North South University

Department of Electrical & Computer Engineering

Project Proposal

Course Name: CSE311 Database Management System

Section: 3

Project Name: Business Analyzer

Submitted To: Dr. Sumaiya Tabassum Nimi

Submission Date: 5 November 2025

Semester: Fall 2025

Group: 8

Student Name	ID
Khan Asfi Reza Fateh Tus Saad Ahmed As Sadik	2321616642 2321638642 2322539042

Project Proposal

Project Overview

Business Analyzer - Investment Decision Support System

Problem Statement

New investors face several significant challenges when attempting to invest their money. First, they lack access to the powerful analysis tools used by large investment companies. These professional tools cost thousands of dollars per month, which regular people cannot afford.
Second, finding information about companies is a challenging and time-consuming process. Stock prices are available on one website, company news is on another, financial reports are located elsewhere, and expert opinions are accessible only behind paid subscriptions. An investor might spend hours collecting basic information about a single company.
Third, understanding whether an investment is safe or risky requires financial knowledge that many beginners lack. They need to read financial statements, understand profit margins, analyze market trends, and predict future performance. Without this expertise, people often make poor investment choices based on guesswork or incomplete information, which can result in financial losses. Last but not least, the majority of financial and investment solutions offer insights for the US Market but not the regional (Bangladesh) market.

Solution

A web-based platform that collects financial data, news, and reviews from various sources, applies AI sentiment analysis and statistical algorithms, and provides clear investment recommendations with risk ratings. The system automatically collects data from stock markets, news websites, and company reports, eliminating the need for users to visit multiple sites.

The platform uses AI to read news articles and social media posts about companies. It determines whether people are saying good things or bad things about a company, assigning it a positive, negative, or neutral rating. This helps users quickly understand what others think about an investment. The system then combines this sentiment data with actual financial data, such as stock prices, company revenue, and profit. It utilizes mathematical formulas to analyze this information and provides users with recommendations on company or asset investments. Each recommendation includes a risk level (Low, Medium, or High) and explains why the system assigns this level. Users can search for any company, see price charts, read financial data, and save interesting companies to their watchlist. The system continually checks these saved companies and alerts users if any important changes occur.

The web-based application solution not only provides insight into the Global Market but also the regional (Bangladesh) market, and it can be easily extended to multinational markets.

Target Users

- Beginner investors
- Retail traders
- Students learning about markets
- Anyone seeking investment guidance

Significant Features

1. User Management
    - User registration and login
    - Secure authentication with password encryption
    - Profile management
2. Company Search
    - Search any public or private company
    - View stock prices and historical trends
    - See financial statements (revenue, profit/loss by quarter and year)
    - Company information: founding date, industry, type
3. Asset Tracking
    - Gold, silver, and commodity prices
    - Historical price data and charts
    - Price trend analysis
4. Bookmarking
    - Save companies and assets to the watchlist
    - Track multiple investments in one place
    - Add personal notes to bookmarks
5. Web Scraping
    - Automated collection of news articles and reviews
    - Multiple sources: financial news sites, blogs, social media
    - Scheduled scraping every few hours
    - Content stored with source URL and timestamp
6. Sentiment Analysis
    - AI-powered natural language processing
    - Classification: Positive, Negative, Neutral
    - Sentiment score (-1 to +1)

7. Investment Recommendations
    - Sentiment scores from news
    - Stock price trends and volatility
    - Financial metrics (revenue growth, profit margins)
    - Statistical indicators
    - Recommendation: Invest, Don't Invest, Wait
    - Risk level: Low, Medium, High

Database Entities

1. USER

Stores user account information.

- user_id (Primary Key)
- username
- email
- password_hash
- full_name
- registration_date
- role (Admin / Customer / Moderator)

2. COMPANY

Stores business information.

- company_id (Primary Key)
- company_name
- company_type (public/private)
- industry
- logo_url
- founded_date
- description
- market_cap

3. STOCK_PRICE

Historical and current stock prices.

- price_id (Primary Key)
- company_id (Foreign Key → COMPANY)
- date
- open_price
- currency
- close_price
- high_price
- low_price
- volume

4. FINANCIAL_STATEMENT

Company financial data.

- statement_id (Primary Key)
- company_id (Foreign Key → COMPANY)
- statement_type (quarterly/yearly)
- period_start_date
- period_end_date
- currency
- revenue
- profit

5. ASSET

Investment assets like gold and silver.

- asset_id (Primary Key)
- asset_name
- asset_type (commodity/precious_metal)
- unit_of_measurement
- description
- logo

6. ASSET_PRICE

Historical asset pricing.

- asset_price_id (Primary Key)
- asset_id (Foreign Key → ASSET)
- date
- price
- currency

7. SCRAPED_CONTENT

News and reviews collected from the web.

- content_id (Primary Key)
- company_id (Foreign Key → COMPANY)
- source_url
- title
- content_text
- content_type (news/review/social_media)
- scraped_date
- publish_date
- author
- source_name

8. SENTIMENT_ANALYSIS

AI analysis results for scraped content.

- sentiment_id (Primary Key)
- content_id (Foreign Key → SCRAPED_CONTENT)
- sentiment_score (-1 to +1)
- sentiment_label (positive/negative/neutral)
- confidence_level

- analysis_date

9. INVESTMENT_RECOMMENDATION

AI-generated investment advice.

- recommendation_id (Primary Key)
- company_id (Foreign Key → COMPANY, nullable)
- asset_id (Foreign Key → ASSET, nullable)
- recommendation_date
- recommendation_type (invest/don't_invest/hold/wait)
- investment_score
- risk_level (low/medium/high)
- expected_return
- rationale_summary

10. BOOKMARK

User-saved companies and assets.

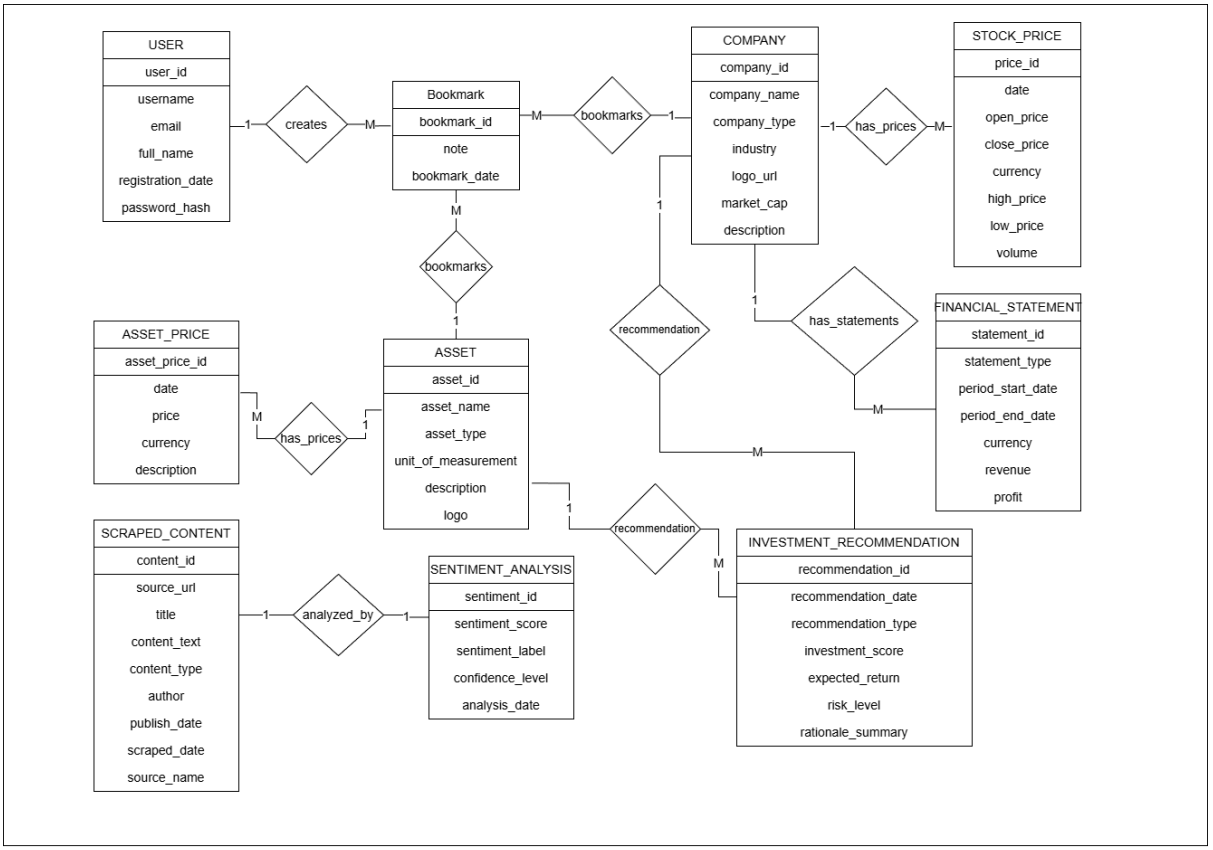
- bookmark_id (Primary Key)
- user_id (Foreign Key → USER)
- company_id (Foreign Key → COMPANY, nullable)
- asset_id (Foreign Key → ASSET, nullable)
- bookmark_date
- notes

Entity Relationships

Here, (P) → Partial , (F) → Full

From Entity (Participation)	Relationship Name (Cardinality)	To Entity (Participation)
USER (P)	creates 1: M	BOOKMARK (F)
COMPANY (P)	has_prices 1:M	STOCK_PRICE (F)
COMPANY (P)	has_statements 1:M	FINANCIAL_STATEMENT (F)
COMPANY (P)	mentioned_in 1:M	SCRAPED_CONTENT (F)
COMPANY (P)	bookmarked_by 1:M	BOOKMARK (F)
COMPANY (P)	has_recommendations 1:M	INVESTMENT_RECOMMENDATION (F)
ASSET (P)	has_prices 1:M	ASSET_PRICE (F)
ASSET (P)	bookmarked_by 1:M	BOOKMARK (F)
ASSET (P)	has_recommendations 1:M	INVESTMENT_RECOMMENDATION (F)
SCRAPED_CONTENT (F)	analyzed_by 1:1	SENTIMENT_ANALYSIS (F)

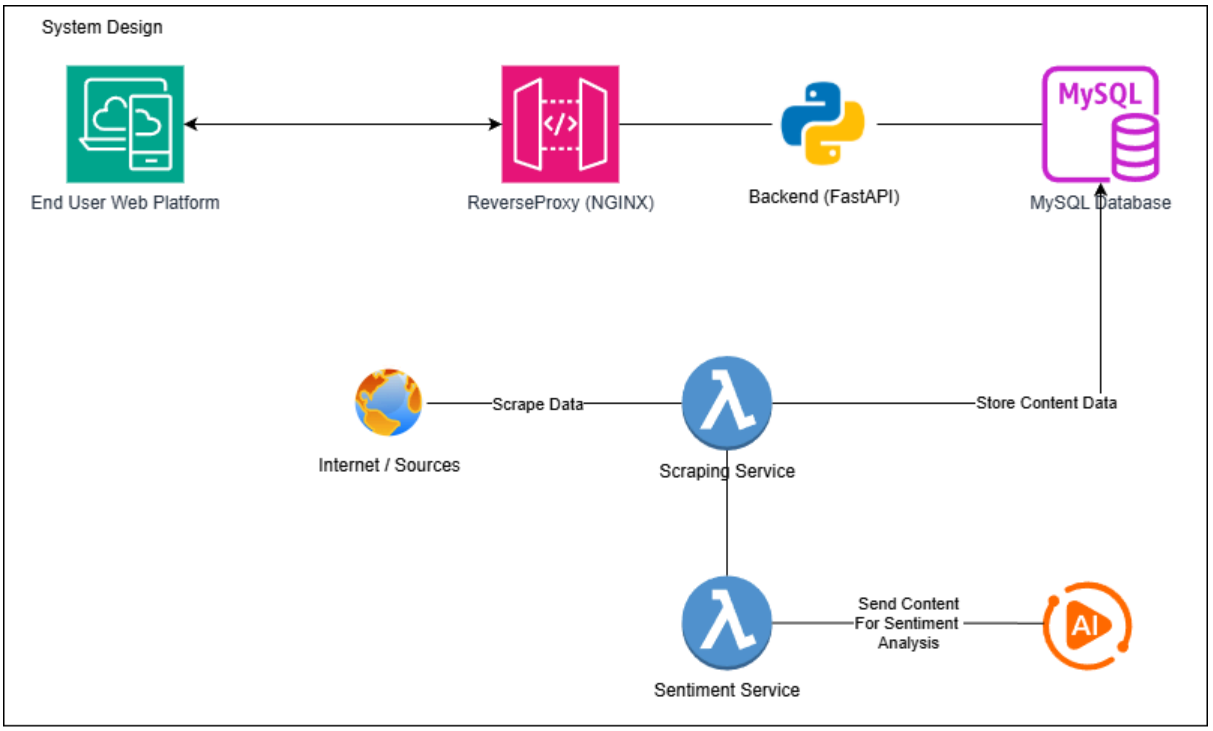
Entity Relationship Diagram



Key Design Decisions

1. Nullable Foreign Keys: BOOKMARK can reference either COMPANY or ASSET, so both foreign keys are nullable (only one will be filled).
2. Separate Price Tables: STOCK_PRICE and ASSET_PRICE are separate because they have different attributes and use cases.
3. Historical Data: Time-series data (prices, recommendations) stored with timestamps for trend analysis.
4. Weak Entities: ASSET_PRICE, STOCK_PRICE, FINANCIAL_STATEMENT, SENTIMENT_ANALYSIS

System Design



Tech Stack

1. Backend (Python FastAPI)
2. Frontend (HTML, CSS, JS)
3. Database - MySQL
4. Scraper (Python, BeautifulSoup4, Scrapy)
5. AI (HuggingFace, LLMA)

Formula

Recommendation Without AI

$Stock\ Price\ Function\ fp(t),\ here\ t\ is\ today's\ date\ (\ recommendation\ date\ )$

$Last\ 30\ days\ Average\ Price,\ AvgP1\ =\ \frac{\sum_{i=0}^{30} fp(t-i)}{30}$

$Average\ Price\ Previous\ Period,\ AvgP2\ =\ \frac{\sum_{i=30}^{60} fp(t-i)}{30}$

$Price\ Change,\ pc\ =\ \frac{AvgP1 - AvgP2}{AvgP2}$

$Price\ Score,\ Ps\ =\ 50\ +\ (50\ \times\ Pc)$

$Revenue\ Function\ r(t),\ here\ t\ is\ current\ time\ frame\ \ (\ recommendation\ date\ )$

$Financial\ Score,\ Fs\ =\ 50\ +\ (50\ \times\ \frac{r(t) + r(t-1)}{r(t-1)})$

$Investment\ Score,\ Is\ =\ (0.5\ \times Ps\ )\ +\ (0.5\ \times\ Fs)$

$If:\ Is\ \geq\ 60\ \rightarrow\ Invest$

$If:\ Is\ \geq\ 40\ \rightarrow\ Hold$

$If:\ Is\ <\ 40\ \rightarrow\ Don't\ Invest$

Recommendation With AI

$Sentiment\ Score\ Function\ s(t),\ here\ t\ is\ today's\ date\ (\ recommendation\ date\ )$

$Sentiment\ Confidence\ Function\ cf(t),\ here\ t\ is\ today's\ date\ (\ recommendation\ date\ )$

$$Last\ 30\ days\ Average\ Sentiment,\ Sc\ =\ \frac{\sum_{i=0}^{30} s(t-i)}{30}$$

$$Last\ 30\ days\ Average\ Sentiment\ Confidence,\ AvgConf\ =\ \frac{\sum_{i=0}^{30} cf(t-i)}{30}$$

$If:\ AvgConf\ >\ 0.5$

$Investment\ Score,\ Is\ =\ (0.3\ \times\ Ps)\ +\ (0.3\ \times\ Fs)\ +\ (0.4\ \times\ Sc)$

$Else:$

$Investment\ Score,\ Is\ =\ (0.4\ \times\ Ps)\ +\ (0.4\ \times\ Fs)\ +\ (0.2\ \times\ Sc)$

$Risk\ Level$

$if:\ Sc\ <\ 40\ |\ Fs\ <\ 30\ \rightarrow\ High\ Risk$

$if:\ Sc\ >\ 70\ \&\ Fs\ >\ 60\ \rightarrow\ Low\ Risk$

$else:\ Medium\ Risk$

$If:\ Is\ \geq\ 70\ \rightarrow\ Invest\ (Strong\ )$

$If:\ Is\ \geq\ 55\ \rightarrow\ Invest\ (Moderate\ )$

$If:\ Is\ \geq\ 40\ \rightarrow\ Hold$

$If:\ Is\ <\ 40\ \rightarrow\ Don't\ Invest\ Or\ Short\ Sell$

Note: For Asset Recommendation, Only Price and Sentiment will be used

$If:\ AvgConf\ >\ 0.5$

$Investment\ Score,\ Is\ =\ (0.5\ \times\ Ps)\ +\ (0.5\ \times\ Sc)$

$Else:$

$Investment\ Score,\ Is\ =\ (0.7\ \times\ Ps)\ +\ (0.3\ \times\ Sc)$

Future Improvement

1. Add a User Subscription system and track search history
2. Add a caching layer to improve the latency issue
3. Use advanced NLP Models.
4. Improve Investment Formula accuracy and add a more versatile algorithm

Software Requirements Specification (SRS)
Frontend — Business Analyzer

1. Introduction

1.1 Purpose

The purpose of this document is to specify the frontend requirements for the web-based platform Business Analyzer – Investment Decision Support System. The frontend will provide beginner investors, retail traders, and students with a clear and easy-to-use interface to explore companies, view financial data, inspect sentiment, and manage a personal watchlist.

This SRS focuses only on the presentation layer. Backend logic, database operations, scraping, and AI computation are assumed to be handled by the FastAPI-based backend and are outside the scope of this document.

1.2 Scope

The frontend will:

- Present investment-related information such as company data, stock price history, financial summaries, sentiment scores, and recommendations.
- Allow users to register, log in, and interact with their personalized dashboard.
- Provide a simple, minimal, and responsive UI using HTML templates rendered via FastAPI Ninja templating, styled with Tailwind CSS.
- Serve as a thin client: it will consume backend APIs and present data in a structured, user-friendly manner without heavy client-side logic.

The frontend will not:

- Perform complex calculations (formula-based investment scores, risk classification).
- Run AI models in the browser.
- Implement advanced SPA behavior or complex JavaScript frameworks.

1.3 Definitions and Acronyms

- IDSS – Investment Decision Support System
- UI – User Interface
- UX – User Experience
- API – Application Programming Interface
- Ninja Templates – Template system used with FastAPI for server-side HTML rendering
- Watchlist – User-specific list of tracked companies and assets

1.4 References

- *Project Proposal – Business Analyzer, CSE311 Database Management System*

2. System Overview (Frontend Perspective)

From the frontend perspective, Business Analyzer is a server-rendered web application. Users access a set of pages:

- A public landing page describing the platform and offering a search bar.
- Authentication pages (login and registration).

- An authenticated dashboard summarizing the user's tracked companies and assets.

- Detail pages for companies and assets, including price trends, sentiment, and recommendations.
- A watchlist page for managing bookmarks.
- A news and sentiment view for reading recent market content.

The frontend receives structured data via backend APIs or template context and renders it using HTML + Tailwind CSS.

---

3. Overall Description

3.1 Product Perspective

The frontend is a presentation layer on top of:

- Backend: FastAPI, responsible for user authentication, business logic, scraping, and AI.
- Database: MySQL, storing entities such as USER, COMPANY, STOCK_PRICE, ASSET, SCRAPED_CONTENT, etc.

The frontend does not access the database directly. All content is provided via backend endpoints and passed to templates.

3.2 User Classes and Characteristics

- Beginner Investors
    - Limited financial knowledge.
    - Need simple, non-technical explanations and visual cues.
- Retail Traders
    - Familiar with basic market concepts.
    - Expect quick access to price trends and sentiment.
- Students / Learners

    - Use the platform as a learning tool.
      Need clarity, structure, and easy navigation.

The frontend must remain clean and self-explanatory, without overwhelming users with dense financial jargon.

3.3 Operating Environment

- Web browser (desktop and mobile)
- Modern browsers: Chrome, Firefox, Edge, Safari (latest versions)
- Served by a FastAPI backend with Ninja templating

3.4 Design and Implementation Constraints

- Technology:
    - HTML templates rendered server-side.
    - Tailwind CSS for styling.
    - Minimal JavaScript, used only when necessary (e.g., basic interactivity, chart placeholders).
- Complexity:
    - Pages and components must be simple enough to be implemented within a short project window.
- No SPA Frameworks:
    - No React, Vue, or similar frameworks.
- No Future Enhancements in Scope:
    - Subscription systems, advanced caching, and sophisticated AI model selectors are explicitly out of scope for this version.

3.5 Assumptions and Dependencies

- Backend APIs will provide data in a structured and predictable format.
- User authentication, authorization, and session handling are correctly implemented in the backend.
- Tailwind CSS is properly integrated into the build or served as a compiled CSS file.
- Test data for companies, assets, and recommendations will be available during development.

4. Specific Functional Requirements (Frontend)

This section defines what the frontend must display and allow the user to do, assuming backend APIs work as expected.

4.1 Navigation & Layout

FR-1: Global Navigation Bar

- The application must display a top navigation bar on all main pages (except optional auth pages).
- The navbar must include at least:
    - Logo or text: *Business Analyzer*
    - Links: *Home*, *Search*, *Watchlist*, *News* (optional link name: *Insights*), *Login/Register* or *Logout*.
- For logged-in users, show:
    - *Dashboard*
    - *Watchlist*
    - *Logout* button/link.

FR-2: Responsive Layout

- Navigation must collapse or adjust on smaller screens.
- All main layout containers must be responsive using Tailwind utility classes.

4.2 Landing Page

FR-3: Introduction Section

- The landing page must present:
    - Project name: *Business Analyzer – Investment Decision Support System*.
    - A short description summarizing its goal, consistent with the problem and solution statements in the proposal.

FR-4: Search Entry Point

- A central search bar must allow the user to type a company or asset name.
- On form submission, the user is redirected to the Search Results page with the query applied.

FR-5: Quick Access Cards

- The page must provide quick-access tiles or buttons, such as:
    - *Bangladesh Market*
    - *Global Market*
    - *Commodities*

- These may route to a filtered search or a generic listing page (as supported by backend).

-

4.3 Authentication UI

FR-6: Login Page

- Must provide:
    - Input for email.
    - Input for password.
    - *Login* button.

- On invalid credentials, show a user-friendly error message.
- Provide a link: *Don't have an account? Register*.

FR-7: Registration Page

- Must provide:
    - Input for username.
    - Input for email.
    - Input for password (and optionally password confirmation).

- After successful registration, show a confirmation message and provide a link or redirection to the Login.
- Basic validation messages must be displayed for missing or malformed input (client-side or server-side).

4.4 Dashboard

FR-8: User Summary View

- After login, users are directed to a Dashboard.
- The dashboard must show:
    - A welcome message with the user's name.

    - A count or summary of:
        - Number of companies in watchlist.
        - Number of assets in watchlist.
    - A section labeled *Recent Recommendations*.

FR-9: Recommendation Overview

- The dashboard must list a small set of recent recommendations with:
    - Company/asset name.
    - Recommendation type (Invest / Don't Invest / Hold / Wait).
    - Risk level (Low, Medium, High).

4.5 Search Interface

FR-10: Search Results Page

- A page dedicated to displaying:
    - Search query.
    - A list of matching companies and/or assets.
- Each result card must show:
    - Name (company or asset).
    - Type (Company / Asset; for companies: public/private).
    - Industry or asset type where available.
    - A link/button: *View Details*.

FR-11: Empty State

- If no results are found, show an informative message such as:
    - "No matching companies or assets were found for this search.

4.6 Company Details Page

FR-12: Company Header Section

- The company detail page must display:
    - Company name.
    - Logo (if provided).
    - Industry.
    - Company type (public/private).
    - Founded date.

FR-13: Stock Price Summary

- The page must include:
    - A simple price summary (e.g., latest close price, currency).
    - A placeholder for a line chart or price history visualization area.
        - Implementation can be a basic SVG, static image, or simple canvas placeholder.
        - The main requirement is a *visual area* designated for price trends.

FR-14: Financial Summary Section

- Show key financial metrics provided by backend, such as:
    - Revenue (latest period).
    - Profit (latest period).
    - Statement type (quarterly/yearly).
- Displayed in a clear table or card layout.

FR-15: Sentiment & Recommendation Section

- A Sentiment Card:
    - Display sentiment label (Positive / Neutral / Negative).
    - Optional sentiment score if provided (e.g., -1 to +1 scaled or simplified).
- A Recommendation Card:
    - Show recommendation type (Invest / Don't Invest / Hold / Wait).
    - Risk level (Low / Medium / High).
    - Short rationale or summary text.

FR-16: Watchlist Interaction

- Include a button:
    - *Add to Watchlist* if the company is not yet saved.
    - *Remove from Watchlist* if already saved.

- The button must visually change state after the action (under the assumption backend confirms).

4.7 Asset Details Page

FR-17: Asset Header Section

- Display:
    - Asset name.
    - Asset type (commodity/precious metal, etc.).
    - Unit of measurement (e.g., "per gram", "per ounce").

FR-18: Asset Price Summary & Trend

- Show current price and currency.
- Provide a visual area similar to a company price chart for historical asset pricing.

FR-19: Sentiment & Recommendation

- Show sentiment and recommendation, similar to the company detail view, but only using price and sentiment (as per proposal logic, though computation is backend-side).

FR-20: Watchlist Button

- Same behavior as for the company detail page.

4.8 Watchlist Page

FR-21: Watchlist Listing

- Display a list/grid of all watchlisted items:
    - Company name or asset name.
    - Type (Company/Asset).
    - Optional tag for recommendation or last known risk.

FR-22: Access to Details

- Each item must link to its corresponding detail page.

FR-23: Notes Section

- For each watchlist item, there must be:
    - A short text area or display area for user notes.
    - A way to save updated notes (via form submission).

FR-24: Remove from Watchlist

- Each item must include a *Remove* button that, when triggered, removes the bookmark (backend-driven) and updates the UI accordingly.

4.9 News & Sentiment Page

FR-25: News/Scraped Content Listing

- Provide a page that lists recent news and review entries related to companies:
    - Title.
    - Source name.
    - Content type (news/review/social media).
    - Publish date and scraped date.
    - Short description or excerpt.

FR-26: Sentiment Indicator

- Each content item must show:
    - Sentiment label (Positive / Neutral / Negative), visually distinguished (e.g., colored badge).
- Optional:
    - Confidence score if provided.

FR-27: Link to Source

- Each news item must include:
    - A link to the original source URL (opening in a new tab).

4.10 Error & Status Messaging

FR-28: Global Status Messages

- The frontend must display user-friendly messages for:
    - Failed login.
    - Invalid form input.
    - Empty watchlist.
    - No search results.

FR-29: Consistent Styling

- Messages must use a consistent style (e.g., Tailwind classes for success/error alerts).

5. Non-Functional Requirements (Frontend)

5.1 Usability

- Clear headings and section titles.
- Simple navigation, minimal number of clicks to reach key features.
- Avoid jargon where possible; when present, keep it short and contextual.

5.2 Performance

- Pages must remain lightweight.
- Avoid heavy client-side libraries and unnecessary animations.

5.3 Reliability

- The UI must degrade gracefully if some data fields are missing (e.g., no logo, no sentiment yet).

5.4 Security (Frontend Concerns)

- Password fields must be <input type="password">.
- No sensitive data stored in client-side scripts or local storage by default.
- Forms must be protected via backend (CSRF, etc.), but templates must be compatible.

5.5 Compatibility

- Must render correctly on common desktop resolutions.
- Mobile view must be usable: readable font sizes, touch-friendly buttons.

6. User Interface Design Guidelines

6.1 Visual Style

- Use Tailwind to maintain a consistent design:
    - Neutral light background (e.g., bg-gray-50).
    - White cards with rounded corners (rounded-xl) and subtle shadow (shadow or shadow-md).
    - Headings in darker gray (text-gray-800), body text in medium gray (text-gray-600).

6.2 Layout

- Overall layout:
    - Sticky top navbar.
    - Centered container for primary content (max-w-4xl or max-w-6xl, mx-auto).
- Sections separated with reasonable spacing (mt-6, mt-8, etc.).

6.3 Typography

- Utilize Tailwind's default font stack.
- Hierarchy:
    - Page titles: text-2xl / text-3xl + font-bold.
    - Section titles: text-xl + font-semibold.
    - Body text: text-sm or text-base.

6.4 Color Coding

- Sentiment badges:
    - Positive: green-ish background.
    - Neutral: gray/blue background.
    - Negative: red/orange background.
- Recommendation types can use similar visual cues.

7. Frontend Deliverables

The frontend implementation will produce the following main deliverables:

1. Base Layout Template
    - Shared navbar, footer, basic container structure.
    - Tailwind CSS integration.

2. Page Templates

    - index.html – Landing page with search and quick links.
    - login.html – User login UI.
    - register.html – User registration form.
    - dashboard.html – User dashboard with summary and recent recommendations.
    - search_results.html – Company/asset search result listing.

- company_detail.html – Company information, price trend placeholder, financial summary, sentiment, recommendation, and watchlist button.
- asset_detail.html – Asset information, price summary, sentiment, recommendation, and watchlist button.
- watchlist.html – List of saved companies and assets, with notes and remove actions.
- news.html – News and sentiment listing with source links.
- admin.html – Admin portal for administration.

3. Shared UI Components (within templates)
    - Navigation bar partial.
    - Alert/notification partial (success/error).
    - Reusable cards for:
        - Recommendation display.
        - Sentiment badge.
        - Company/asset list items.
4. Static Assets (if needed)
    - Placeholder icons/logos.
    - Basic chart placeholder (SVG/div structure ready for data).

Frontend — Business Analyzer

Business Analyzer – Investment Decision Support System

Backend only. The frontend SRS and project proposal are above

1. Purpose & Scope

The backend provides all server-side functionality for Business Analyzer, including:

- Authentication and user management
- Company/asset search and detail views
- Watchlist management
- Web scraping and data ingestion
- Sentiment analysis using pluggable AI providers
- Investment recommendation generation based on formulas in the proposal
- HTML rendering via Jinja2 templates for the frontend pages (defined in frontend SRS)

This SRS describes backend architecture, data model, API contracts, background jobs, AI abstraction, and non-functional requirements.

2. Technology & Hard Constraints

1. Backend framework: FastAPI (required)
2. Database: MySQL (required)
3. No ORM:
    - No SQLAlchemy ORM, Tortoise, etc.
    - All DB access must be via plain SQL using utility functions.
4. Templates: Jinja2 (FastAPI Jinja2Templates) for all HTML views.
5. Schema file:
    - There must be exactly one SQL schema file: schema.sql
    - It must contain all CREATE TABLE, indexes, and constraints.
6. SQL utility layer:

Common function used everywhere, e.g.:
data = sql(db, "SELECT ... WHERE id=%s", (id,))

    - Additional helpers for fetch_one, fetch_all, execute, and transactions.

7. Background workers: Celery (with Redis or RabbitMQ) for scraping, sentiment analysis, and recommendation jobs.
8. AI APIs:
    - Must support multiple providers (e.g., Hugging Face, OpenAI, Gemini).

There must be one internal AI service interface:
ai_service = AIService(model="openai")
ai_service.classify_sentiment(text)
Default provider MUST be Hugging Face when model is omitted:
ai_service = AIService() # uses huggingface by default

9. Prompts:
    - All prompts used for AI calls must be kept in one universal location (e.g., /prompts module) and referenced by key, not duplicated inline.

3. High-Level Architecture

3.1 Components

- FastAPI app
    - HTTP routes (HTML + JSON)
    - Dependency injection for DB connection, current user, etc.
- Jinja2 Template layer
    - Renders all frontend pages described in the frontend SRS (landing, dashboard, search, details, watchlist, news, admin).

- Database access layer (no ORM)

    - Connection factory
    - sql() and small helper functions
    - Uses MySQL parameterized queries only

- Background workers (Celery)
    - Scraping tasks
    - Sentiment analysis tasks
    - Recommendation generation tasks

- AI layer
    - AIService class with pluggable providers (huggingface, openai, gemini, …)
    - Centralized prompts repository
- Business logic services
    - User service, company/asset service
    - Watchlist service
    - Recommendation engine (implements formulas)

4. Data Model & schema.sql

The MySQL schema must match the entities and relationships defined in the project proposal .

4.1 Required Tables

schema.sql must define at least:

1. users
    - user_id (PK, auto-increment)
    - username, email, password_hash, full_name, registration_date, role

2. companies
    - company_id (PK)

    - company_name, company_type, industry, logo_url, founded_date, description, market_cap

3. stock_prices
    - price_id (PK)
    - company_id (FK -> companies)
    - date, open_price, close_price, high_price, low_price, volume, currency

4. financial_statements
    - statement_id (PK)
    - company_id (FK -> companies)
    - statement_type, period_start_date, period_end_date, currency, revenue, profit

5. assets
    - asset_id (PK)
    - asset_name, asset_type, unit_of_measurement, description, logo_url
6. asset_prices
    - asset_price_id (PK)
    - asset_id (FK -> assets)
    - date, price, currency

7. scraped_content
    - content_id (PK)
    - company_id (FK -> companies, nullable)
    - source_url, title, content_text, content_type, scraped_date, publish_date, author, source_name

8. sentiment_analysis

- sentiment_id (PK)
  content_id (FK -> scraped_content, unique)
- sentiment_score, sentiment_label, confidence_level, analysis_date

9. investment_recommendations
   - recommendation_id (PK)
   - company_id (FK -> companies, nullable)
   - asset_id (FK -> assets, nullable)
   - recommendation_date, recommendation_type, investment_score, risk_level, expected_return, rationale_summary

10. bookmarks
    - bookmark_id (PK)
    - user_id (FK -> users)
    - company_id (FK -> companies, nullable)
    - asset_id (FK -> assets, nullable)
    - bookmark_date, notes

11. (Optional) settings
    - For runtime config like active AI provider if wanted.

Indexes must be added for main lookup columns such as company_name, asset_name, date fields, and foreign keys.

5. Database Access Layer (No ORM)

5.1 Connection Management

- Use a connection pool compatible with MySQL (e.g., mysqlclient or aiomysql).

- Expose a FastAPI dependency to get a DB connection per request.

5.2 SQL Utility Functions

Core contract:

```
def sql(db, query: str, params: tuple = ()) -> list[dict]:
    """Execute SELECT / INSERT / UPDATE / DELETE using parameterized SQL."""
```

Recommended helpers:

- `fetch_one(query, params) -> dict | None`
- `fetch_all(query, params) -> list[dict]`
- `execute(query, params) -> int` (returns affected rows / last insert id)
- `transaction()` context manager or decorator

All modules must use these utilities; no direct cursor use outside the DB layer.

6. Core Backend Functional Areas

6.1 Authentication & User Management

- Register (GET/POST): create user with hashed password (bcrypt).
- Login (GET/POST): validate credentials, issue session (cookie or token).
- Logout (POST/GET): clear session.
- Backend must supply the current user to templates.

6.2 Page Routes (Jinja2 Rendering)

Routes returning HTML must exist for:

- `/` – landing + search bar

- /auth/login
- /auth/register
- /dashboard – user summary (counts, recent recommendations)
- /search?q=... – companies/assets list
- /company/{company_id} – full company view
- /asset/{asset_id} – full asset view
- /watchlist – list + note editing
- /news – scraped content + sentiment labels
- /admin – basic system/admin panel

Each must pass the data required by the frontend SRS into templates (user info, lists, error messages, etc.).

6.3 JSON APIs (optional/auxiliary)

Provide simple JSON endpoints that mirror key data for dynamic components:

- /api/search
- /api/company/{id}/prices
- /api/asset/{id}/prices
- /api/company/{id}/recommendation
- /api/bookmark/add and /api/bookmark/remove

These use the same underlying SQL utilities and data model.

6.4 Watchlist Logic

- Add/remove bookmarks based on (user_id, company_id/asset_id).
- Update notes.
- Ensure a user cannot duplicate the same bookmark.

7. Scraping & Background Tasks (Celery)

7.1 Celery Integration

- A celery.py module must define and configure the Celery app.
- Broker: Redis or RabbitMQ (configurable via env).
- Celery workers run in separate processes from FastAPI.

7.2 Scraping Tasks

- Task: scrape_sources()
- Runs periodically (via Celery Beat or external scheduler).
- For each configured source, fetch HTML/API, extract fields, and upsert into scraped_content.
- Avoid duplicates based on (source_url, publish_date).

7.3 Sentiment Analysis Tasks

- Task: analyze_new_content()
- Selects scraped_content rows with no corresponding sentiment_analysis.

For each, calls:

```
 ai_service = AIService()         # default huggingface
result = ai_service.classify_sentiment(content_text)
```

- Writes results into sentiment_analysis.

7.4 Recommendation Tasks

- Task: update_company_recommendations()

- Task: update_asset_recommendations()
- Use price series, financials, and sentiment to compute scores per formulas in proposal (price score, financial score, sentiment weighted score, etc.).
- Store output in investment_recommendations.

7.5 Scheduling

Typical schedule (can be configuration driven):

- Scraping: every 2–6 hours
- Sentiment: hourly
- Recommendations: daily

Tasks should use retries & logging.

8. AI Layer & Prompts

8.1 Central Prompt Storage

- All AI prompts must be in one universal place, e.g.:
    - /prompts/__init__.py exposing constants, or
    - prompts.yaml loaded at startup.

Examples:

- PROMPT_SENTIMENT_CLASSIFICATION
- PROMPT_SUMMARIZE_RATIONALE

No hard-coded string prompts inside provider classes or tasks; they must reference this central store.

8.2 AIService Interface

Single class used by the rest of the backend:

```
ai_service = AIService(model="openai")   # explicit provider
ai_service = AIService()                 # default to huggingface
result = ai_service.classify_sentiment(text)
```

Constructor requirements

- model is a string key: "huggingface", "openai", "gemini", …
- If model is omitted, AIService must default to "huggingface".

Methods

classify_sentiment(text: str) -> dict must return:

```
 {
   "sentiment_score": float,      # -1 to 1
   "sentiment_label": "positive" | "negative" | "neutral",
   "confidence": float            # 0–1
}
```

- (Optional extensions) e.g. generate_summary(text: str) — if used, same pattern: common signature across providers.

8.3 Internal Provider Implementations

Internally, AIService must dispatch to provider adapters:

- HuggingFaceProvider (default)
- OpenAIProvider
- GeminiProvider

Each provider:

- Reads API keys from environment/config.
- Uses shared prompts from the central store.
- Maps raw provider output into the internal result dict (sentiment_score, sentiment_label, confidence).

Provider logic must be isolated so adding a new provider means:

1. Implementing a new class.
2. Registering it in a mapping inside AIService.
3. No changes required in Celery tasks or routes.

9. Recommendation Engine Logic

9.1 Inputs

Per company:

- Last 60 days of prices (for rolling averages).
- Last 2 financial periods (revenue/profit).
- Last 30 days of sentiment scores & confidence.

Per asset:

- Price history
- Sentiment (no financials)

9.2 Business Logic

Backend must implement the formulas described in the project proposal for:

- Price score (Ps)
- Financial score (Fs)
- Sentiment score (Sc) and average confidence
- Investment score (Is) with and without AI

- Recommendation type (Invest / Don't Invest / Hold / Wait)
- Risk level (Low / Medium / High)

Outputs are persisted to investment_recommendations and surfaced to templates and APIs.

10. Non-Functional Requirements

10.1 Security

- Passwords hashed with bcrypt or equivalent.
- All SQL must be parameterized.
- Sessions/tokens must be signed and tamper-proof.

10.2 Performance

- Use proper indexes on main search and join fields.
- Celery tasks must be idempotent where possible.
- Long-running operations must always be run in Celery, not in HTTP requests.

10.3 Reliability & Observability

- Log to file/STDOUT with at least INFO and ERROR levels.
- Celery tasks must log failures and retry.
- Optional: task_logs table for monitoring.

10.4 Extensibility

- Adding a new AI provider must not require changes in tasks or route handlers; only inside AIService/provider mapping.
- New pages or APIs should reuse shared services and SQL utilities.

11. Expected Backend Deliverables

1. FastAPI application code with routes for all pages & APIs.
2. schema.sql containing full MySQL schema.
3. DB utility layer (connection.py, sql_utils.py).
4. Jinja2 templates (frontend SRS already lists pages).
5. Celery configuration and task modules for scraping, sentiment, and recommendations.
6. AI layer implementation (AIService + provider classes) using centralized prompts.
7. Configuration via environment variables or config file (DB, AI keys, broker URLs, default AI model).
8. Minimal admin page to show system status (optional AI provider selector if desired).

12. Docker & Local Development (Updated for uv)

12.1 Docker Compose Requirements

The repository must include a docker-compose.yml file that defines exactly:

1. MySQL 8

2. RabbitMQ (with management UI)

No other services should be included in compose.
FastAPI, Celery, Celery Beat must run locally via uv, not inside Compose.

12.2 Backend Developer Environment (uv + Python 3.13)

The backend is developed and executed locally using:

- Python 3.13
- uv as:
    - package manager
    - environment manager
    - runner