Senior Design Project

InvestmentHelper-AI

Project Specification Report

M. Onur Özdemir, Hakan Muluk, Emir Şahin Dilli

Supervisor: Özgür S. Öğüz
Innovation Expert: Murat Kalender

1. Introduction

Making accurate and reliable financial decisions involves following financial news, analyzing events that lead to appreciation or devaluation of stocks, and taking the initiative before the market fully adjusts to an event. However, achieving all of these prerequisites in the current settings is not easy.

Manually monitoring and analyzing financial news is challenging, as it requires sustained attention, while modern-day environments often divide one's attention across numerous tasks throughout the day. Yet, the financial markets adjust rapidly, and to gain an advantage regarding personal portfolios, one needs live, real-time knowledge of events that can affect the markets.

Our project aims to provide users with a financial assistant that monitors financial news, analyzes events, and delivers real-time notifications about events that users want to be informed about. The assistant will make it easy for users to follow financial news and gain live, instant knowledge of market-moving events. The assistant, or more formally, the chatbot, will interact with users and support conversational information exchange, referring to financial news stored within the knowledge base. Additionally, users will be able to query the context of important events, upon which relevant notifications will be sent in real-time. Financial news will be analyzed continuously for accurate and up-to-date analysis. Furthermore, AI-driven analysis will be integrated to automatically detect trading patterns, such as head-and-shoulders patterns, double tops, and more, providing advanced insights to help investors make informed decisions based on chart pattern recognition. Additionally, AI-based stock predictions will offer users data-driven forecasts, enhancing their ability to anticipate market movements and make proactive investment decisions. The system will also compare the latest trends of a stock with its past trends, taking into account historical news and the financial environment to identify the closest matching periods from the past. This feature provides users with a comprehensive perspective on historical market behavior, helping them gain a clearer understanding of potential future trends.

1.1 Description

InvestmentHelper-AI is a system that assists users regarding financial decisions. It is essentially a chatbot that answers questions about finance, whether general or specific. It will also support real-time, live notifications about events that the user enters.

Example:
Prompt: Tell me when X happens in BIST.

The web is scraped regarding financial news and when a relevant occurrence has been detected, the user will be notified about the event. This will happen in real-time. That is, financial news will be scraped and processed continuously. Hence, the user will be swiftly

notified about the event, a small amount of time after it happens, giving the user an initiative. Also, the prompt that the user can enter is not heavily constrained, as it is in natural language, making it possible to capture continuous contexts about the requests of users.

InvestmentHelper-AI is a time-dependent system, more formally, it is a causal system. Financial information present within the knowledge base might become irrelevant with time, in the event of receiving more recent news that overrides the effects of the previous one. In such cases, only the most recent, up-to-date information is processed and retrieved. Hence, InvestmentHelper-AI is not constrained to non-time-dependent questions regarding finance; time-dependent questions can also be asked, and the system can be expected to work reliably.

Hence, the chatbot will introduce an interactive environment, upon which the user can learn up-to-date information about a financial event or company and be notified about events that are important for their prospective financial decisions.

The assistant, that is, the chatbot, will utilize Advanced Retrieval Augmented Generation (RAG) pipelines for correct behavior. The techniques we have derived will allow users to ask comparative and multi-hop questions to the chatbot regarding the market. The users will also be able to ask “generic” questions about the markets or certain companies. The chatbot will support such different kinds of questions, which will be explained below.

Note: The advanced RAG techniques that we propose will be benchmarked, and allow users to ask “difficult” questions and receive correct answers. In this regard, the project involves academic studies and combinations of academic natural language processing work.

Multi-hop Question:
Which companies that work with X company also have business in the energy sector?

The question requires information from different parts of the knowledge base, namely, companies that work with X company and companies which are in the energy sector.

Comparative Question:
What are the top 5 companies with the highest profits in the last year?

This requires different information chunks from various sources, and the chatbot also needs to perform computations on them, like sorting.

Generic Question:
How is X company doing?

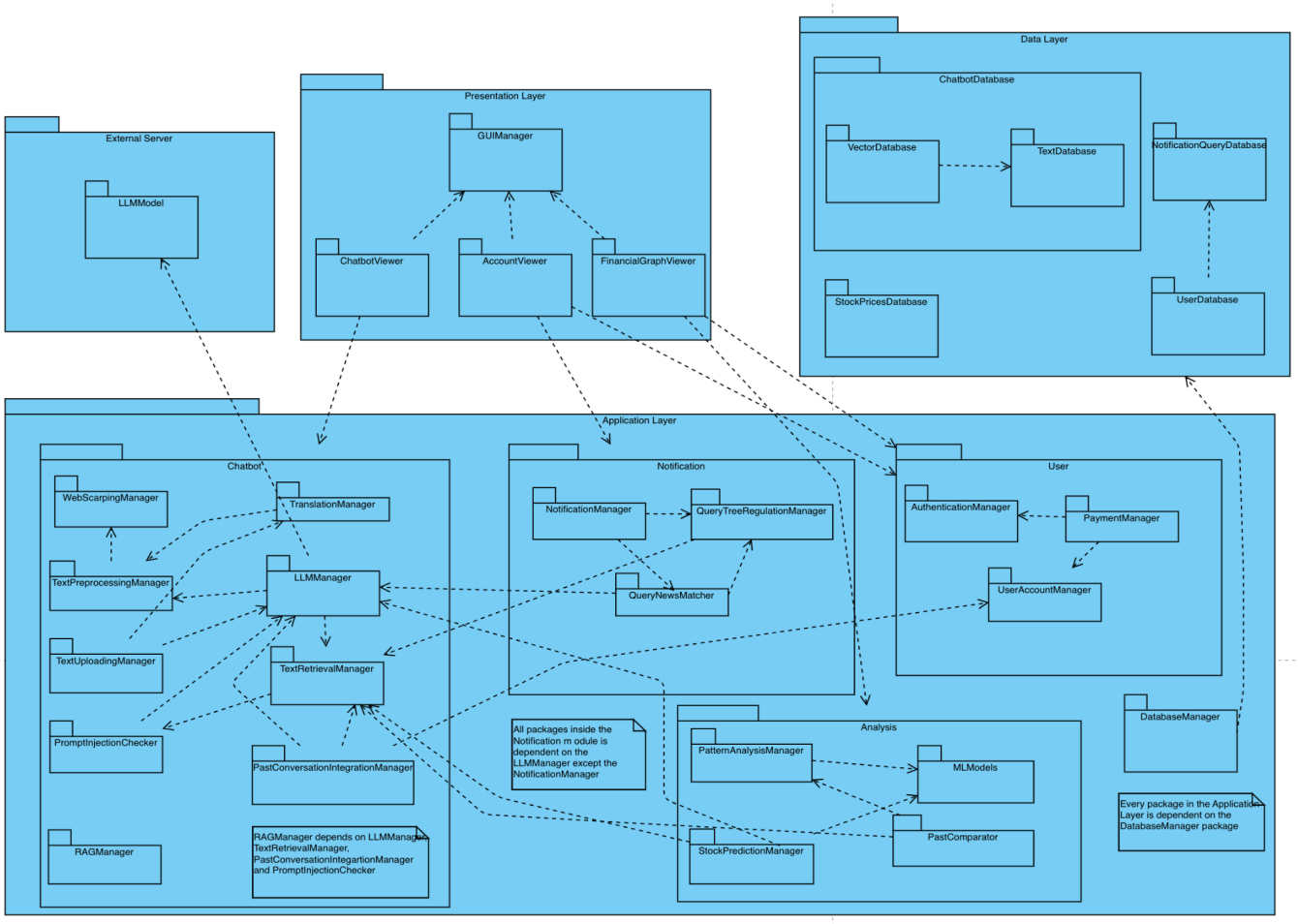
The chatbot will return general information, with emphasis on more recent news about the company, and answer such questions in a generic manner.

Additionally, users will have access to financial charts where they can perform technical analysis on stock prices, compare different stocks, and analyze market trends using visualization tools. This feature enhances their ability to make informed decisions by

providing comprehensive charting and comparison capabilities. AI-driven analysis will also be integrated to automatically detect trading patterns, such as head-and-shoulders patterns, double tops, and more, providing advanced insights to help investors make informed decisions based on chart pattern recognition. AI-based stock predictions will further offer data-driven forecasts, improving users' ability to anticipate market movements and make proactive investment decisions. Finally, the system will compare the latest trends of a stock with its past trends, taking into account historical news and the financial environment to identify similar periods from the past, thereby offering users a comprehensive perspective on historical market behavior to better understand future trends.

Note: Our prioritized features are the chatbot, notification and chart displaying features. Other features mentioned are the optional ones, this is also discussed with the course instructors and has been approved by them.

1.2 High Level System Architecture & Components of Proposed Solution



The link to the high level system architecture:
https://online.visual-paradigm.com/share/book/hla-1ythjbdga2

1.2.1 Presentation Layer

- **GUIManager:**
  This package is responsible for managing the ChatbotViewer, AccountViewer and FinancialGraphViewer packages and the interactions between them.
- **ChatbotViewer:**
  This package is responsible for the interactions between the chatbot and the user. It will display the current and past steps while generating an answer to the user query, user's past interactions with the chatbot in the same chat and the different chats. Lastly, it is responsible for displaying the references(the documents) that are used to generate an answer to a user query.
- **AccountManager:**
  This package is responsible for displaying the account details to the user. These details include the notification queries that the user entered, their status, user's remaining balances etc.
- **FinancialGraphViewer:**
  This package is responsible for displaying the current and past stock prices of a selected company to a user, in a chart format. Also, it will display the comparisons of different stocks and different indicators with respect to a stock. Finally, the users will be able to do technical analysis on these charts(draw lines and do some computations on these charts).

1.2.2 Application Layer

- **Chatbot Module:**
  This module consists of packages that will be used in the chatbot of the project, they are grouped in this module for clarity.
- **WebScrapingManager:**
  This package is responsible for scraping some websites and extracting the content of those, such as KAP notifications or different financial news websites. The contents will be extracted from the HTML files, will be preprocessed and be used as data to the chatbot. Also; for the real time notifications feature, the web scraping will be done in a high frequency in order to decrease the latency.
- **TextPreprocessingManager:**
  This package is related with the preprocessing of the data extracted with the WebScrapingManager. It will do chunking on long texts(for RAG purposes) and will convert the tables in the texts extracted into Markdown format. It is dependent on the WebScrapingManager since the data that will be preprocessed will be supplied to this package from the WebScrapingManager package.
- **TranslationManager:**
  This package's role is to coordinate the translation of scraped texts and user queries to English, and the final output of LLM back to Turkish(if the user's language is Turkish). The reason for these translations is to increase the performance of internal steps; LLMs work better in English since their training data mostly consists of

English texts, the retrievers also work better in English so we decided to add these translation steps.

- **TextUploadingManager:**

  This package is responsible for controlling the upload of data to the databases. Some texts will be paired with the summaries of their originated document for better retrieval accuracyas in [1], some texts will be stored in Knowledge Graph triplets format but also be paired with the original texts and all the texts will be converted to embeddings for retrieval purposes. This package will control these details by cooperating with the LLMManager package so it is dependent on the LLMManager package. Also, it will use the translated texts as inputs so it is additionally dependent on the TranslationManager package.

- **TextRetrievalManager:**

  This package is responsible for retrieving the related texts. However, there are different techniques used in different stages for retrieval. For example, the retrieval might not be solely based on the cosine similarity between the query and the text embeddings but might be a function of also other embeddings, such as the document summary embedding or so. The total retrieval score can be calculated by summing different similarity scores in a weighted manner, and retrieval might be done by reranking based on the total retrieval scores. This package will control these parts according to different stages of the query generation process. Lastly, this package is dependent on PromptInjectionChecker since first the user queries will be checked against prompt injection and only if it passes this check, the retrieval will be done.

- **PromptInjectionChecker:**

  This package's role is to check the user queries to prevent any prompt injection attack that might cause an information leak in the system. This will be done by using an LLM that is specialized on this domain so this package is dependent on the LLMManager.

- **PastConversationIntegrationManager:**

  This package is responsible for integration of past conversations between the user and the chatbot to a new query so that the chatbot will remember the past conversations and answer accordingly. This will enable a coherent conversation in which the chatbot can answer the follow-up questions that are asked by the users. This package will use the TextRetrievalManager in order to get the relevant past conversations and will use the LLMManager for aggregating the past chats and the current query, so it is dependent on them. Lastly, it also depends on the UserAccountManager since it will use the past chat of the user.

- **RAGManager:**

  This package is responsible for arranging the steps for generation of an answer to a user query. It will manage the other packages so that the steps of generation will be finished in a sequential order where parallelization can be used whenever

applicable. For example; when a user enters a query, it will first be checked against prompt injection, after that past conversations will be aggregated and the query will be transformed if necessary, after that the question type will be determined and other steps will be executed according to the type of the question… This package will arrange these steps and will control the general flow for queries. It depends on LLMManager, TextRetrievalManager, PastConversationIntegrationManager and PromptInjectionChecker.

- **LLMManager:**
  This package handles all interactions with the LLMs that will be used, it will organize the prompts and LLM configurations(such as temperature, top_p etc.) for each task and the LLM that will be used for that task. It is dependent on the TextPreprocessingManager since it will use the preprocessed texts, and the TextRetrievalManager because of the usage of retrieved texts in order to generate answers. Lastly, it is also dependent on the LLMModel package which will operate on an external server.

- **Notification Module:**
  This module consists of the packages that will be used for the notification system of the project, they are grouped inside this module for clarity.

- **NotificationManager:**
  This package is responsible for notifying the users if their notification queries are satisfied. This means that if the user has entered a query and while checking a news item for each query, if the thing mentioned in the query happened, a notification will be sent to the user. This package is dependent on the QueryTreeRegulationManager and QueryNewsMatcher.

- **QueryTreeRegulationManager:**
  This package is responsible for creating and maintaining a search tree based on the user queries. In order to avoid checking every news item for every notification query, a search tree will be created based on the user queries, similar to the one in the RAPTOR paper [2]. By constructing this search tree, LLM will start from the root node and traverse the tree all the way down to the leaves, for each news item. So the LLM calls needed will be reduced and notifications will be sent to the users whose notification queries are matched with the news item. This package is dependent on the TextRetrievalManager.

- **QueryNewsMatcher:**
  The role of this package is to traverse the tree starting from the root, it will use the LLMManager in order to decide which child node to select and continue searching. Also, when it reaches a leaf node(the original notification query), it will check if the current news item that is being analyzed is suitable for the notification

query by using the LLMManager. If the news item is suitable, the news item will be summarized and the user will be notified via the NotificationManager.

- **Analysis Module:**
  This module is the collection of packages that are related with stock price prediction, trading chart pattern analysis and comparison of today and past in terms of graphical and financial environment similarity.

- **PatternAnalysisManager:**
  This package is responsible for identifying the patterns in the stock data, such as head & shoulders and so. The package is dependent on the MLModel package because it will use the machine learning models to identify the trends.

- **StockPredictionManager:**
  This package is responsible for making predictions on the future stock prices, for different companies. The package is dependent on the MLModels package since the machine learning model trained is in the MLModel package. It is also dependent on the TextRetrievalManager and LLMManager because it will use the news in order to create some features that will be used in the prediction.

- **PastComparator:**
  This package is responsible for comparing the past and today for each company, and to find some time intervals where the graph and financial environment of the company is similar to today's conditions. It is dependent on the TextRetrievalManager since it will use the financial texts and compare their similarities, and it is also dependent on the PatternAnalysisManager for analyzing the similarities of patterns.

- **User Module:**
  This module consists of the packages that are related with the core functionalities of the project, such as authentication or payment functionalities. The packages are grouped inside this module for clarity.

- **AuthenticationManager:,**

  Responsible for managing user authentication, including verifying login protocol and handling multi-factor authentication. The system guarantees secure access by preventing unauthorized logins and maintaining session integrity.

- **UserAccountManager:**

  Manages user account details and settings, such as profile information, changes in password or mail, and recovery options. It also monitors user activity within the system to enhance personalization, and provide insights for improving user experience.

- **Payment Manager:**
  Handles payment-related functionalities, such as processing transactions, managing payment methods, and ensuring secure handling of financial data.

- **DatabaseManager:**
  This package is responsible for arranging all the interactions with different databases in the Data Layer. All packages in the Application Layer are dependent on this package since it will orchestrate all the interactions with the databases.

1.2.3 Data Layer

- **TextDatabase:**
  This database is responsible for storing the text data that is collected via web scraping, it will store both the preprocessed and original versions of the data. It will also store the summaries created, Knowledge Graph triplets etc.

- **VectorDatabase:**
  This database is responsible for storing the vector embeddings of the preprocessed texts, these vectors will be used for retrieval purposes by doing a cosine similarity search. It is dependent on the text database since it will store the embeddings' of the texts that are stored in TextDatabase.

- **StockPricesDatabase:**
  This database will store the past data of stocks, such as stock prices, volatility etc, for each company that will be in our system.

- **UserDatabase:**
  This database will store the necessary information of users. It will store the authentication info, users' past chats with the chatbot etc.

- **NotificationQueryDatabase:**
  This database is responsible for storing the users' notification queries and also the tree structure that is mentioned above. It is dependent on the UserDatabase since it needs to associate each user's notification queries with their corresponding user account. This ensures that notification queries are correctly linked and managed for individual users, enabling personalized notifications and efficient query tracking.

1.2.4 External Server Layer

- **LLMModel:**
  This package consists of the LLM models that will be used in our application, some of the models will be run in a server that we rent and some of them will be run in the servers of other companies(such as OpenAI).

Note: All packages in the Application Layer are dependent on the DatabaseManager, so these dependencies are not mentioned above, to avoid repetition and maintain clarity in illustrating the relationships between other components.

1.3 Constraints

1.3.1 Implementation Constraints

The following LLMs will be used in this project: GPT4o-mini, Llama 3.1-70b, and Llama 3.2-1b. Except for the GPT4o-mini, these models will be deployed on four NVIDIA A40 GPUs provisioned via Runpod. The LangGraph framework will be used to build a strong state space in the RAG system for enhancing the capability of the chatbot in managing complex queries.

Implementation will be done in Python and JavaScript, following the OOP principles for both modularity and maintainability. The project platform is web-based, thus it will be accessible with the use of web browsers. FastAPI will be used on the back-end. React will be used on the front-end except for the Chatbot and financial chart components. Chainlit will be used for the interface of the Chatbot. Lightweight Charts will be used for the generation of financial graphs.

Financial data, including but not limited to stock price information, will be gathered from the Yahoo Finance API. Financial news articles will be retrieved from reliable online sources, such as Kamuoyu Aydınlatma Platformu-KAP.

Segments extracted from financial news articles will be embedded into vectors to facilitate efficient retrieval and similarity searches within the RAG pipeline. We will use E5-large-v2 as the embedding model. In light of the potential storage challenges associated with substantial volumes of vector embeddings and graph data, we will employ indexing techniques within Neo4j. In particular, we will establish vector search indexes in Neo4j to enhance retrieval speed while facilitating effective storage management. Using Neo4j further aims at incorporating a knowledge graph so that multi-hop and comparative questions can be answered. So, vector embeddings and texts will be kept in Neo4j in a knowledge graph format.

User information, along with helper information like past chat history, will be maintained employing MongoDB, which belongs to the class of No-SQL databases.

In the development process, the work will be guided by Object-Oriented Programming principles to enhance code reusability and scalability. Version control of code will be carried out using Git, whose repositories will be stored on GitHub for ease of code development and change history. In regard to task distribution and project management, Jira will be used.

API service deployment on the web will be done using either Heroku or Amazon API Gateway.

1.3.2 Economic Constraints

For data preprocessing, we will use the GPT4o-mini Batch API to convert scraped texts into a knowledge graph. The cost associated with this service is $0.075 for every 1 million input tokens and $0.300 for each 1 million output tokens [15]. Considering the significant volume of data that requires preprocessing, it is imperative to minimize token usage in order to manage expenses effectively. We will use efficient data processing techniques, like input batching and prompt engineering, to minimize the number of tokens used during pre-processing.

For live interaction with users through a chatbot, we might use GPT4o-mini(there will be many LLM calls, some of the calls might be done to this model). Pricing is as follows: $0.150 per one million input tokens and $0.600 per one million output tokens [15]. High user engagement might be costly for token consumption. To control this cost, we will tune the chatbot responses for brevity and relevance so that the number of output tokens will be as small as possible without any loss of functionality in the interactions.

We will be renting four NVIDIA A40 GPUs from Runpod, to be used for model deployment(the Llama models) and inference at a runtime cost of $1.50 per hour of use [16]. Because a continuous usage of GPUs may amount to very high costs, we will manage the cost by planning GPU activities at times of high user end demand and then turning the GPUs off in less busy periods. In addition, we will implement model optimization techniques to reduce computational burden: model quantization and pruning to increase efficiency and reduce the time that the GPU needs to be on.

AWS API Gateway is free for 1 million requests per month for 12 months, which will keep our API for free during the development and testing period [17]. Since the AWS service is free within these limits, they won't contribute to the costs of the project for the first year. We will monitor our usage to make sure we don't go over the free tier limits and prepare for scaling costs if we think we might go over.

Heroku is used as a web server, with a free tier available that is suitable for a single project. This will enable hosting of our application without additional hosting costs when developing. We will monitor our usage to ensure the free tier is sufficient for our needs and consider an upgrade to a paid plan if needed as the project grows.

We will use Neo4j locally during development and Neo4j AuraDB Free tier for inference, based on the suitability of our project needs [18]. The AuraDB Free tier allows small applications; however, if our needs are much more than the free tier offerings, we will have to

move to AuraDB Professional, which incurs a $65 monthly expense. This release will also bring improved performance and scalability in order to handle larger datasets and more complex queries.

Similarly, we will be using the local version for MongoDB during development and MongoDB Serverless while doing inference . The serverless costs $0.10 per 1 million reads [19]. We will need to keep a close eye on our reads to keep our costs in check. Using the serverless option means we are only paying for resources used, rather than provisioning up-front resources, potentially more economical for workloads that have variable or unpredictable traffic patterns.

1.3.3 Ethical Constraints

The system was built around strict ethical principles to protect user privacy, ensure fairness, and comply with regulations. Any monitoring of user activity for personalized recommendations was done openly, with users giving clear consent beforehand. They were told exactly what data was being collected, why it was being collected, and how it would enhance their experience. Sensitive details, like financial account information or private transactions, weren't tracked, and all collected data was anonymized where possible. Users also had the choice to opt out of monitoring without losing access to the system's main features. The approach to data collection was minimal—only gathering what was truly necessary.

For web scraping, the system only used reliable and publicly available sources to make sure the information provided was accurate and trustworthy. Platforms like official stock exchange websites and other credible financial resources were prioritized. To maintain transparency, the chatbot clearly showed where its information came from in its responses. Rate-limiting and timed delays were built in to avoid overwhelming the websites being scraped, and the system steered clear of anything behind login walls or that required credentials.

User data was handled with robust security measures. All information was encrypted during transmission and while being stored, using AES-256 for storage and TLS 1.3 for transmission. Data was stored in Turkey to meet KVKK requirements, with access strictly limited to authorized staff. Detailed logs were kept to monitor access and ensure everything stayed secure. No user data was ever shared with third parties without explicit permission, and users were given full control to delete their data if they wanted.

The system's recommendations were designed to be neutral and unbiased, with clear explanations provided to help users understand the insights. Information was always pulled from up-to-date and verified sources to maintain accuracy. Misleading or harmful content was actively filtered out to ensure users only got reliable, actionable advice. Safeguards were also in place to block fraudulent activities or market manipulation, ensuring the system stayed fair and trustworthy.

1.3.4 Legal and Social Constraints

InvestmentHelper-AI provides financial data, forecasts, and analysis that are meant to help users make better decisions. However, to comply with legal requirements and avoid the unlawful practice of providing financial advice, all insights and predictions offered by the system will be accompanied by a clear disclaimer stating that they do not constitute financial advice. The users will also be advised to consult a licensed financial advisor before making any investment decisions based on the information received from this website.

In order to accommodate and be inclusive of the diversity of our user base across different ethnicities and regions, InvestmentHelper-AI supports multiple languages. Currently, the system is supporting Turkish and English, though it will be quite easy to expand to many more. The internal operations of the chatbot are conducted in English. To support users in different languages, the documents and user inputs are translated to English using LLMs. Regardless of the starting language, user input is converted to English for processing. After a response has been generated, the output-first in English-is then translated back to the user's original language before being shown to them.

Since the translation depends on large language models, any language that can be translated into English using these models is eligible to be integrated into the system. This approach allows InvestmentHelper-AI to handle a great variety of languages and thus be accessible to users worldwide; it is assured that the language barrier will not impede access to important financial information.

1.4 Professional and Ethical Issues

Monitoring user activity for compliance or personalization can raise significant privacy concerns. If users are unaware of being monitored or if the scope of monitoring is excessive, it infringes on their autonomy and trust. Ethical constraints ensure that monitoring is limited to necessary actions, conducted transparently, and anonymized wherever possible to prevent misuse or overreach.

User account details, such as passwords or profile information, are sensitive and require protection against breaches or unauthorized access. Without robust encryption and access restrictions, this data could be exposed, causing harm to users.

Data collected during account management or monitoring could be misused for unauthorized profiling or marketing. This misuse violates user trust and ethical standards of data minimization. Constraints ensure that data is used only for its intended purpose, such as improving user experience, and only with explicit user consent.

Users have a right to know how their data is monitored and used. Lack of transparency undermines trust and violates principles of informed consent. Ethical constraints require clear communication about monitoring practices, ensuring users are fully aware of what data is collected and why.

Using data from external sources or user-generated content raises potential intellectual property concerns. Ensuring that the system respects data ownership rights while providing reliable functionality is a critical professional responsibility.

Building a complex system that involves monitoring, secure data handling, and compliance with regulations requires diverse expertise. Addressing gaps in the team's technical or legal knowledge is essential to ensure professional execution of the project.

Prompt injection risks and the accuracy of prompts are key concerns for maintaining reliable outputs. Improperly constructed prompts or vulnerabilities in the prompt-handling process can lead to misleading responses, compromising the system's reliability and user trust.

Building a robust model to ensure the LLM provides highly accurate answers is a critical issue. Without a well-optimized retrieval and generation pipeline, the system risks delivering incomplete or irrelevant information, undermining its reliability and user confidence.

1.5 Standards

- Use of AES-256 encryption for data storage.
- TLS 1.3 for secure data transmission.
- WCAG compliance for accessibility, including text scalability and color contrast ratios.
- Adherence to KVKK regulations for data protection in Turkey.
- OAuth 2.0 for secure authentication.
- Compliance with international data handling and security guidelines.
- Automated backups every 6 hours for redundancy, ensuring RPO and RTO standards are met.
- Role-Based Access Control (RBAC) for user access management.
- Secure password storage using hashing algorithms like bcrypt.
- Compliance with GDPR principles for data protection outside Turkey.
- Ethical web scraping practices to respect copyrights and intellectual property laws.
- Transparent sourcing of financial data, with citations for news and reports.
- Respect for licensing agreements when integrating third-party tools, libraries, and APIs.
- Proper attribution for academic and external resources referenced in the development.
- Use of open-source software licenses (e.g., MIT, Apache) where applicable, ensuring compliance with licensing terms.
- Compliance with **robots.txt** directives to avoid scraping restricted sections of websites.

2. Design Requirements

2.1 Functional Requirements

The user can:

- Ask questions to the chatbot about general topics/themes about Borsa Istanbul.
- Ask about specific companies by asking the chatbot for information (e.g., "Tell me about Company X").
- Receive up-to-date financial information via chatbot unless they request past information.
- Ask complex multi-hop questions that require information from multiple news events or sources.
- Ask comparative questions that require information from multiple news events or sources, and require computations on them.
- Set up personalized notifications by entering natural language prompts specifying the financial events they are interested in (e.g., "Notify me when Company Y's stock price drops below $50").
- Receive real-time notifications when important financial events occur that match their specific interests, along with the summary of the financial event.
- Visualize stock data using different types of charts
- Perform technical analysis by applying indicators and drawing tools on stock charts.
- Save past chat interactions with the chatbot for future reference.
- Create new chats that are independent of others.
- Customize notification preferences by setting the notification method as mail or SMS.
- Export data and charts for personal use or further analysis in formats like CSV or PNG.
- Create and manage a personal watchlist of stocks or companies.
- Set up multi-factor authentication for enhanced account security.
- Request personalized recommendations based on their interests or past interactions
- Enable dark mode to enhance user experience.
- Allow users to define and save their own indicators for technical analysis.
- Add advanced filtering options for visualized stock data (e.g., by market cap, sector, or performance).
- Enable the chatbot to search for specific timeframes (e.g., "Show me the stock performance of Company X in Q1 2023").
- Ensure user data, settings, and saved chats sync seamlessly across devices.
- Allow users to upload their own datasets (e.g., CSV files) for personalized analysis or insights.
- Show real-time updates for breaking news or significant stock movements in an interactive timeline.

The below ones are optional functional requirements:

- Analyze detected trading patterns such as head-and-shoulders patterns and double tops identified by the AI on stock charts.
- Access stock predictions generated by machine learning models based on past stock prices and financial news.
- Analyze historical financial news and environments to find time intervals similar to the present day.
- Access educational resources or tutorials to learn more about financial concepts and technical analysis.
- Schedule periodic reports or updates to be sent at regular intervals (e.g., daily market summaries).

2.2 Non Functional Requirements

2.2.1. Usability

- The design must require no more than **30 minutes of training** of users for basic operations.
- All features must be **accessible within three clicks** from the main interface.
- Support both **dark and light themes** to accommodate various working environments.
- Implement **keyboard shortcuts** for common operations to enhance efficiency.
- Ensure **consistent navigation patterns** across all screens.
- Provide **clear, concise, and actionable error messages** to avoid user confusion.
- Offer **real-time feedback** for user actions with a maximum response time of **0.5 seconds**.
- Support **English and Turkish languages** with seamless switching functionality.
- Ensure text is **scalable up to 200%** without loss of functionality or readability.
- Meet **minimum color contrast standards (4.5:1)** for accessibility.
- Provide **contextual help and tooltips** for all features to guide users.
- Include **interactive tutorials** to assist with user onboarding.

2.2.2. Reliability

- Maintain **99.9% uptime** during trading hours.
- Conduct planned maintenance **outside trading hours** to minimize disruptions.
- Implement **automatic failover mechanisms** for seamless switching to backup systems during failures.
- Limit maximum allowable downtime to **43.2 minutes per month**.
- Ensure all financial transactions adhere to **ACID principles** for reliability and correctness.
- Maintain **100% accuracy** in financial calculations.
- Automate data backups every **6 hours** to prevent data loss.
- Use **data verification mechanisms** to validate all user inputs.
- Automate backups every **6 hours** for redundancy.
- Recovery objectives:
  - **RPO (Recovery Point Objective):** Maximum data loss of **1 hour**.
  - **RTO (Recovery Time Objective):** Recovery within **15 minutes**.
- Maintain at least **three backup copies** in geographically distinct locations.
- Provide highly accurate responses to financial questions:

    - **General Questions:** Example: “What is the current stock price of XYZ?”
    - **Multi-Hop Questions:** Example: “What are the quarterly earnings of ABC, and how do they compare to last year’s performance?”
    - **Comparative Questions:** Example: “Which stock performed better this week, ABC or DEF?”
- Use reliable data sources and advanced algorithms to ensure query accuracy.

2.2.3. Performance

- **Chat Response Time:** Standard queries processed within **10 seconds**.
- **Multi-Hop Query Response Time:** Complex queries resolved within **20 seconds**.
- **Real-Time Data Updates:** Maximum latency of **100 milliseconds** for updates like stock prices.
- **UI Interaction Response Time:** Actions such as button clicks should respond within **500 milliseconds**.
- Support **100 concurrent users** without performance degradation.
- Process at least **1,000 queries per minute**.
- Perform **10,000 web scraping operations per hour** for data updates.
- Support **1,000 simultaneous notification checks** for user-defined events.
- **CPU Usage:** Must not exceed **70%** under normal conditions.
- **Memory Usage:** Remain below **80%** of available RAM.
- **Database Connections:** Limit active connections to **75%** of the pool size.
- **Network Bandwidth Utilization:** Keep usage below **60%** of available capacity.

2.2.4. Supportability

- Allow **hot deployments** for updates and patches without downtime.
- Conduct **non-disruptive updates** without system restarts.
- Run **automated health checks** every 5 minutes.
- Enable **automated dependency management** for security and compatibility.
- Implement **real-time monitoring** for all system components.
- Use **automated alerts** to notify the support team of issues.
- Provide a **performance dashboard** with key metrics like CPU usage and query throughput.
- Maintain **detailed logs** for performance analysis and troubleshooting.
- Include detailed **error logs** with timestamps and severity levels.
- Support **remote debugging** for efficient problem diagnosis.
- Provide **query performance analysis tools** to identify bottlenecks.
- Enable **session replay** for reproducing and investigating user issues.
- Support **environment-specific configurations** for development, staging, and production.
- Provide a **user-friendly admin interface** for configuration management.
- Use **version control** to track and audit configuration changes.
- Maintain **audit logs** with timestamps and user details for configuration changes.

2.2.5. Scalability

- The project’s main focus is to satisfy the needs of the BIST investors. However other financial markets can be integrated in the future.

- The current languages supported are Turkish and English but other languages can also be added in the future, the structure of the project is suitable for this(internal mechanism is using English, the user queries, scraped texts or the LLM outputs can be translated back to another language without any effort).
- Allow manual scaling of servers based on workload demands.
- **HAProxy** for cost-effective load balancing.
- Transition incrementally from a monolithic architecture to **microservices**.
- Support manual resizing of cloud instances or VMs for CPU and memory.
- Integrate caching.
- Handle hardware upgrades seamlessly without major redesigns.
- Start with **single-node databases**, scaling to distributed systems as required.
- Use **indexing and query optimization** for efficient execution.
- Implement **data archiving** strategies, e.g., **AWS S3** for cost-effective storage.
- Ensure performance is optimized for the current user base.
- Implement scalable authentication systems like **OAuth 2.0**.
- Begin with predefined user roles, expanding as necessary.
- Allow integration of new data sources with minimal disruption.
- Add additional data sources such as financial news websites without major architectural changes.
- Use standardized mechanisms for **data mapping**.

2.2.6. Security

- **Encrypt sensitive data** during transmission using TLS 1.3.
- **Store user passwords securely** using hashing (e.g., bcrypt).
- Implement **role-based access control (RBAC)** for admin and regular users.
- Use **email or OTP verification** for user authentication.

- Validate and sanitize inputs before passing them to chatbots or AI models.
- Restrict AI responses to predefined contexts, avoiding execution of user-provided commands.
- Ensure backups are **encrypted** and stored locally or with a trusted provider in Turkey.
- Set **automatic session expiration** to prevent unauthorized access.
- Log only **critical events** (e.g., login attempts) without storing sensitive data.
- Use **firewalls** to block unauthorized access.
- Implement **rate limiting** to prevent brute force attacks.

3. Feasibility Discussions

3.1 Market & Competitive Analysis

The market for AI-driven financial tools and chatbot solutions is highly dynamic, with many different platforms offering specific functionalities but often falling short in some critical areas. For instance, FinChat [3] provides notifications; still, the extent and details of this

feature are not very clear. Furthermore, FinChat supports financial charts so users can do technical analyses comparing stocks—efficiently visualizing market trends. Along with this feature comes the chatbot, which can discuss financial information with users and help in basic question management. Still, even with this useful attribute, the platform does not support more advanced tasks in its chatbot, like multi-hop reasoning or comparative question capabilities. This limitation prevents FinChat's ability to synthesize and cross-reference data points in different documents or contexts smoothly, thus restricting users' ability to make deeper, interconnected analyses and make more nuanced, data-driven decisions.

Similarly, AI Ticker Chat [4] has a unique focus on financial analysis while rendering insights derived from financial reports; users can input any financial report (including their business reports) and can chat with the chatbot on them. However, it does not provide notifications, which is doubtless a very important feature that most, if not all, users depend on to get on-time alerts and updates; otherwise, little purpose would be served. It has a relatively narrow focus, which, while favoring in-depth report-related insight, constrains its overall value for broader, ongoing financial analyses and engagement with users.

Uptrends.ai [5] is set apart by its notification system, which provides immediate updates and alerts. It is, however, highly discretized in its contextual analysis—users select among the available notification topics predefined in the system, users cannot enter their own queries—and does not have the fluid, continuous language processing capabilities of more advanced systems. This somewhat limits Uptrends.ai from providing cohesive narratives or analyses that span across various data points for a more comprehensive financial insight. But there is a huge gap in these platforms: they are not specialized enough to deal with companies listed on Borsa Istanbul. In many cases, these companies don't exist in their datasets or are poorly represented. This points out an opportunity missed by those platforms that want to provide localized and specialized financial intelligence about companies listed on Borsa Istanbul. On the other hand, this approach brings much more comprehensive, language-driven context coverage and will enable much more continuous analysis, both tailored to the needs of Borsa Istanbul companies and the in-depth insight-seeking users about them. This specialization positions our system uniquely in the marketplace and fills a need not being satisfied by other alternatives.

Note: Market Analysis is done based on the prioritized features since the remaining ones are not mandatory for this project.

3.2 Academic Analysis

Although the Transformer Architecture was proposed in 2017 [6], it took a while to create sufficiently scaled LLMs that can be used for various kinds of tasks. However, LLMs' capabilities today are more than enough for many tasks and Retrieval Augmented Generation is one of the hottest topics in academia nowadays, especially because of the popularity of LLMs. The main goal of RAG is to provide the necessary information to the Large Language Models to generate a response to a user query by using LLMs' in-context learning

capabilities. However, naive RAG approaches have some limitations, they are not well suited for multi-hop question answering, comparative question answering or generating answers to general questions. Also, there are lots of design choices since the in-context learning capabilities of LLMs are limited to some extent, they cannot process very long texts to produce an answer to a query, as stated in [7]. So, only a limited amount of text should be given to a LLM for RAG purposes, otherwise the answers will not be reliable. Also, LLMs do not know the things after their information cutoff dates and it is not practical to train or fine-tune an LLM for every new information available. In addition to this, LLMs are prone to hallucination and might give wrong answers to user questions. RAG also solves the hallucination problem by giving the correct data to LLM while it is asked to generate an answer to a user query, thus reducing the hallucination problems by using LLMs in-context learning abilities. It is also possible to increase the performance of in-context learning by giving some examples(few-shot prompting), as the LLMs are good at few-shot learning, as stated in Language Models are Few-Shot Learners[8].

Multi-hop question answering requires chatbot systems to collect and analyze different pieces of texts from different documents(chunks), in order to answer a user query. One of the most promising solutions to this problem is offered in the paper IRCoT [9], which makes multiple LLM calls and multiple retrievals for answering a multi-hop question. For example, if the user enters a question such as:
“Who is the most famous soccer player in the country where X was born?”, LLM first generates a RAG query such as: “Where did X born?”, by retrieving the necessary documents for answering his sub-question. After analyzing the retrieved documents, LLM might see that X was born in Y and generate another RAG query for answering the user’s question: “Who is the most famous soccer player in Y?”. However, this solution is slow and is not suitable for real time chatbots since it makes multiple LLM calls in a sequential manner and cannot be parallelized. Another paper that is related with this problem is the HippoRAG paper [10], it creates a Knowledge Graph from the texts, extracts the entities from the user queries and execute the Personalized Pagerank algorithm on the knowledge graph, and retrieve the text pieces according to the resulting scores of entities in the Knowledge Graph, after the PPR algorithm(by using TF-IDF on entities and text pieces). This might seem like a promising solution however, it does not generalize the problem well and is not reliable. PPR might give wrong importance results in the entities and cause wrong retrieval of texts. Comparative questions are similar to multihop questions, different texts are again needed but also some computations on these texts are required to produce the correct answer. For example, a user might ask: “Which company has made the highest profit?”, this question requires the profit of each company, to sort them and give the highest one as the answer. Some solutions such as in the paper Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM [11] creates different database schemas by using the texts available, in the beginning. After that when a user asks a comparative question, the LLM transforms it into a SQL query and tries to answer the question in this way. However, the schemas created might not be reliable or might be incomplete so the answer might not be found. In addition to that, for the generation of the SQL query, the structure of the schemas should be provided to the LLM and this might not be a scalable solution.

General question answering is also a problem in RAG applications. For example, if a story is given to the RAG as the text data and the main theme of the story is asked by the user, RAG applications struggle to generate a good answer. This is because the text data is separated into chunks and separate chunks are retrieved and used for answering the questions. However, the chunks retrieved might not contain enough information to answer such a general question. One of the most popular architectures is the GraphRAG architecture, proposed by the paper GraphRAG [12]. GraphRAG proposes to create a KG based on the texts provided, create summaries of the entities by using the relations connected to entities, and create more abstract summaries by partitioning the KG into sub-graphs by using Leiden's Algorithm and create summaries based on the sub-graphs. For generating an answer to a general question, LLM is supported with these summaries(relevant ones according to the query). Although the fact that this is one of the best solutions to the general question answering problem, it is not suitable for KGs that grow because it is hard to integrate new data; Leiden's Algorithm must be executed again, new summaries should be constructed and so on. However, integration of new data is crucial for this project. Another alternative was proposed in the LightRAG paper [13], it again creates a KG but does not partition the graph or create summaries. It generates high level key-value pairs for the relations and chunks. For general question answering, it creates high level keywords(keys) for chunks and relations, and generates values for these keys. An example of a key-value pair might be "theme" and the explanation of the theme. When a user enters a query, some high level keywords are generated and a similarity search is done between the user query keys and data keys. The values of the most related keys are retrieved and given to the LLM for answering the question. Although this might not be as successful as GraphRAG for general question answering, it is more suitable for this project.

For the chatbot part, we plan to first identify the user's question type(multi-hop, comparative etc.), and act accordingly. We are designing a new architecture for answering the multi-hop and comparative questions, which we believe will be a suitable solution for our case. For general questions, we plan to use an architecture similar to the LightRAG.

As far as we know, there is no academic paper related to the notification feature. However, as mentioned in the High Level System Architecture part, we plan to use a search tree that is built with the users' notification queries. For building the search tree, the structure that was proposed in the RAPTOR [2] will be used. RAPTOR's approach is to recursively embed, cluster and summarize chunks of texts; and to construct a tree with different levels of summarization from the bottom up. By constructing such a tree, we plan to traverse the tree with an LLM from the root node to leaves, for each news item in order to find the user notification that is related with that news item. Lastly, prompt engineering techniques such as few-shot prompting or Chain of Thoughts [14] will be used to increase the performance of the LLMs.

Note: Academic Analysis is done based on the prioritized features since the remaining ones are not mandatory for this project.

4. Glossary

KG: Knowledge Graph
LLM: Large Language Model
RAG: Retrieval Augmented Generation

5. References

[1] L. Gao, X. Ma, J. Lin, and J. Callan, “Precise Zero-Shot Dense Retrieval without Relevance Labels,” *arXiv.org*, Dec. 20, 2022. https://arxiv.org/abs/2212.10496

[2] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, “RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval,” *arXiv.org*, Jan. 31, 2024. https://arxiv.org/abs/2401.18059

[3] “The complete AI powered stock research platform - FinChat.io,” *FinChat.io*. https://finchat.io/

[4] “AI TICKER CHAT - Intelligent Investing simplified.” https://aitickerchat.com/

[5] “Uptrends.ai | Track trending stocks.” https://www.uptrends.ai/

[6] A. Vaswani *et al.*, “Attention is all you need,” *arXiv.org*, Jun. 12, 2017. https://arxiv.org/abs/1706.03762

[7] N. F. Liu *et al.*, “Lost in the Middle: How language models use long contexts,” *arXiv.org*, Jul. 06, 2023. https://arxiv.org/abs/2307.03172

[8] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” *arXiv.org*, May 28, 2020. https://arxiv.org/abs/2005.14165

[9] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal, “Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions,” *arXiv.org*, Dec. 20, 2022. https://arxiv.org/abs/2212.10509

[10] B. J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, “HiPpoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models,” *arXiv.org*, May 23, 2024. https://arxiv.org/abs/2405.14831

[11] Z. Hong, Z. Yuan, H. Chen, Q. Zhang, F. Huang, and X. Huang, “Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM,” *arXiv.org*, Feb. 18, 2024. https://arxiv.org/abs/2402.11517

[12] D. Edge *et al.*, “From Local to Global: A Graph RAG Approach to Query-Focused Summarization,” *arXiv.org*, Apr. 24, 2024. https://arxiv.org/abs/2404.16130

[13] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, “LightRAG: Simple and Fast Retrieval-Augmented Generation,” *arXiv.org*, Oct. 08, 2024. https://arxiv.org/abs/2410.05779

[14] J. Wei *et al.*, “Chain-of-Thought prompting elicits reasoning in large language models,” *arXiv.org*, Jan. 28, 2022. https://arxiv.org/abs/2201.11903

[15] OpenAI, “Pricing,” *OpenAI*, 2024. https://openai.com/api/pricing/

[16] RunPod, “Pricing for GPU Instances, Storage, and Serverless,” *Runpod.io*, 2024. https://www.runpod.io/pricing (accessed Nov. 21, 2024).

[17] “Amazon API Gateway Fiyatlandırması | API Yönetimi | Amazon Web Services,” *Amazon Web Services, Inc.*, 2024. https://aws.amazon.com/tr/api-gateway/pricing/ (accessed Nov. 21, 2024).

[18] “Neo4j Pricing,” *Neo4j Graph Data Platform*. https://neo4j.com/pricing/

[19] MongoDB, “Pricing,” *MongoDB*. https://www.mongodb.com/pricing