Frontend Development Commands

The development server runs on `http://localhost:3000` with hot-reloading enabled.

9.6 Deployment

The frontend includes a `Dockerfile` for containerized deployment as part of the Docker Compose stack. For cloud deployment, Vercel is the recommended platform for Next.js applications.

Chapter 10

Data Layer

10.1 Database

The platform uses **PostgreSQL 16** with the **TimescaleDB** extension for time-series optimized storage. The database schema is initialized via the SQL script at `database/init.sql`.

10.2 Historical Data

The `data/` directory contains historical BVMT quotation data spanning from 2016 to 2026:

Table 10.1: Historical Data Files

Period	Format
2016–2021	Text files (`.txt`)
2022–2026	CSV files (`.csv`)

10.3 Data Engineering Scripts

The `data/data engineering/` directory contains utility scripts for data preparation:

- `fix_seance_format.py` — Fixes trading session date formats
- `modify_and_save.py` — Data transformation and persistence
- `visualize.py` — Data visualization utilities
- `write_csv_to_db.py` — CSV to PostgreSQL ingestion
- `write_xls_to_db.py` — Excel to PostgreSQL ingestion

Chapter 11

Testing

The platform includes comprehensive test suites across multiple services:

```
# NestJS backend
cd backend/core && pnpm run test

# Portfolio service (49 unit tests)
cd backend/services/portfolio_management_service && pytest
   tests/ -v

# K.O. features
python test_ko_features.py
```

Listing 11.1: Running All Tests

Chapter 12

Integration Patterns

12.1 Service Communication Flow

The system follows a request-response pattern where the frontend communicates with the API Gateway, which in turn proxies requests to the appropriate microservices.

1. User interacts with the **Next.js frontend**
2. Frontend sends HTTP requests to the **API Gateway** (`:8000`)
3. Gateway routes to the target **Python microservice**
4. Microservice processes the request (database queries, ML inference, external API calls)
5. Response propagates back through the chain

12.2 Backend Integration Example (Python)

```python
import httpx

PORTFOLIO_API = "http://localhost:8000/api/v1"

async def get_portfolio_recommendation(
    user_profile: str, capital: float
):
    async with httpx.AsyncClient(timeout=120) as client:
        # 1. Get recommendation
        rec = await client.post(
            f"{PORTFOLIO_API}/recommend",
            json={"profile": user_profile}
        )
        recommendation = rec.json()

        # 2. Simulate with user capital
        sim = await client.post(
            f"{PORTFOLIO_API}/simulate",
            json={"profile": user_profile, "capital": capital}
        )
        simulation = sim.json()

        return {
            "allocation": recommendation["weights"],
            "metrics": recommendation["metrics"],
            "explanation": recommendation["explanation"],
            "simulation": simulation
        }
```

Listing 12.1: Python Integration Example

12.3 Backend Integration Example (TypeScript)

```typescript
async function getPortfolioRecommendation(
  userProfile: 'conservateur' | 'modere' | 'agressif',
  capital: number
) {
  const API = 'http://localhost:8000/api/v1';

  const recResponse = await fetch(`${API}/recommend`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ profile: userProfile }),
  });
  const recommendation = await recResponse.json();

  const simResponse = await fetch(`${API}/simulate`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify({ profile: userProfile, capital }),
  });
  const simulation = await simResponse.json();

  return { ...recommendation, simulation };
}
```

Listing 12.2: TypeScript Integration Example

Chapter 13

Conclusion

FAIN demonstrates a comprehensive, production-oriented approach to building an **intelligent trading assistant** for the Tunisian Stock Exchange. The platform's key technical achievements include:

- A **modular microservice architecture** (10 independently deployable services) enabling independent development, testing, and scaling of each functional domain
- **On-the-fly ML model training** ensuring forecasts always reflect the most current market data, with **40+ engineered features** including lagged prices, MA, EMA, MACD, RSI, and Bollinger Bands
- **Multi-source sentiment analysis** with native support for French, Arabic, and Tunisian dialect, scraping from 3 Tunisian financial news sources (IlBoursa, Tustex, TunisieNumerique)
- A **Tunizi/Arabizi NLP engine** (K.O. feature) — a purpose-built linguistic processor for Tunisian financial slang with Arabizi normalization, 30+ sentiment-weighted terms, company nickname mapping, and hybrid scoring (60% dictionary + 40% LLM)
- **Social media sentiment search** (K.O. feature) via Perplexity AI Sonar, capturing retail investor sentiment from Twitter/X, Reddit r/tunisia, Facebook, and Tunisia-Sat forums, with automatic divergence detection between social media and official news sentiment
- A novel **Reinforcement Learning portfolio optimizer** (PPO via Stable-Baselines3) managing 8 Tunisian bank stocks, augmented with real macroeconomic data from the World Bank, IMF, and BCT
- **Explainable AI** through SHAP feature importance analysis combined with LLM-generated natural language explanations in French
- A **RAG-powered chatbot** using ChromaDB and Llama 3.3 70B for context-grounded Q&A, with multi-turn conversation memory and a floating bubble accessible from every page
- **JWT-based authentication** (NestJS 11) protecting all backend routes with user management
- **Containerized deployment** via Docker Compose for reproducible, single-command infrastructure setup

The platform is designed with extensibility in mind — additional stock markets, news sources, ML models, and risk profiles can be integrated with minimal changes to the existing architecture.