TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Rene Väljaots 232758IABB

Steven Ernits 233169IABB

Publicly Accessible Platform for Company Financial Analysis Using Large Language Models

Bachelor's Thesis

Supervisor: Karl-Erik Karu
MSc

Tallinn 2026

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rene Väljaots 232758IABB

Steven Ernits 233169IABB

Suurtel keelemudelitel põhinev avalikult kasutatav finantsanalüüsi platvorm ettevõtete analüüsimiseks

Bakalaureusetöö

Juhendaja: Karl-Erik Karu
MSc

Tallinn 2026

Authors' declaration of originality

We hereby declare that we have compiled the thesis independently, and the same paper has not been previously presented for grading. All mentioned works, important standpoints, and data by other authors have been properly referenced.

Authors: Rene Väljaots, Steven Ernits

17.05.2026

Abstract

This thesis presents the further development of a financial analysis platform for companies listed on the Baltic Stock Exchange, transforming an existing internal prototype built by the authors into a publicly accessible application. The project aims to solve the problem that relevant financial information about Baltic companies is scattered across news portals, press releases and financial reports, making research time-consuming especially for retail and beginner investors.

The developed web application combines web scraping, automated data-processing pipelines and large language models to collect, process and summarise publicly available company information. During the work, an initial internal-use prototype was redesigned into a public platform by refactoring the backend to hexagonal architecture, automating the generation of summaries for news articles, financial reports and companies, and adding secure user authentication along with an improved user interface. The finished system provides users with company overviews generated by large language models, summarised news articles, financial report summaries, stock price charts and intuitive filtering options.

The result is a scalable and extensible platform that reduces the manual research required to analyse the companies listed on the Baltic Stock Exchange and presents financial information in a more accessible format. The thesis also evaluates the technical decisions made during development, including the architectural changes, automated pipelines and large language model selection. The completed platform is validated through user feedback.

The thesis is in English and contains 52 pages of text, 6 chapters, 19 figures, 6 tables.

Annotatsioon

Käesolev töö käsitleb autorite loodud asutusesiseseks kasutamiseks mõeldud finantsanalüüsi platvormi prototüübi edasiarendust avalikult kasutatavaks platvormiks Balti börsil noteeritud ettevõtete analüüsimiseks. Projekt keskendub probleemile, et Balti ettevõtteid puudutav asjakohane finantsteave on killustunud uudisteportaalide, börsiteadete ja finantsaruannete vahel, mistõttu on vajaliku informatsiooni leidmine ajamahukas, eriti jae- ja algajatele investoritele.

Arendatud veebirakendus ühendab veebikammimise, automatiseeritud andmetöötluse töövood ja suurkeelemudelid, et koguda, töödelda ja kokku võtta avalikult kättesaadavat teavet ettevõtete kohta. Töö käigus kujundati esialgne asutusesiseseks kasutamiseks mõeldud prototüüp ümber avalikuks platvormiks. Selleks viidi rakenduse tuumloogika üle heksagonaalsele arhitektuurile, automatiseeriti kokkuvõtete genereerimine uudiste, finantsaruannete ja ettevõtete jaoks ning lisati turvaline kasutajate autentimine koos täiustatud kasutajaliidesega. Valminud süsteem pakub kasutajatele suurkeelemudelite loodud ülevaateid ettevõtetest, lühikokkuvõtteid uudisartiklitest, finantsaruannete kokkuvõtteid, aktsiahindade graafikuid ja intuitiivseid filtreerimisvõimalusi.

Töö tulemusena valmis skaleeritav ja laiendatav platvorm, mis vähendab Balti börsil noteeritud ettevõtete analüüsimiseks vajalikku manuaalset infootsingut ning esitab finantsteavet kasutajale arusaadavamal ja kergemini hoomataval kujul. Lõputöös hinnatakse ka arenduse käigus tehtud tehnilisi otsuseid, sealhulgas arhitektuurilisi muutusi, automatiseeritud töövoogusid ja suurkeelemudelite valikut. Valminud platvormi valideeritakse kasutajate tagasiside kaudu.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 52 leheküljel, 6 peatükki, 19 joonist, 6 tabelit.

List of Abbreviations and Terms

API	Application Programming Interface
Alembic	Database migration tool
Backend	The server-side logic of an application
Branch	Parallel line of development
Certbot	An open-source software tool for automatically enabling HTTPS using Let's Encrypt certificates
Commit	A saved change in version control
Container	Isolated environment for running applications
Content-Security-Policy	An HTTP response header that allows to restrict the resources browsers are allowed to load
CRUD	Create, Read, Update, Delete operations
Docker	Tool for running software in isolated containers
Domain	Website address or application-specific business area
Entity	Object in the database
FastAPI	A Python framework for building REST APIs
Framework	A structured foundation for software development
Frontend	The user-facing part of an application
GDPR	General Data Protection Regulation, EU regulation for personal data protection
GitHub	A platform for storing and collaborating on code
HTML	HyperText Markup Language, standard language for structuring web pages
ISIN	International Securities Identification Number

JWT	JSON Web Token, a compact token for authentication
Link table	Many-to-many relationship table
LLM	Large Language Model
Migration	Modification to the database schema
Nginx	An intermediary server that handles incoming web traffic and routes it to internal backend containers
Nuxt	A framework for building Vue-based applications
OAuth 2.0	Authorisation framework
ORM	Object-Relational Mapper
Prompt	Instruction text for an LLM
RBAC	Role-Based Access Control, user-permission model
Redis	Used to send messages between services
REST API	Style of API that uses standard HTTP methods
SQLModel	A Python library for defining database models
SSL/TLS certificate	A digital object that allows systems to verify the identity and establish an encrypted network connection to another system using SSL/TLS protocol
Strict-Transport-Security	A web security policy mechanism that forces browsers to interact with the website only via secure HTTPS connections
Systemd	A system and service manager for Linux operating systems
URL	Uniform Resource Locator, web address
VPS	Virtual private server
X-Frame-Options	An HTTP response header used to prevent clickjacking by restricting whether a browser can render the page in a frame
XML	A machine-readable markup language

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 General background and project overview

The project was initially developed as a team project as part of the Information Systems Development Team Project: Procurement course at Tallinn University of Technology. The original idea was to develop an application that would automatically gather information about Baltic and Swedish companies and use LLMs (*Large Language Models*) to create comprehensive summaries that simplify the analysis of a company for a bank employee. During the project's research stage, it became apparent that the solution would serve a much greater purpose if it were to be publicly usable, since the market lacked such an analysis tool for retail investors. Unlike bank employees, the general public often finds financial data difficult to understand or too time consuming to read and analyse, which can lead to lower confidence in investing. The goal of this platform is to make investing more accessible by bringing useful information together in one place.

Many non-investors around the world share the same reasons for staying out of the stock market. For beginners, investing can feel intimidating and risky. Studies show that this hesitation usually comes down to a general lack of knowledge about investing. People often do not understand how the stock market works, feel overwhelmed by complex financial information or simply do not have the time to do necessary research [1] [2] [3] [4]. A survey conducted by SEB Bank in 2021 revealed that only 18% of Estonians, 5% of Latvians and 8% of Lithuanians had invested in stocks [5]. An article about the same survey published by Delfi Ärileht mentions that the main reason why Estonians had not invested was the lack of savings (56%), while roughly 20% of the participants said that they did not know where to start or lacked sufficient knowledge, and another 20% said that lack of time was the reason why they had not invested [6]. These statistics show that for about 40% of non-investors, the main barrier is a lack of time and knowledge. The Baltic Stock Exchange is relatively small compared to other European stock exchanges, consisting only of about 70 companies. Due to its small

scale, major global investment and news platforms do not cover much of the information about the Baltic companies, making this information difficult to find.

1.2 Problem and project objectives

Anyone looking to invest their money should be aware of the risks involved and take measures to lower their chances of losing money. This requires research, which can take a long time if relevant information is scattered across multiple sources and large documents. There is currently no publicly available tool that would provide investors a quick and easy way to perform financial analysis for companies listed on the Baltic Stock Exchange. This is a problem especially for retail and beginner investors, who use their own money to invest and might not have enough time or resources to search for and consume news, press releases or reports that might provide them with adequate information before making investment decisions. The project aims to see if LLMs coupled with web-scrapers and a scalable software architecture can eliminate the need to manually search for individual pieces of information about a given company. The further development of the project has three primary objectives:

- Refactoring the existing solution to hexagonal architecture for increased scalability and testability
- Automating data-processing and pipelines completely by removing user-initiated jobs to minimise wait times and cost
- Transforming the application from an internal tool to a platform that is publicly usable

These changes and additions ensure the application is suitable for users of varying experience levels in investing and allow for more straightforward scalability options.

1.3 Structure of the thesis

The thesis is structured as follows. Chapter 2 provides theoretical and practical background for the project by discussing financial data accessibility in the Baltic region, relevant software architecture patterns, web scraping and the use of large language models in financial text analysis. Chapter 3 describes the methodology of the

project, including project object, selected tools and technologies, development processes and user testing approach. Chapter 4 presents the developed solution in detail, covering the system architecture, hexagonal architecture implementation, data model, automated pipelines, LLM operations, user management, API, user interface and production deployment. Chapter 5 analyses the completed project, including an assessment of the implementation outcomes, the results of user validation, the comparison of different LLMs used for the application's main tasks and possible future developments. Chapter 6 summarises the main outcomes of the thesis, evaluates whether the project objectives were achieved and outlines possible future improvements.

2 Background

This chapter covers the technical background of the project and establishes the real-world need for the solution. Existing solutions do not provide comprehensive coverage for the Baltic region and finding relevant financial information is often more time-consuming than it should be. The existing solution developed by the authors, which was meant for internal use, is reworked to use architectural patterns which would keep the project scalable for future additions, like adding new markets or data sources. What is web scraping and how it is used to gather information for the LLMs is explained in detail, as this is the primary method of data collection and processing in this application. The analysis of these topics will provide justification for the further development of the existing project.

2.1 Financial data accessibility and the Baltic market

OECD defines financial literacy as "a combination of financial awareness, knowledge, skills, attitudes and behaviours necessary to make sound financial decisions and ultimately achieve individual financial well-being" [7, p. 6]. Financial literacy is recognised as one of the primary barriers to capital market investing. A global survey conducted by the World Economic Forum found that 40% of non-investors chose not to invest because they did not know how or found the process too confusing [8, p. 34].

The OECD/INFE 2023 International Survey of Adult Financial Literacy, which measured financial literacy across 39 countries including Estonia, Latvia and Lithuania, further confirms the low rate of financial literacy, showing that the percentage of adults who scored at least 70 points out of 100 on a financial literacy test is 48% for Estonia, 33% for Latvia and 23% for Lithuania. The OECD average score was 39%. [7, p. 16]

In highly active US and European markets, retail investors can easily access both company data (such as financial reports, news, and press releases) and professional made summaries. However, for the Nasdaq Baltic Exchange, the level of information

accessibility is not the same. The Nasdaq Baltic Exchange lists only 31 companies on their main list, which consists of the most respected companies on the Nasdaq Tallinn, Nasdaq Riga and Nasdaq Vilnius stock exchanges [9]. Including the secondary and bond lists, the number of companies rises to approximately 70.

While premium platforms like Bloomberg and Refinitiv offer comprehensive data and analysis for the Baltics, their high costs make them inaccessible to the average investor. Free platforms like Yahoo Finance do present press releases for Baltic companies, but do not provide regional news, which may also provide valuable information. Local financial institutions, for example LHV, offer a paid service to get market and company summaries. This means that for the Baltic market the information is highly scattered. For the retail investors, gathering scattered news and data from multiple sources and analysing it manually is a time-consuming process.

The combination of low financial literacy rates, low stock market participation and scattered coverage by existing platforms creates an information gap in the Baltic market. A tool that consolidates publicly available financial data for Baltic-listed companies and presents it in a summarised, accessible format could help reduce the barrier that prevents retail investors from participating in the local stock market.

2.2 Software architecture patterns for web applications

Deciding on an architectural pattern is an important step during the process of software development, it sets clear boundaries and criteria for how the project should be structured, which results in efficient development and a robust, scalable, and testable result. There are many patterns to choose from when designing a web application, most common choices are the hexagonal, clean, and onion architectures. For this project the hexagonal architecture was chosen for its simplicity and adaptability, as it can be adjusted to fit for any use case.

Hexagonal architecture, often also called the Ports and Adapters pattern, is a software architectural style introduced by Alistair Cockburn in 2005. The fundamental idea behind the pattern is that the core business logic of an application should be completely isolated from its external dependencies, such as databases, user interfaces or third-party services. To achieve this separation, the architecture defines ports and adapters.

A port is an interface that describes how the application expects to interact with the outside world, an adapter is the implementation of a port for a specific technology. For example, a port might define how the application retrieves company data and one adapter could implement this by querying a database, another adapter might fetch the same data from an external REST API (*Representational State Transfer Application Programming Interface*). The pattern is called hexagonal architecture due to the hexagon shape used in diagrams to represent application core, which Cockburn chose specifically to move away from the traditional top-to-bottom layered drawings that were the default way to visualise software architecture. The number of sides has no special significance, the shape simply provides enough space to attach as many ports and adapters as needed. [10]

The advantage of hexagonal architecture is that it provides strict separation between business logic and technical infrastructure, which improves testability and maintainability, making replacing or adding new external technologies seamless and low risk. As the core logic only communicates through ports, it is possible to replace external components, such as databases, APIs or third-party services by simply writing a new adapter without modifying the business rules. This also means that real infrastructure can be replaced with in-memory or mock adapters during testing, which makes the core logic testable in complete isolation. Netflix adopted hexagonal architecture in their Studio ecosystem for these reasons, noting that the pattern allowed them to swap data sources without impacting business logic and that the tests could verify business rules without relying on specific protocols or technologies. [11]

The hexagonal architecture was not the only pattern to suggest isolating business logic from the outside world. In 2008, Jeffrey Palermo introduced the Onion Architecture, which builds on the same rule of directing all dependencies toward the centre but adds explicit internal layers to organise the application core, separating domain models from application services [12]. In 2012, Robert C. Martin stated the Clean Architecture pattern, which he described as an attempt to integrate the ideas behind hexagonal architecture, onion architecture and several other patterns into a single approach [13]. Clean Architecture further refines the business logic by distinguishing between enterprise-wide business rules called entities and application specific logic called use cases, which is a level of detail neither of the other patterns define [13]. Despite the

differences in structure and naming, all three patterns share the same objective to separate the concerns of business logic and external services. Neither onion nor clean Architecture reinvents the idea of hexagonal architecture, rather they build upon it by adding more granular organisation to the application core.

2.3 Web scraping as a data collection method

Web scraping is a technique of automatically gathering data from a website using software. It allows for extracting data from text such as HTML (*HyperText Markup Language*) code and is especially useful for situations where the data is not provided in a machine-readable format like JSON (*JavaScript Object Notation*) or XML (*Extensible Markup Language*) through an API [14]. In the context of this project, the Baltic news portals do not provide publicly available APIs to fulfil the project's goals, this makes web scraping the only practical method for collecting necessary data.

Web scraping is a powerful tool for data collection, but there are ethical and legal considerations that must be taken into account. Scrapers should respect the target website's terms of service and implement rate limiting to avoid overloading the server and disrupting access for regular users [15]. In the European Union, GDPR (*General Data Protection Regulation*) imposes additional constraints for scraping regarding personal data, which requires a lawful basis for processing [16]. The scrapers developed for this project collect only publicly available financial data and news texts, which do not include personal data.

2.4 Large language models in financial text analysis

A large language model is a type of artificial intelligence that can process and produce natural text. Modern LLMs are built on the transformer architecture introduced by A. Vaswani et al. in 2017, which replaced earlier sequential neural network approaches with a self-attention mechanism that allows the model to process all words in a sequence simultaneously and capture relationships between them regardless of distance [17]. This breakthrough enabled language models to be scaled to billions of parameters and trained on vast amounts of text data, during which they learn patterns of grammar, factual knowledge and reasoning [18]. The models function by predicting

the next most probable continuation of a given input [18]. This is important to note, because it means that LLMs can produce fluent and believable text that may not be completely factual.

The application of LLMs in the financial world has been an active area of research in recent years. A survey by Y. Nie et al. covering LLM applications in finance found that these models have shown strong capabilities in sentiment analysis, text classification, information extraction and financial reasoning, which makes them valuable in a sector where understanding market sentiment is crucial for making decisions [19, p. 2]. Research conducted by A. Kim, M. Muhn and V. Nikolaev in 2024 revealed that GPT-4.0 Turbo could perform financial statement analysis similarly to a professional human analyst and found that the model outperformed the median financial analyst in predicting the direction of future earnings changes [20]. The research also found that GPT's predictions were on a par with the custom trained machine learning models [20]. These results suggest that LLMs can extract valuable insights from financial data with correct prompting.

The use of LLMs for any purpose comes with known limitations. The most significant risk, especially in financial contexts, is hallucination. Hallucination is a model generating content that sounds plausible but is factually incorrect, inconsistent or completely fabricated [21]. Inaccurate outputs for financial questions could lead to poor investment decisions, which makes the reliability of LLMs a real concern. Research has shown that structured prompt engineering such as chain-of-thought prompting and role assignment can reduce the occurrence of hallucinations by guiding the model to reason more carefully [22]. Retrieval-augmented approaches, where the model is provided with verified source documents rather than relying only on its own training data, have also proven effective in increasing factual accuracy in outputs [22].

3 Methodology

The methodology chapter covers the approaches, tools and processes applied in the further development of the initial project started in the Information Systems Development Team Project: Procurement course. It introduces the technical and structural changes required to transform the existing internal use solution into a publicly accessible web application.

3.1 Detailed description of the project object

The initial version of the application was developed as part of a university team project for AS SEB Pank. It was originally designed strictly for internal use by the bank's employees to assist with financial market analysis and strategic decision making. This early version allowed users to search for listed Baltic or Swedish companies and instantly view or generate LLM-based comprehensive company analyses. It also enabled employees to browse LLM-generated summaries of news articles and press releases retrieved by the web scrapers, eliminating the need for manual research across multiple sources.

Future additions were not a primary consideration when the team designed the initial project. Although the codebase was organised into logical layers, some of the underlying business logic became tightly coupled, limiting the overall extensibility of the system. Additionally, user accounts were maintained manually by an administrator, and users had unrestricted access to generate LLM summaries for any company listed on the Baltic exchange, as long as the set LLM token limits were not reached. While this previous architecture and unrestricted access functioned correctly for a small and trusted internal user base, transitioning the platform to a publicly accessible application required a structural redesign.

During the further development multiple structural changes were made. This mainly consisted of adopting the hexagonal architecture in the backend to separate the core of the application from outside sources and adding automated data pipelines to ensure

that the database was always populated with new data, while minimising user wait times and keeping LLM usage under control. User management was also changed to allow users to create their own accounts. The frontend was refactored to be more secure and structured. This turned the project into a publicly usable web application that can be easily extended in the future as the LLM technology is constantly changing and news sources might alternate.

At a high level, the system's operational flow is divided into five conceptual layers:

- The data collection layer consists of multiple web scrapers, each designed for a specific news source. The scrapers are responsible for gathering news articles, press releases, and financial reports as well as returning the results in a structured manner.
- The data persistence layer ensures that the collected data is stored correctly with the necessary relations.
- The data processing layer is responsible for managing the automated data pipelines and LLM operations to provide latest data for the database.
- The data access layer communicates to the database to handle user requests and enforces system security by implementing RBAC (*Role-Based Access Control*) and token-based authentication using JWT (*JSON Web Token*). This ensures that access to data is regulated based on defined user privileges.
- The user interaction layer provides access to the application's functionalities through a web interface. The two most important views in this interface are the company summary view and the news view. The former allows the user to query and view LLM-generated summaries of companies. The latter is for viewing LLM-summarised news articles and press releases stored in the database.

The finished web application provides users with comprehensive summaries of the Baltic companies, relevant news articles and ease-of-use features that would attract beginner investors to start investing and help experienced investors make more informed decisions, without committing to manual, time-consuming searches.

3.2 Description of tools and technologies

In this subchapter, an overview is provided for the tools and technologies used in the development process. Although the chosen technology stack is large, the choices of technologies were guided by the objective of creating a reliable and scalable solution, while taking into account the existing competencies of the team members.

Web scraping is the main method of data collection in this application. Both the scraping and the backend logic were implemented in the Python programming language. The primary library used for scraping is BeautifulSoup, which enables access to and parsing of a website's HTML structure.

The server-side logic was developed using the Python-based framework FastAPI. FastAPI was chosen because it is lightweight and has a less steep learning curve compared to its alternatives like Django, which is another popular Python web development framework. Secondly, FastAPI offers excellent documentation and numerous code examples. Sebastián Ramírez, the author of FastAPI provides comprehensive public examples of how to structure a REST API for a full-stack project while adhering to the best design practices. Since web scraping was implemented in Python, keeping the rest of the backend in the same language avoided unnecessary complexity. Data validation and database interactions were handled using SQLModel, which combines SQLAlchemy for database operations and Pydantic for data validation. For security, role-based access control was implemented, dividing users into three categories: user, admin and rootadmin. User authentication was managed using JWT and Google OAuth 2.0. Regular users authenticate using Google OAuth 2.0, admins and rootadmins authenticate using email and password. The passwords of administrative users are hashed using the Argon2 algorithm.

PostgreSQL was chosen for storing and maintaining data. This choice was based on the proven reliability of the technology and the team's prior experience with it. The use of NoSQL databases, such as MongoDB, were also considered, as the project's architecture was not expected to become highly complex in terms of data relationships. PostgreSQL was ultimately chosen because of the superior reliability of relational databases and the team's previous experiences with them.

Scheduled scraping was implemented using Celery as the task queue system, Celery Beat as a scheduler and Redis as the message broker. Celery provides tools for constructing scheduled tasks, which were used in the automated data pipelines to scrape data about companies, create news summaries, extract financial reports and generate company summaries. For LLM operations, the OpenAI API was used.

Automatic testing was used to ensure the quality and reliability of the software as it was being developed. Backend testing was done with the Pytest framework, which was used to write both unit and integration tests. Codecov was used to systematically monitor and evaluate code quality by examining untested parts of the codebase.

The frontend user interface was built on the Nuxt 4 framework, which is based on the Vue.js JavaScript framework. Nuxt was chosen because of the authors' prior experience and its developer-friendly features. State management was implemented using Pinia, which allows for efficient handling of data across different views. Additionally, the NuxtUI component library was used to create many of the frontend's components.

The application was containerised using Docker to ensure consistency across development and production environments. Docker Compose was used to orchestrate the multi-container architecture.

GitHub was used for version control during the entire development process. This ensured smooth teamwork by utilising branches and maintaining quality control with pull requests. GitHub Actions was implemented to run the automated tests which added another layer of validation to prevent non-functional code from reaching the main branch. Specific issue templates were created to ensure that all tasks and bug reports followed a similar structure. GitHub Actions was also used to automatically label issues based on their type and priority.

3.3 Description of the work process

The platform was developed by a team of two consisting of Rene Väljaots and Steven Ernits. The initial prototype was developed from September to December 2025, while further development took place from January and throughout the process of writing

the thesis. Specific roles were not assigned, instead, the members naturally took responsibility for particular aspects of development. Rene focused on implementing the technical architecture and infrastructure of the system. He developed the core backend code, transformed it to match the hexagonal architecture, managed the database schemas and design, containerised the application using Docker and handled the deployment to the VPS (*Virtual Private Server*). Steven also wrote a large part of the backend code, implementing new features and refactoring existing code. He mainly focused on the technical analysis of the system and researching relevant topics matching the project's theory and background. Additionally, Steven worked on the application's frontend, creating the user interfaces and logic to connect with the backend.

The team used an Agile inspired development methodology, which allowed for quick and efficient development of the project. Work progress was managed through GitHub Issues, which were used to document tasks, track progress and completion, and assign responsibilities. Issues were created by either member who identified the need for a new feature, refactor or a fix. Issues were typically self-assigned based on familiarity, expertise and interests. Larger tasks were often divided into sub-issues to simplify implementation. Pull requests served as the primary mechanism of code-validation, members reviewed each other's code before merging it to the main branch. This allowed for both members to stay up to date on changes and also spot any possibilities for corrections within the code.

The team followed strict version control standards by using the Conventional Commits specification for all commit messages and pull request titles. A structured branching strategy was implemented using the Conventional Branch specification, which requires branches to be prefixed according to their purpose. Shared documents and materials were stored in a common Google Drive folder, which included project analyses, diagrams, demonstration materials, and technical notes, such as commands for formatting code, executing Docker components, and running tests.

Communication within the team took place through frequent texts and voice calls, as well as real-life meetings. These meetings were held to discuss the next steps of development and reflect on what had been done beforehand. No larger feature or design change was implemented before prior discussion between the members.

3.4 User testing and validation of the solution

Testing with potential users of varying backgrounds was done after finishing the project's development. This was done to get feedback on the solution, validate its use case, and collect thoughts for future developments. Qualitative feedback was prioritised over quantitative for this project to allow for a deeper understanding of how users interact with the application and what they think of it. The test group was kept intentionally small, because this type of testing takes considerable time. Test users were deliberately chosen to have different experiences when it comes to investing, some had never invested in their life, and some had been investing for many years.

Test users were first asked to research a Baltic company of their choice, selected from a premade list. The list was put together by the authors since Baltic companies vary greatly in media coverage, with some having only a handful of news articles per quarter while others publish updates almost weekly. The list contained companies with a consistently high volume of financial news. The sources available for research were also predetermined based on each company's country of origin and to match the sources used in the application itself. Test users were given the objective to learn as much as possible about the company and its financial situation in limited time. The goal was not to achieve investment decisions, but rather to find a direction in which to perform further research or things to keep in mind about the company. After the manual research phase was complete, the users were asked to rate their position on a scale of 1 to 10 in the following aspects:

- Knowledge of the company after research
- Accessibility and understandability of the information found
- Information relevancy
- Whether the time given felt sufficient
- Motivation to research the company further
- Ability to decide what to analyse next
- Confidence in making a further investment decision

After that the users were given the same amount of time to achieve the same goal using the developed application and no outside sources. Once complete, they were tasked to assess the same aspects again. This allowed the authors to see if the developed

application was able to provide more value for them compared to the alternatives currently existing in the Baltic market. At the end, the users were asked some open questions about using the application, like what was difficult or confusing, what functionalities could be added or improved, and how they feel about the user interface.

4 Results

This chapter provides an in-depth overview of the system's finished features, architectural patterns, data models, internal processes, LLM integration, authentication, and user interface.

4.1 System architecture

The system was built as a containerised application orchestrated by Docker Compose. Separating concerns into independently deployable services simplified development, deployment and horizontal scaling of individual components. The production environment is explained in more detail in section 4.8.

The architecture implements the five-layer model described in section 3.1. The data collection layer consists of the scraper adapters. The data persistence layer is the PostgreSQL database. The data processing layer includes the Celery task queue and pipeline classes. The data access layer is the FastAPI REST API. The user interaction layer is the Nuxt 4 frontend.

At the centre of this architecture sits the FastAPI backend, which exposes the REST API consumed by the frontend. The backend communicates with a PostgreSQL database used for persistent storage and with Redis for task queue management. The Nuxt 4 frontend communicates exclusively with the backend via the REST API and has no direct access to the database or the message broker. Figure 1 presents the high-level system diagram.

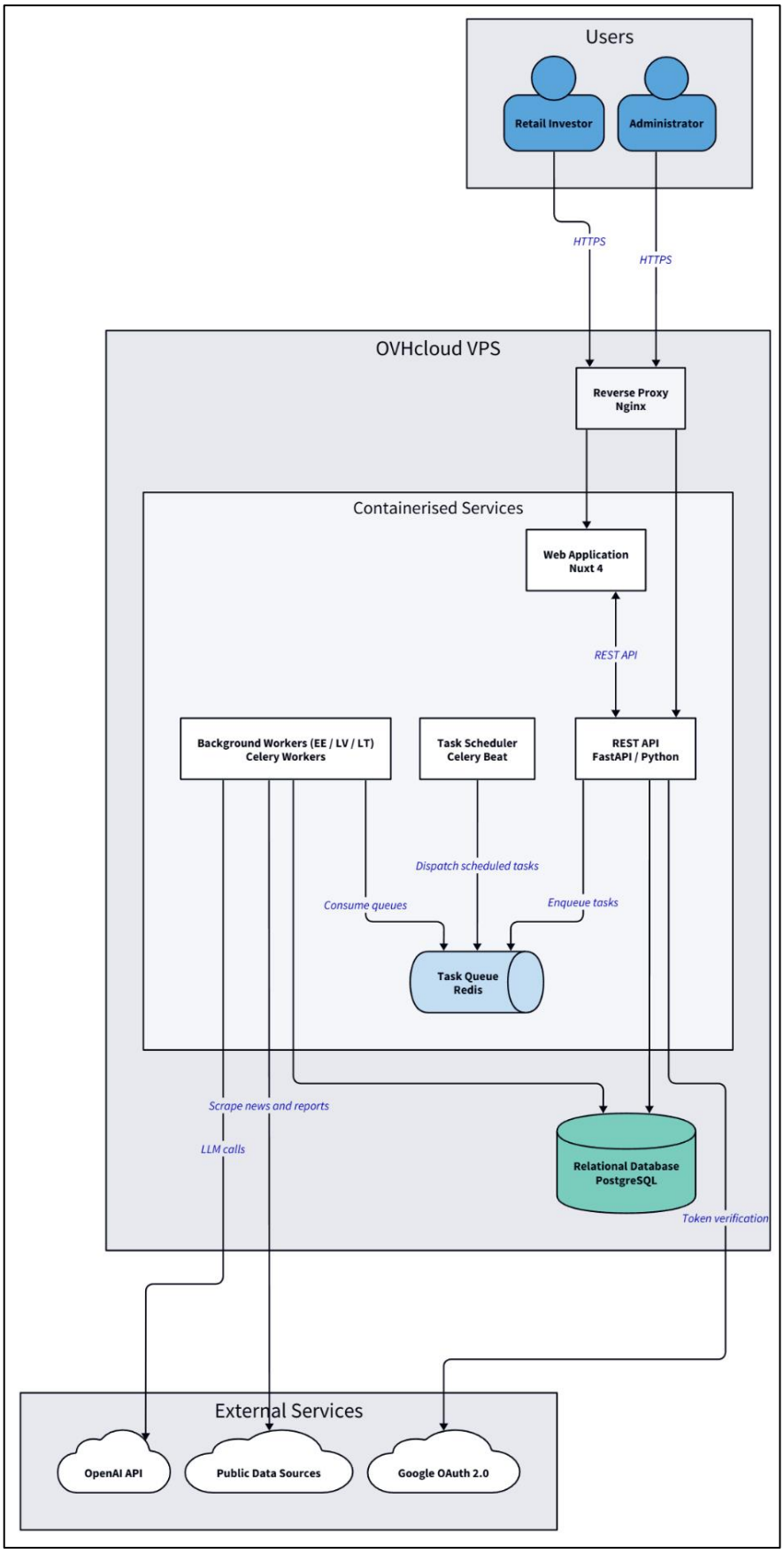


Figure 1. High-level system diagram

Background processing is handled by three Celery worker containers, one for each Baltic country. The workers consume tasks from country specific queues. A separate Celery Beat container is responsible for scheduling periodic tasks, such as data collection, article summarisation, financial report extraction, and company summary generation. Isolating the workers by country meant that a slowdown in one country's data pipeline does not affect the others. Figure 2 presents the high-level background processing diagram.

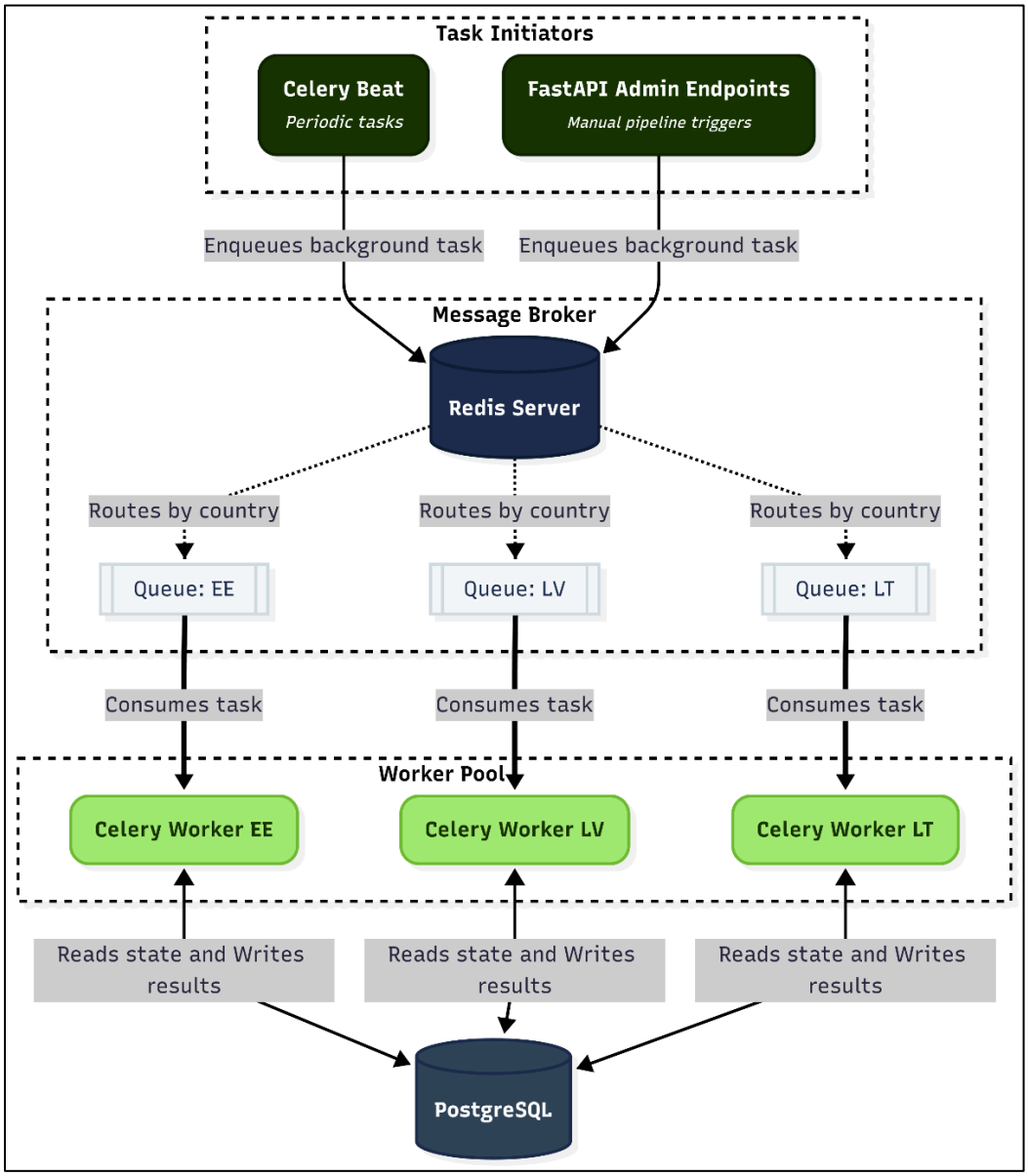


Figure 2. Background processing architecture diagram

4.2 Hexagonal architecture implementation

The initial version of the application did not follow explicit architectural patterns beyond basic layer separation. Because of that, the business logic became more coupled than it should have been. Components depended on concrete implementations rather than abstractions, which made it hard to change components in the codebase. Changing the logical steps of the article summarisation process required, for example, modifying code across multiple unrelated files. When the authors developed the initial version of the application, extensibility was not a requirement nor something likely to happen, which is why the time of development was spent elsewhere. Transitioning to a publicly accessible platform with automated pipelines and the expectation of future growth and developments made this tightly coupled structure unsuitable.

Refactoring to hexagonal architecture changed this dependency structure. Communication with external systems such as the LLM provider, news scrapers and financial report sources was made abstract using interfaces and the domain core was made independent of any specific external system. Figure 3 illustrates the resulting architecture, showing how the primary adapters (FastAPI route handlers and Celery task functions) send requests to the domain's services and pipelines, which in turn interact with secondary adapters via defined port interfaces.

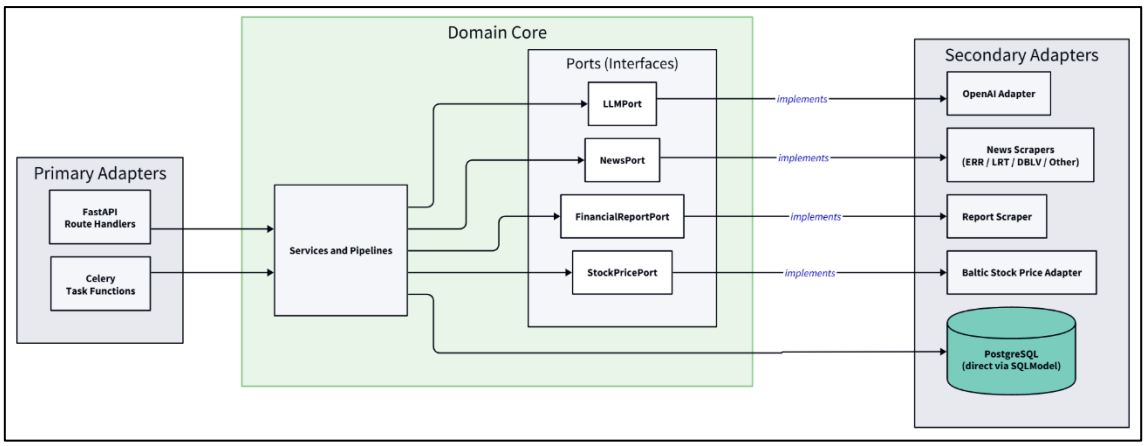


Figure 3. Hexagonal architecture diagram

4.2.1 Domain layer and business rules

The domain layer contains the business logic of the application. It contains the domain models, service classes, ports and pipelines. This layer is only aware of its own models, and the port interfaces it depends on.

Domain models are Python classes that define the data structures used throughout the application. Each model is implemented using SQLModel, a library that allows a single class definition to act as both a data validation schema and a database table mapping simultaneously. This means that the same article or company class is used to validate incoming data, to query the database and to construct API responses, without requiring separate schema and model classes to be kept in sync. This layer defines models for all core entities: companies, articles, article summaries, financial report summaries, company summaries, users and watchlists.

A port is a Python protocol class that defines what the domain expects from an external system without specifying how that system works. A port describes the method signatures (names, parameters and return types) that any implementing class must provide. Any class that provides the required methods satisfies the contract automatically. Port definitions are described in detail in section 4.2.2.

Services contain the core logic of the application. Each service is responsible for a single domain concern, such as company service or article summary service. Services enforce business rules, validate data and process information. For example, the article summary service contains methods for generating, validating, saving and querying article summaries. Because services are completely decoupled from FastAPI and Celery, the exact same service functions can be called by the REST API or the scheduled background jobs.

Pipelines are orchestration classes that use multiple services in a fixed sequence to complete a multi-step workflow. The application contains four pipelines: data collection pipeline, article summary pipeline, financial report pipeline and summary pipeline. Each pipeline receives its service dependencies through dependency injection and is designed to run as a single method taking a company as a parameter. The method completes workflow steps in order and handles errors gracefully. For instance, when one article fails to summarise, the entire batch of articles is not aborted. All errors are

logged for observability. Because pipelines depend only on injected services and ports, new steps can be added to existing workflows by inserting a new service call into the pipeline without breaking the surrounding code. Pipelines are also called identically from both FastAPI route handlers and Celery worker tasks.

4.2.2 Port definitions and adapter implementations

The application defines four port interfaces: LLM port, news port, financial report port and stock price port. LLM port defines five methods for communicating with the large language model. Each method accepts a system prompt string and relevant input data and returns a generic LLMResponse object. Figure 4 presents the LLMPort contract, showing the five method signatures and the generic LLMResponse object that all methods return.

```
class LLMResponse[T](BaseModel):
    data: T
    tokens: TokenUsage = Field(default_factory=TokenUsage)

class LLMPort(Protocol):
    def generate_company_summary(self, system_prompt: str,
company_data: str) -> LLMResponse: ...
    def generate_article_summary(self, system_prompt: str,
article_text: str) -> LLMResponse: ...
    def validate_article_summary(self, system_prompt: str,
validation_input: str) -> LLMResponse: ...
    def generate_financial_report_summary(self, system_prompt: str,
report_url: str) -> LLMResponse: ...
    def validate_financial_report_summary(self, system_prompt: str,
validation_input: str) -> LLMResponse: ...
```

Figure 4. LLMPort contract

The news port defines two methods which are abstract to support collection of news from scraped resources and from an API. One method is for finding article URLs (*Uniform Resource Locators*), which accepts a company name and a result count limit as inputs and returns a list of article URLs. The other method is for scraping the article, which accepts a URL and returns an object containing the article's title, publication date and full body text.

The financial report port defines a method that accepts a company ISIN (*International Securities Identification Number*) identifier and returns a list of URLs of financial report PDFs. The stock price port defines a method that accepts an ISIN, a start date, and an end date, and returns a list of timestamped price entries.

The adapters are concrete implementations of these interfaces. The application contains three categories of adapters: scraper adapters that collect data from external web sources, an LLM adapter which communicates with the OpenAI API and a stock price adapter that retrieves historical market data.

Each news source has a separate implementation of the news port, that scrapes the data from the relevant source and extracts the required content. For articles and press releases, the article title, publication date, body text and source URL are extracted. Example sources of articles are ERR for Estonia, DbLv for Latvia and Lrt for Lithuania. All adapters return a list of article objects that satisfy the interface.

The Baltic Stock Exchange financial report adapter implements the financial report port. This adapter fetches direct PDF URLs for annual and quarterly financial reports for the company. These URLs are later consumed by the financial report pipeline, which passes them to the LLM as file inputs.

The Baltic stock price adapter implements the stock price port. The stock price service uses this adapter to populate the company stock price table with historical daily closing prices, which the frontend uses to render interactive company price charts.

The OpenAI adapter implements the LLM port. It is responsible for all communication with the OpenAI API. It has five methods for article summarisation, financial report analysis and extraction and company summary generation. Every method calls OpenAI's responses API with a Pydantic model as an argument, activating OpenAI's structured outputs mode and ensuring that each response can be deserialised into the expected domain type without requiring error handling for malformed JSON.

4.3 Data model and persistence

The application's data layer is built upon a PostgreSQL database. To manage the interaction between the Python backend and the database, the system uses SQLModel as the ORM (*Object-Relational Mapper*). SQLModel combines the capabilities of SQLAlchemy and Pydantic. In a standard Python stack, developers define two separate classes for the same entity: one for the database table (SQLAlchemy) and one for API

data validation rules (Pydantic). SQLModel eliminates this duplication by allowing a single class definition to serve both purposes.

Database migrations are handled by Alembic. As the application requirements change, the database structure must be updated without losing existing data. Alembic tracks the state of the database models in the code. When a model is modified (for example adding a token usage column), Alembic generates a version-based migration script. These scripts are executed during the deployment process to safely apply schema changes to the production PostgreSQL instance.

The database consists of 7 primary tables:

- **Company**: This table stores identification data of companies including the name, ISIN, ticker symbol, industry sector, and country of origin, which is defined by an enum class that lists three company codes: EE, LV, and LT. The Company table data is seeded to the database at the initial startup of the application from a CSV file.
- **Article**: This table stores raw article data scraped from external sources. Its fields represent the article's title, URL, source, body text and publication date. The source field is defined by an enum class that lists three sources: ERR, DBLV, and LRT. A unique constraint on the URL field ensures that every article is scraped and stored only once.
- **ArticleSummary**: This table stores the LLM-generated mini-summary of a specific article. It maintains a one-to-one relationship with its parent Article entity. It includes the article's English title, summarised body text, verification status, creation date, and token usage statistics. As the ArticleSummary table is the primary source of article information across the application, this table also includes fields such as the source article URL and publication date for negating the need to call the large Article entity for these small details.
- **FinancialReportSummary**: This table stores the LLM-generated summary of a company's financial report. It stores the LLM's structured output as a JSON string, alongside the report's URL, verification status, creation date, validation information and token statistics. This table has a many-to-one relationship with the Company entity.

- **Summary**: This table represents the LLM-generated company summary. It stores the summary's introductory paragraph in the text field and the rest of the data in the structured_data field, similarly to the FinancialReportSummary. The introductory paragraph is stored separately from the structured output because it is used in multiple places, some of which do not require the entire structured output to be fetched. This table has a many-to-one relationship with the Company entity and includes additional fields for the used financial report's URL and token usage statistics.
- **User**: This table stores the users of the application. It holds identification fields such as the user's name and email, which has a unique constraint to prevent duplicate users. The role field is defined by an enum class that lists three roles: user, admin and rootadmin.
- **CompanyStockPrice**: This table is for holding a company's historic stock price data. It has a many-to-one relationship with the Company entity and includes two fields for persisting the stock price and the price's close date. The CompanyStockPrice table with its fields and relation to Company is presented in Figure 5.

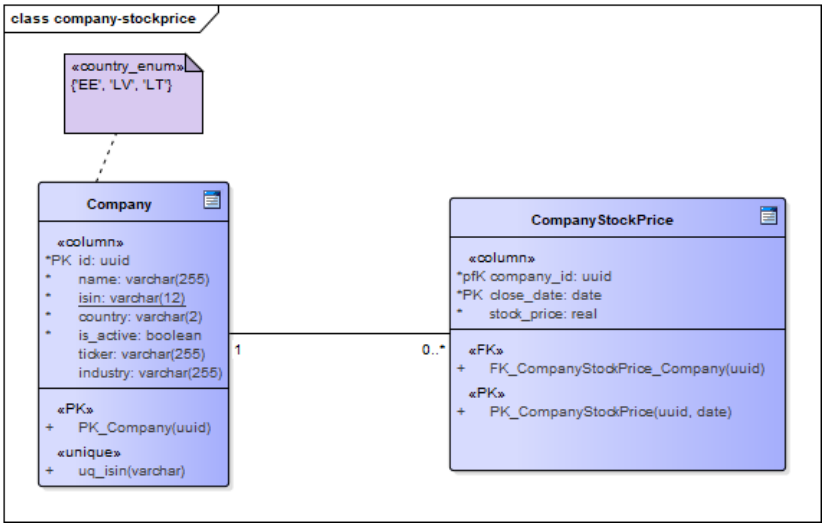


Figure 5. CompanyStockPrice entity relation diagram

The database also has 4 link tables, which handle additional relationships between the primary tables: ArticleCompanyLink, ArticleSummaryCompanyLink, SummaryArticleSummaryLink, and UserWatchlist. All link tables have fields only for referencing the tables they are related to, they have no unique properties of their own.

The ArticleCompanyLink table enables a many-to-many relationship between Company and Article. The ArticleSummaryCompanyLink table enables a many-to-many relationship between Company and ArticleSummary. The Company, Article, and ArticleSummary entity relationships are presented in Figure 6.

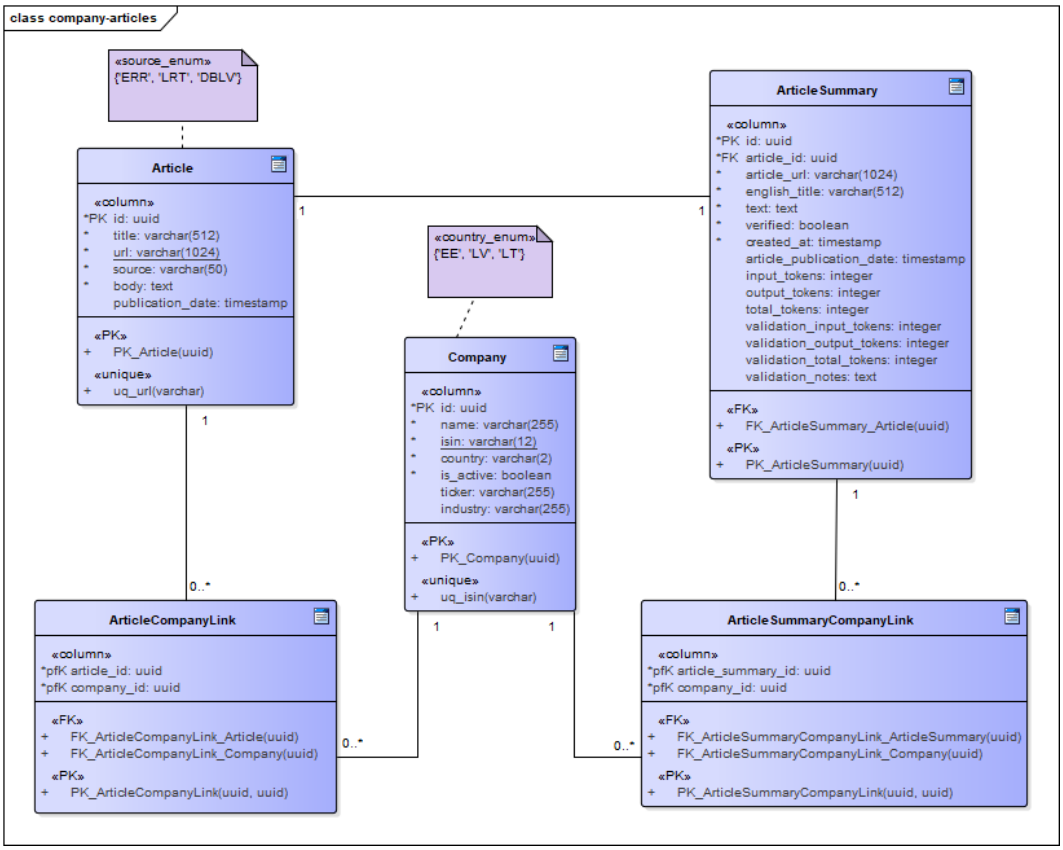


Figure 6. Company Article entity relation diagram

The SummaryArticleSummaryLink table enables a many-to-many relationship between Summary and ArticleSummary. The Company, Summary, FinancialReportSummary, and ArticleSummary entity relationships are presented in Figure 7.

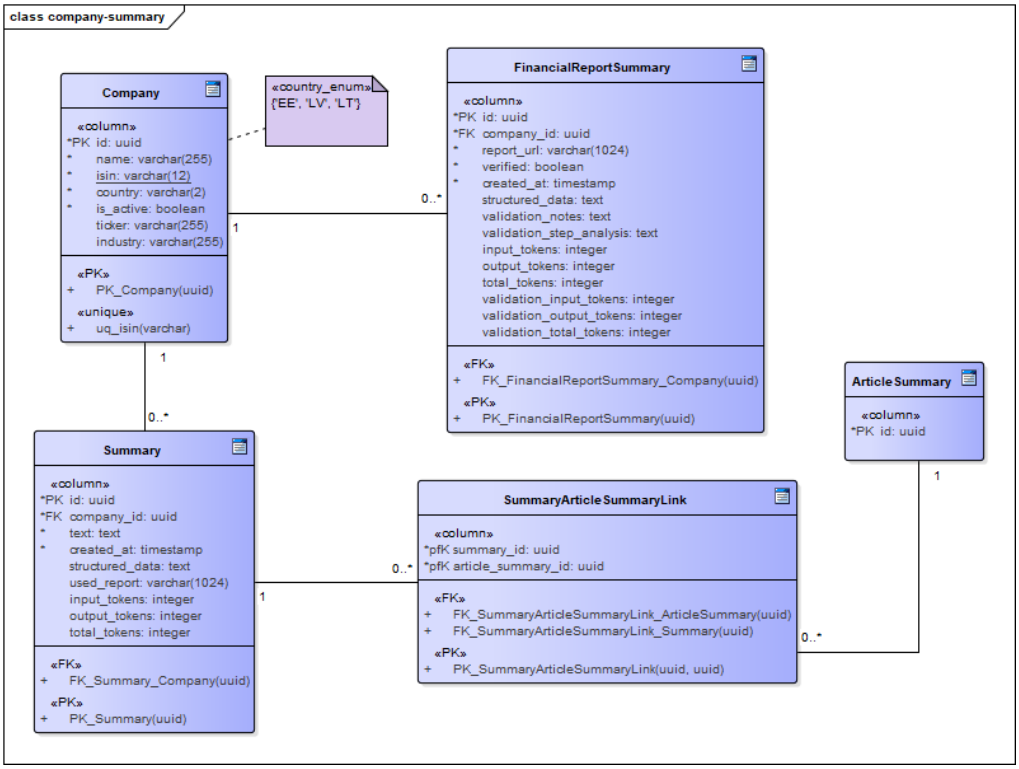


Figure 7. Company Summary entity relation diagram

The UserWatchlist table enables a many-to-many relationship between User and Company, implementing the favouriting feature available to the user. The User, Company, and UserWatchlist entity relationships are presented in Figure 8.

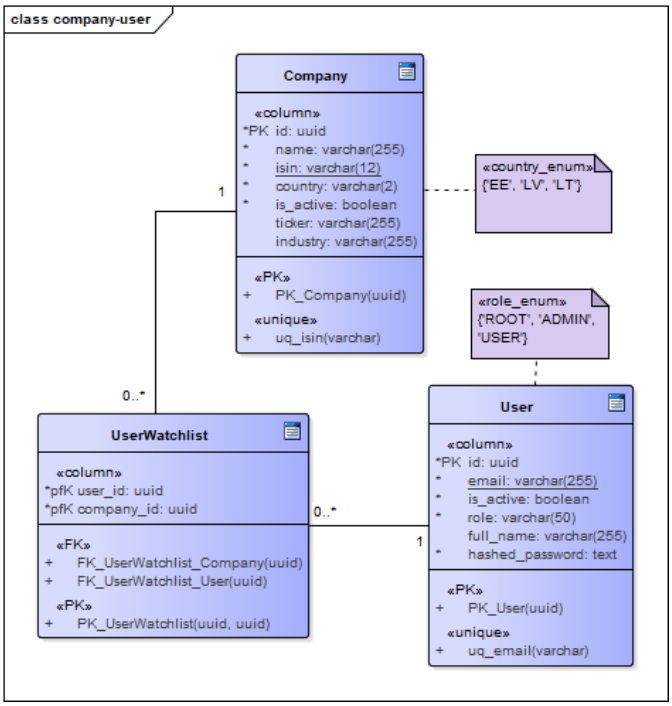


Figure 8. UserWatchlist entity relation diagram

4.4 Pipelines

The application's core data processing is organised into four pipelines. Each pipeline works on a per company basis and is triggered either automatically by the Celery Beat scheduler at fixed times or manually through protected admin API endpoints.

The data collection pipeline is responsible for gathering raw inputs from external sources. For each company, the pipeline selects the appropriate scraper adapters based on the country's registered country code. For each configured source, the pipeline calls the source's URL discovery function to retrieve candidate article URLs and then it partitions the results into URLs that are already present in the database and URLs that are new. For URLs that already exist, the pipeline checks whether the article has been linked to the current company and creates that association if it has not. This handles the case where a single news article references multiple companies and was already collected during an earlier run for a different company. New URLs are passed to the scraper, which fetches the full article text and metadata and persists the record in the database. The pipeline overall result can be either success, partial success or failure. If all sources completed without errors, the pipeline result is success. If errors occurred but at least some content was still produced or linked, the result is partial success. If URLs were found but every scraping attempt failed and nothing was linked, the result is failure. The pipeline is scheduled to run daily at 00:00 UTC. Figure 9 presents the sequence diagram of the data collection pipeline.

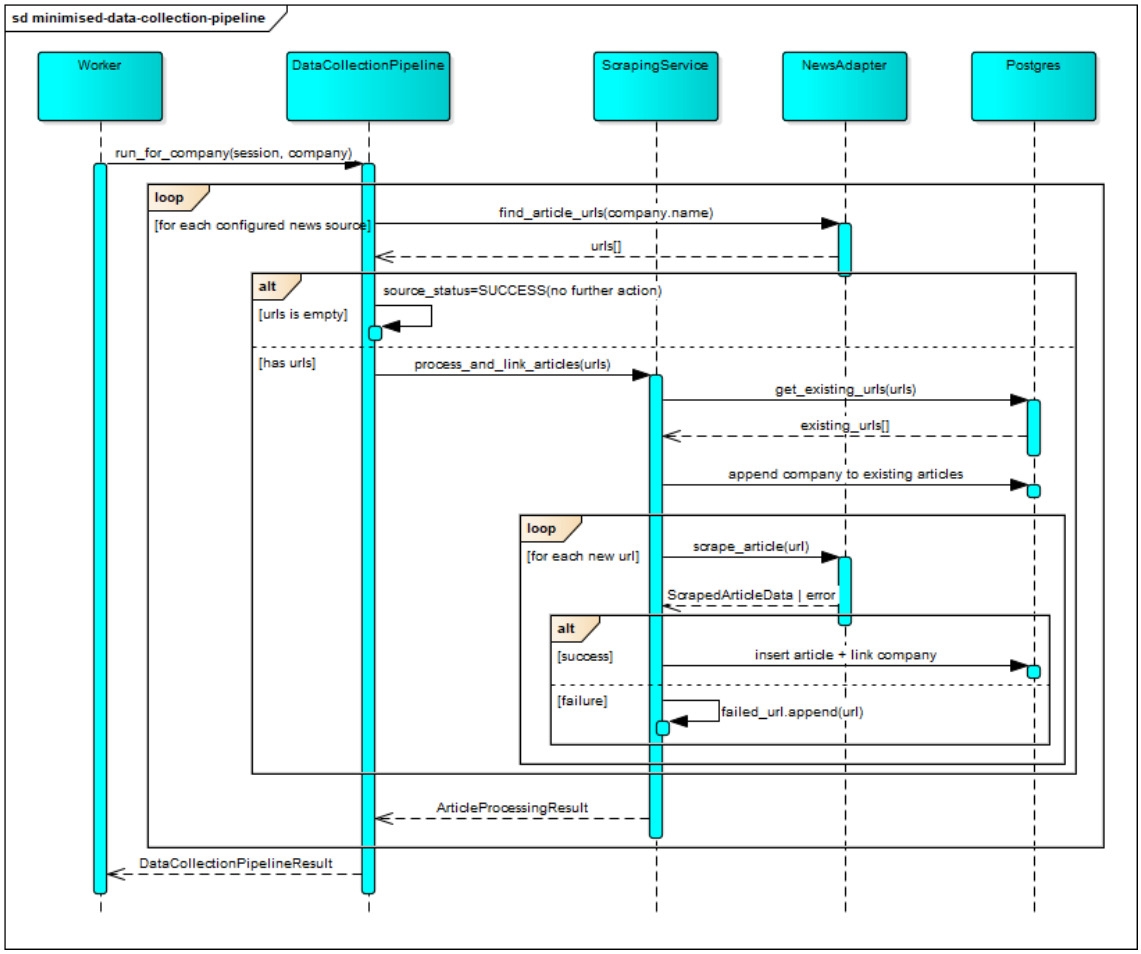


Figure 9. Data collection pipeline sequence diagram

The article summary pipeline processes articles that have been collected by the data collection pipeline but have not been summarised. For each unsummarised article, the pipeline calls the LLM adapter to produce an initial article summary containing a title in English and a short body, which is then sent to the LLM to be validated. The validation mainly checks if the mentioned numeric values are accurate and exist in the original article and if the tonality of the summary is neutral. The validation flag and validation notes are saved in the database alongside the article summary. If no unsummarised articles are found at the start of a run, the pipeline exits immediately with a skipped status. Otherwise, the overall result follows the same pattern as the data collection pipeline: success if all articles were processed without error, partial success if errors occurred but at least some summaries were created, and failure if no summaries were produced at all. The pipeline is scheduled to run daily at 01:30 UTC. Figure 10 presents the sequence diagram of the article summarisation pipeline.

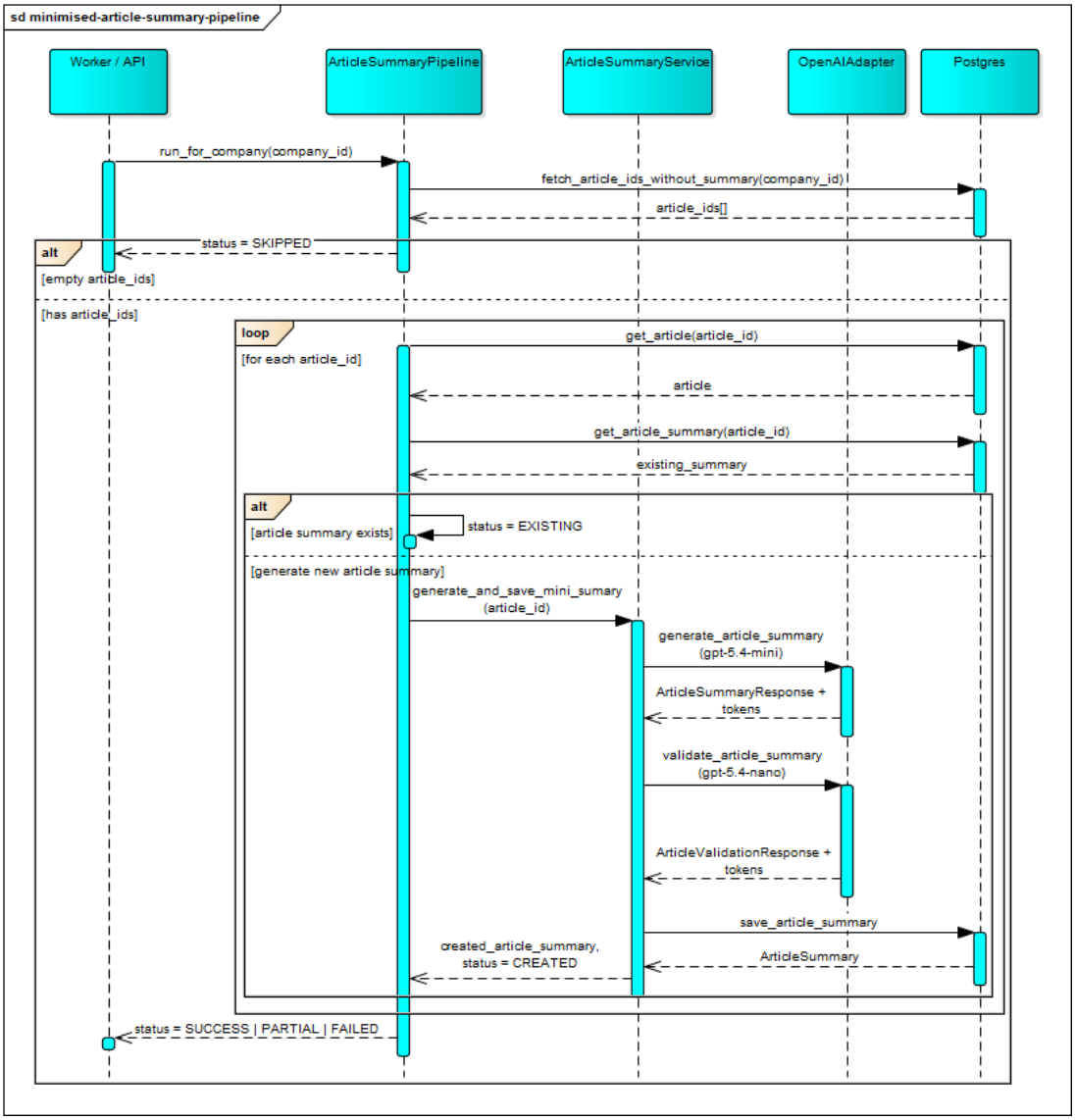


Figure 10. Article summary pipeline sequence diagram

The financial report pipeline handles the structured extraction of annual and quarterly financial reports. Each run begins by finding the most recent financial report URL for a company using the company's ISIN code. The result URL is then compared against the URL of the most recently stored financial report summary for that company. If the URLs match, the report has already been processed, and the pipeline exits with a SKIPPED status. If a new URL is detected, the pipeline sends the report to the LLM for structured data extraction. The LLM returns a typed response containing income statement metrics, balance sheet positions, cash flow data, business segment breakdowns and management commentary. A second LLM call then validates the extracted data by checking that all required fields exist and that the numbers and dates are logical. The validation step returns a step-by-step analysis, a data quality

assessment, and a confidence score. The report is persisted to the database regardless of the validation outcome, with a verified flag indicating whether the validation passed. The pipeline is scheduled to run daily at 03:00 UTC. The sequence diagram of the financial report pipeline is presented in Appendix 2.

The company summary pipeline produces the LLM-generated company overview that is displayed on the summary page. The pipeline has a knowledge lookback window of 180 days to ensure that only the latest and most relevant information is used to generate the company summary. This window is configurable through environment variables. Processing happens in two stages. In the first stage, a structured overview object is assembled by taking the latest 20 article summaries that are not older than the lookback window and the latest extracted financial report data for a company from the database. In the second stage, this overview is passed to the LLM, which produces a structured response containing a company overview, financial facts, article context and executive takeaways. The LLM is also asked to classify the overall tone of the summary based on financials and article sentiment. Unlike the previous pipelines, the company summary pipeline does not include a dedicated validation step. This decision was made because the inputs for the company summary are individually verified at earlier pipeline stages, which in practice resulted in the model producing accurate company summaries without requiring an additional verification pass. Administrators are able to manually trigger summary generation for individual companies through the API, with rate limits applied to control the frequency of generation. The pipeline always generates a new summary rather than updating an existing one and the summary history is maintained. The history of summaries serves both regular users, who can view how a company overview has changed over time, and administrators, who can use it for auditing purposes. The pipeline is scheduled to run every two weeks. The sequence diagram of the company summary pipeline is presented in Appendix 3.

4.5 LLM operations

The LLM operations are responsible for three distinct tasks: summarising individual news articles, extracting structured financial data from reports and generating a consolidated company overview for retail investors. Each task is implemented as a separate service that communicates with the LLM through a common interface. Before

generating the final company overview, a single structured input is used by fetching up to 20 most recent and verified article summaries and the latest extracted financial report from the database.

4.5.1 Prompt engineering

All system prompts are stored as plain text files and loaded at runtime, allowing prompt content to be updated without requiring code changes. Three separate system prompts are used for main functionalities of the application: one for article summarisation, one for financial report extraction and one for company overview generation. All prompts are written using XML format and prompting patterns and guidelines outlined in the latest OpenAI GPT-5.4 prompting guide [23].

Each prompt opens with a role definition to constrain the model's tone, vocabulary choice and the definition of what counts as relevant content. As an alternative to using thinking tokens, all prompts use a completeness contract and verification loop sections, which are recommended by the GPT-5.4 prompting guidance [23].

The company summary prompt uses a source priority tag (Figure 11) to define a strict priority order for resolving conflicting financial information. Financial report data is chosen over information derived from articles and more recent information has higher importance than older one. Articles are restricted using article usage rules tag to only provide context, sentiment signals and recent event information. This boundary prevents the model from switching confirmed financial metrics derived from the financial reports with editorial commentary or opinions.

```
<source_priority>
      1. The financial report is the primary source for financial
      performance and facts.
      2. Articles provide context, sentiment, and recent
      developments.
      3. If there is a conflict, prefer the report unless an article
      clearly describes a newer event.
</source_priority>
```

Figure 11. Company summary source priority section

All prompts include a critical rules section that helps ground the model's response by reducing fabricated references and unsupported claims. All prompts require that the model preserves numbers, currencies, dates and percentage values exactly as they

appear in the source material. The financial report prompt adds a seven-step internal verification loop (Figure 12) that the model must pass before producing output, verifying that the financial indicators are consistent with reported values, the currency usage is uniform throughout and that the date ranges are logical.

```
<verification_loop>
      Before finalizing output, perform these checks in order:
      1. Required fields check: Are company_name, report_period (all
      sub-fields), currency, management_commentary.business_summary,
      and data_completeness populated?
      2. company_profile check: Is
      company_profile.business_description populated with at least 5
      sentences of substantive content? Is it specific to this
      company (not generic boilerplate)?
      3. Period consistency check: Does period_start predate
      period_end? Does fiscal_year match the year of period_end (for
      annual reports)?
      4. Currency denominations check: Does every financial metric
      value include a currency unit or denomination?
      5. Income statement minimum check: Does income_statement
      contain at least one revenue entry and one profit/loss entry?
      6. Balance sheet minimum check: Does balance_sheet contain at
      least one total assets entry and one equity entry?
      7. data_completeness accuracy check: Does the
      data_completeness value accurately reflect what was actually
      extracted?
      If any check fails, correct the output before returning it. Do
      not return an incomplete response.
</verification_loop>
```

Figure 12. Financial report verification loop

4.5.2 LLM execution and response handling

All language model interactions are routed through a single port interface (LLMPort) and implemented by OpenAIAdapter. Every API call uses OpenAI's structured outputs by passing a Pydantic model as a parameter to the OpenAI's responses API. Figure 13 shows the Pydantic model used to define the expected structure of the financial report response, and Figure 14 shows its usage in the corresponding OpenAIAdapter method.

```
class FinancialReportResponse(BaseModel):
    company_name: str
    report_period: ReportPeriod
    currency: str
    company_profile: CompanyProfile
    income_statement: list[FinancialMetric] =
Field(default_factory=list)
    balance_sheet: list[FinancialMetric] =
Field(default_factory=list)
    cash_flow: list[FinancialMetric] = Field(default_factory=list)
    segments: list[BusinessSegment] = Field(default_factory=list)
    management_commentary: ManagementCommentary
    key_events: list[str] = Field(default_factory=list)
    data_completeness: Literal["complete", "partial", "limited"]
    completeness_notes: str | None = None
```

Figure 13. Financial report response Pydantic model

```
def generate_financial_report_summary(
      self, system_prompt: str, report_url: str
) -> LLMResponse[FinancialReportResponse]:
      try:
            content = [
                  {"type": "input_text", "text": "Analyze the
            attached financial report PDF and extract the structured
            data as specified in your instructions."},
                  {"type": "input_file", "file_url": report_url}
            ]

            response = self._client.responses.parse(
                model=settings.LLM_MODEL_FINANCIAL_REPORT,
                instructions=system_prompt,
                input=[{"role": "user", "content": content}],
                max_output_tokens=8192,
                text_format=FinancialReportResponse,
            )

            if not response.output_parsed:
                  raise LLMException("No parsed financial report
            output returned")

            return LLMResponse[FinancialReportResponse](
                  data=response.output_parsed,
                  tokens=TokenUsage(
                        input_tokens=response.usage.input_tokens,
                        output_tokens=response.usage.output_tokens,
                        total_tokens=response.usage.total_tokens,
                  )
            )
      except OpenAIError as e:
            raise LLMException("OpenAI API exception during
      financial report summarization") from e
```

Figure 14. Generate financial summary report method

Most major LLM providers currently allow models to return structured outputs. Structured output is a feature that ensures the model will always generate responses that adhere to a supplied JSON Schema. This allows developers to prevent the risk of

the model omitting a required key or hallucinating an invalid value. One major benefit of using structured outputs is that you do not need to validate or retry incorrectly formatted responses, as the model does it itself. Because of reliable type-safety, the prompts do not need to be strongly worded to achieve consistent formatting. [24]

Both article summarisation and financial report extraction use a two-step workflow, which consists of generation and validation steps. After the primary generation call produces a structured response, a second independent LLM call with a different model receives both the original source and the generated output and performs a tonality and factual correctness check. For example, it verifies that all the numerical values match the source values and that the response does not try to provide financial advice and is written in a neutral tone. Token usage (input, output and total tokens) is tracked for every generation and validation and stored in the database to support cost analysis and resource planning. The complete flow of all LLM operations, from article summarisation and financial report extraction to company overview generation, is illustrated in Figure 15.

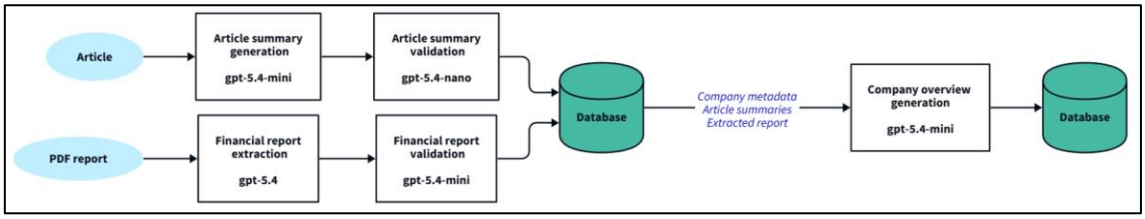


Figure 15. High-level LLM operations

4.6 User management

The initial application that was built as an early prototype for the team project had minimal user-facing features. Since it was only accessible for the internal bank employees, account creation was managed by administrators. This prevented independent user signups and ensured high security. However, to become a publicly usable platform, the system required a way for users to independently create and access their own accounts through a process that was both secure and user friendly.

The original solution stored user session JWT tokens client-side, meaning that the JWT payload was decodable for anyone on the same network. This made the application vulnerable to cross-site scripting attacks. To prevent this Nuxt cookie encryption was

implemented to ensure that only encrypted data was stored client-side. The Nuxt server is responsible for decrypting the data and communicating it to the FastAPI backend.

The entire authentication logic for regular users was transformed to work on Google accounts via OAuth. This simplified the authentication process for users and helped defend the system against malicious endpoint calls related to the authentication logic. Upon a request to sign in with Google, the Nuxt server first redirects to a Google popup, which prompts the user to choose a Google account to sign in with. Once the user selects an account, Google authenticates them and redirects back to the Nuxt server with an authorisation code appended to the callback URL. The Nuxt server then exchanges this code directly with Google for two tokens: an access token and an ID token. The ID token is a signed JWT containing the user's information, this token is forwarded to the FastAPI backend via a POST request. FastAPI verifies the ID token's signature with Google to confirm its authenticity and extracts the user's email and name. It then looks up or creates a corresponding user record in the application's database and issues its own application-level JWT back to the Nuxt server. The Nuxt server uses this JWT to fetch the full user profile from the backend, then seals both the user data and the token together in an encrypted server-side session cookie using a secret session password. The browser receives this cookie, and the user is marked as authenticated while the application never handles a password.

The addition of personal accounts allowed for the creation of user-facing features, which increase the platform's value and usefulness for the regular user. One such feature is the company watchlist, which enables users to add attractive companies to a watchlist which can then be used for easier filtering when browsing news or summaries. This feature is rather minimal, but the greater focus was on enabling further developments with the implementation of self-service authentication. New features may be added using the existing user system and authentication logic.

4.7 API and user interface

As the core backend features and data pipelines were completely automated, the endpoints accessible to the regular user were brought to a minimum. Excluding authentication related endpoints, users have access to simple READ operations which

allow searching for companies and fetching existing summaries or news from the database as well as POST methods for the user-facing features, such as adding companies to the watchlist (Appendix 4). Endpoints that enable more complex operations, such as creating individual summaries or initiating web scraping pipelines were kept to improve development and testing but were made accessible only to admin or rootadmin users (Appendix 5). Admin users also have endpoints for performing basic CRUD (*Create, Read, Update and Delete*) operations on companies and articles for fixing possible mistakes in the data, manually triggering pipelines as well as source-specific scraper initiation endpoints to gather URLs by company and scraping text from the gathered URLs for each source.

The user-accessible endpoints are available through the frontend interface built with Nuxt 4. The interface consists of three primary views and a home page. All three primary pages contain a company filtering section which is identical for all pages to simplify the learning process of the application for the user. Companies are filterable by company name, ticker symbol, country and industry as well as the user's own watchlist. The summary view allows viewing LLM-generated summaries. Upon selecting a company, the most recent summary for that company is displayed, with the option to go back and view older summaries. The sources used for the summary are referenced as clickable hyperlinks, which also show the article summary title in English when hovering. The user can navigate through the summary's three sections using the tabs at the top of the summary. Dividing the content into these focused segments enhances readability and prevents the information from feeling overwhelming, keeping the user engaged and motivated to read. The first tab of the summary is displayed in Figure 16.

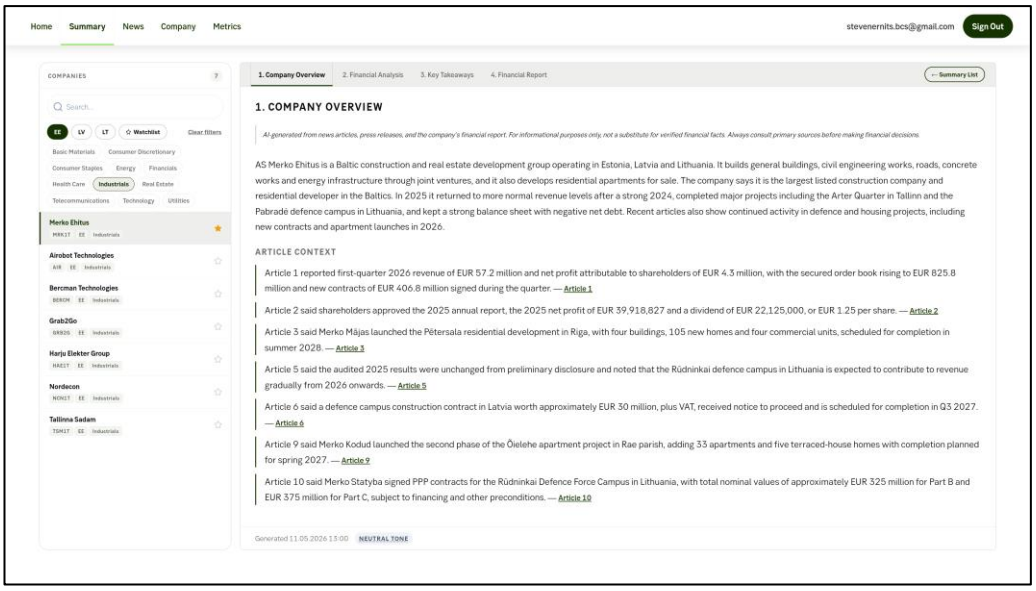


Figure 16. Summary view

An additional fourth tab displays the summary of the financial report used for the loaded company summary. This is the most comprehensive tab, designed for users seeking deeper analytical insights, as it provides detailed financial data including in depth descriptions of the report's content and tables for the report's income statement, balance sheet, and cash flow (Appendix 6).

The news view shows all stored articles related to a company. Each article is presented with its title, publishing date, source and options to either view the LLM-generated article summary (Appendix 7) or navigate to the original source as shown in Figure 17.

Figure 17. News view

The company view displays the selected company's stock price chart, which also includes news or press release publications as highlighted points directly on the chart itself. Hovering over the points shows the article headline and allows navigation to the original source. To the right of the chart is the company's overview, which is fetched from the company's latest summary and five of the latest news articles for the company. The company view is presented in Figure 18.

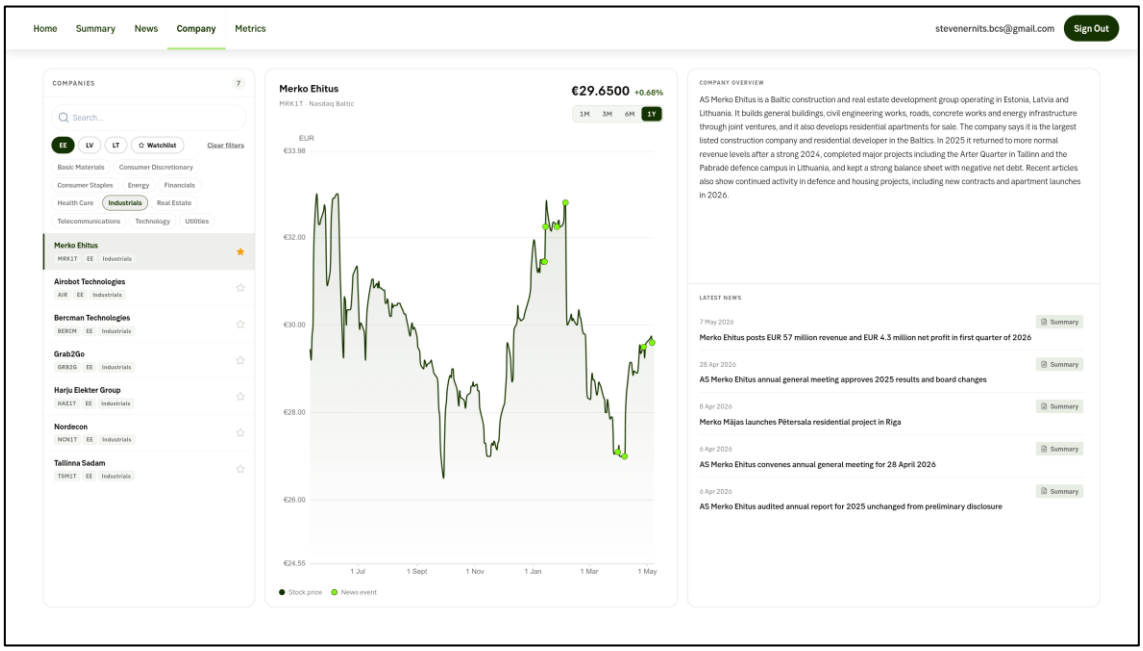


Figure 18. Company chart view

Admin users have access to an additional metrics view to easily monitor the token usage of the LLM processes over a desired time period. The metrics view provides filtering by date range and LLM process. Token usage details are divided into three separate sections: the date range section, today section and yesterday section, each one shows input, output, and total tokens as well as the estimated total cost and cost for each individual process. Figure 19 presents the metrics view.

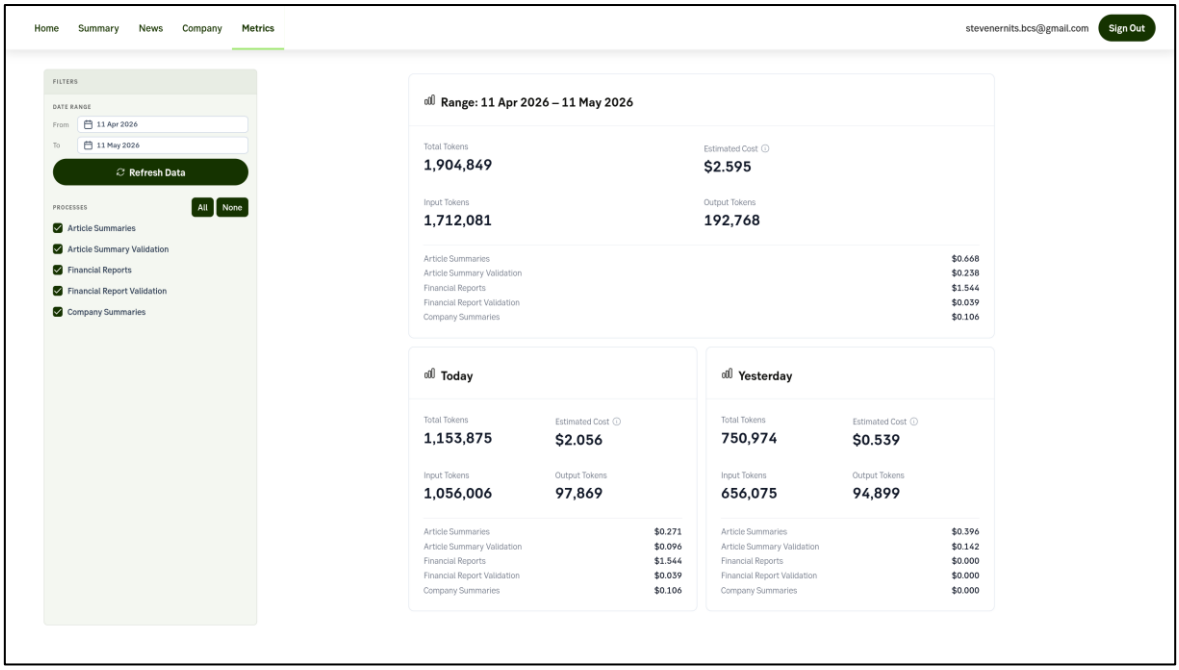


Figure 19. Metrics view

4.8 Production and Deployment

For the production environment, the application was deployed on a Virtual Private Server (*VPS*) hosted by OVHcloud, alongside a registered domain name. This provider was selected due to the authors' prior experience with the platform and its highly competitive pricing. An OVHcloud VPS proved to be significantly more cost-effective than alternative VPS providers and substantially cheaper than fully managed cloud platforms such as Microsoft Azure or AWS. Although configuring a VPS requires more manual setup and system administration compared to managed cloud environments, it was determined that the financial savings justified the additional operational complexity.

To ensure environmental consistency, the production setup closely mirrors the local development environment, running the application services (frontend, backend, Celery workers, Celery beat, Redis, and Adminer) within Docker containers. However, to prioritise data persistence and performance, the PostgreSQL database was installed and configured directly on the host VPS rather than within a container.

To guarantee high availability and automated recovery, a custom systemd service was created. This service natively integrates the application's lifecycle with the host operating system, ensuring that all Docker containers gracefully stop and automatically restart during server maintenance or unexpected reboots.

Nginx was used as a reverse proxy to manage incoming web traffic and route requests to the appropriate internal Docker containers based on configured subdomains. Significant emphasis was placed on securing the public facing application. SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) certificates were created using Certbot to enforce HTTPS (*Hypertext Transfer Protocol Secure*) encryption and redirect all unencrypted HTTP (*Hypertext Transfer Protocol*) traffic. Network level access control was implemented, because the publicly accessible server quickly became a target for automated vulnerability scanners looking to exploit the system. Using the Nginx GeoIP2 module, incoming connections were restricted exclusively to Baltic IP addresses, actively dropping unauthorised traffic.

Furthermore, comprehensive HTTP security headers, including Strict-Transport-Security, Content-Security-Policy and X-Frame-Options, were integrated into the server configuration to mitigate common web vulnerabilities. Finally, basic authentication was added to the database management interface, and global rate limiting was configured to prevent server abuse.

5 Analysis and Conclusions

This chapter provides analysis of the finished project and the work process: what went well, what was difficult, how the application performed in real user testing and what could be the project's future developments. Additionally, as thorough testing and assessment of various LLMs were done on the application's use cases, the results of this assessment are also described in this chapter.

5.1 Assessment of implementation outcomes

Refactoring to hexagonal architecture was justified by the resulting decoupling between business logic and external services. Switching to a different LLM provider or adding new sources of news can now be done with minimal code changes that do not affect the core logic of the application. Communication with the database was left out of the hexagonal pattern, as introducing ports and adapters for persistence operations would have added unnecessary clutter and defeated the purpose of using SQLModel as an ORM, whose entire goal is to make those operations simpler. Switching the underlying database to a different one was not considered a realistic future requirement.

The transition to structured outputs through Pydantic schemas was one of the most impactful technical changes. The original prototype forced the LLM to produce a response that already contained HTML formatting tags directly in the text, because the model did not reliably follow JSON formatting rules at the time. The resulting text was saved in the database as is, which tightly coupled the data layer to a specific frontend presentation format. The structured output approach decoupled the stored data format from the presentation layer entirely, meaning that the frontend rendering can now be changed independently of how the data is stored. Adding new fields or modifying existing sections of a summary became a matter of updating the Pydantic schema and the rendering logic, without affecting the rest of the pipeline. The structured format also significantly improved the reliability of LLM responses. During development, the

authors observed more consistent and higher quality outputs once the structured output was used.

Decoupling financial report summarisation from company summary generation produced a large performance gain in the new system. Because reports are only published four times a year per company while company summaries are regenerated more frequently, reusing the already extracted report summary avoids sending the full report to the LLM on every run. This reduced the average response time of company summary generation from several minutes down to seconds and made the system progressively cheaper to operate as more summaries are produced, without reducing the quality of the financial information extracted from the report. The change also enabled a new user-facing feature: the dedicated financial report tab on the summary view, which provides users with structured access to the extracted income statement, balance sheet, cash flow and management commentary data. As a result, all three input categories are now accessible to the user in an LLM summarised format: news articles, press releases and financial reports. Table 1 presents the approximate average response time and cost per call for each LLM process used in the production system.

Table 1. Tasks, used models and statistics

Process	**Model**	**Avg. cost ($)**	**Avg. response time (s)**
Article summarisation	GPT-5.4-mini	0.0021	2.67
Article summary validation	GPT-5.4-nano	0.0005	2.02
Financial report summarisation	GPT-5.4	0.1592	42.43
Financial report summary validation	GPT-5.4-mini	0.0041	4.59
Company summarisation	GPT-5.4-mini	0.0093	6.77

Splitting background processing into three country specific Celery queues improved the throughput of the system compared to the team project version. Previously, all tasks went through a single shared queue with up to four tasks being processed concurrently, but this introduced a risk of conflicts when two workers attempted to process the same news article at the same time, since some news sources reference multiple companies in a single article. The authors observed that articles published in one Baltic country rarely reference companies from another Baltic country, which

made country-specific partitioning safe for parallel task execution. The new structure allows three companies to be processed in parallel, one per country, which directly improved the overall pipeline runtime. If additional countries are added in the future where news sources cross-reference companies across borders, additional coordination logic such as database level article locking would need to be introduced. Country-specific partitioning also keeps the queues isolated from each other, so a slowdown in one country's pipeline does not affect the other two.

One of the greatest challenges in the project was finding an optimal trigger for company summary regeneration. The authors initially wanted to use the LLM itself to classify the importance of each incoming news article on a numerical scale and to trigger a new company summary only when an important article was published or when a new financial report had been processed. This approach would have guaranteed that a new summary was only produced when there was indeed any new and meaningful information to include. However, the authors were unable to develop a prompt that would have produced stable importance scores. The model consistently graded news either too high or too low. Several prompt variations were tested, but none of them produced sufficiently reliable scores to be used as an automatic trigger. In production, a fixed twice a month generation interval was chosen, based on the observation that the average news volume for Baltic companies is relatively low. Solving this problem is one of the more important future development directions.

5.2 LLM selection and comparison

A comparison assessment was conducted using the weighted sum method on the following models: GPT-5.4, GPT-5.4-mini, GPT-5.4-nano, Gemini 3.1 Pro Preview, Gemini 3 Flash Preview, Gemini 3.1 Flash Lite Preview, Claude Sonnet 4.6, Claude Haiku 4.5. The models were tested separately for three processes: article summary generation, financial report summary generation and company summary generation. The models were divided into comparison groups for each task based on their pricing tier and intended use case outlined by the model providers. Larger models were more suitable for complex processes such as financial report extraction, while smaller models were suitable for summary and article summary generation. The tasks and tested models are presented in Table 2.

Table 2. Tasks and tested models

Task	**Tested models**
Article summary generation	GPT-5.4-mini GPT-5.4-nano Gemini 3 Flash Preview Gemini 3.1 Flash Lite Preview Claude Haiku 4.5
Financial report summary generation	GPT-5.4 GPT-5.4-mini Gemini 3.1 Pro Preview Gemini 3 Flash Preview Claude Sonnet 4.6 Claude Haiku 4.5
Company summary generation	GPT-5.4-mini GPT-5.4-nano Gemini 3 Flash Preview Gemini 3.1 Flash Lite Preview Claude Haiku 4.5

Each model was tested on the same input companies, and the results were evaluated with a weighted sum method. The assessment considered output quality, estimated cost per call and response speed. Quality was assessed with task-specific checklists, cost and speed were calculated from the measured token usage and response time. In all three assessment tables below, the quality score and the weighted total are reported on a 5-point scale, where 5 represents the best possible result. Average cost and average response time are shown as raw measured values for readability, but the weighted total is computed from their normalised scores.

For article summary generation, GPT-5.4-mini performed best overall with a weighted score of 4.85. It also received the highest quality score and the best response speed score. GPT-5.4-nano was the cheapest model for this process and also performed well with a score of 4.64, but its quality was lower than GPT-5.4-mini. The weakest result was Gemini 3.1 Flash Lite Preview with a score of 3.14, mainly because of slow response time. Therefore GPT-5.4-mini remains the model of choice for this task. The results of article summary generation assessment are presented in Table 3.

Table 3. Article summary generation assessment

Model	Quality score	Avg. cost ($)	Avg. response time (s)	Weighted total
GPT-5.4-mini	5.00	0.0021	2.67	4.85
GPT-5.4-nano	4.50	0.0006	3.16	4.64
Claude Haiku 4.5	4.80	0.0033	4.76	4.16
Gemini 3 Flash Preview	4.90	0.0048	6.09	3.61
Gemini 3.1 Flash Lite Preview	4.80	0.0007	9.63	3.14

For financial report summary generation, the winning model was also GPT-5.4-mini with a score of 4.89. The currently used model for this task is GPT-5.4, which received a lower score of 4.25. GPT-5.4 had the best quality score, but the difference compared to GPT-5.4-mini was small, the mini model was significantly faster and cheaper. However, GPT-5.4 should perform better when there is greater ambiguity in the data, as OpenAI's own prompting guidance notes that GPT-5.4-mini is weaker on ambiguity handling compared to the full model [23]. This is the reason why the authors chose to use the larger model over the mini model. Gemini 3 Flash Preview had the best cost score, but its response time was much slower than GPT-5.4-mini. The worst model in this category was Claude Sonnet 4.6 with a score of 2.91. It was the slowest and most expensive model in these tests. Gemini 3.1 Pro Preview was not considered usable in practice, as it often did not respond due to model availability issues. Table 4 presents the financial report summary generation assessment.

Table 4. Financial report summary generation assessment

Model	Quality score	Avg. cost ($)	Avg. response time (s)	Weighted total
GPT-5.4-mini	4.95	0.0449	19.54	4.89
GPT-5.4	5.00	0.1592	42.43	4.25
Gemini 3 Flash Preview	4.44	0.0253	61.15	4.08
Claude Haiku 4.5	4.33	0.0943	36.03	4.06
Gemini 3.1 Pro Preview	4.44	0.0986	55.26	3.95
Claude Sonnet 4.6	4.74	0.3016	98.89	2.91

For company summary generation, GPT-5.4-mini achieved the highest score of 4.81. It had the highest quality and response speed score. Gemini 3.1 Flash Lite Preview had the best cost score, but its response time was weak. Claude Haiku 4.5 had the worst score for this process with 3.12 out of 5, mainly due to it being the slowest and most expensive model in this test. The currently used model GPT-5.4-mini is the strongest choice for company summary generation. Table 5 presents the company summary generation assessment.

Table 5. Company summary generation assessment

Model	Quality score	Avg. cost ($)	Avg. response time (s)	Weighted total
GPT-5.4-mini	5.00	0.0093	6.77	4.81
GPT-5.4-nano	4.86	0.0031	11.94	4.35
Gemini 3 Flash Preview	4.86	0.0089	13.14	3.91
Gemini 3.1 Flash Lite Preview	4.86	0.0027	15.29	3.85
Claude Haiku 4.5	4.86	0.0165	19.77	3.12

The assessment showed that the OpenAI models were the most reliable for the structured pipelines used in this application. GPT-5.4-mini outperformed other models because it provided high quality responses with reasonable speed and cost. GPT-5.4 produced the highest quality score and GPT-5.4-nano performed strongly relative to its price tier. The results also showed that using GPT-5.4-mini for financial report summaries could be considered, because it had almost the same quality rating as GPT-5.4 while being faster and cheaper.

The Claude models were not suitable for the current use cases. Their quality scores were sometimes acceptable, but the models were very expensive, especially in larger tasks like the financial report summary generation. They also had inconsistent language behaviour and sometimes added random non-English words into an otherwise English output, trying to reference the source data which may have been in non-English languages. This would be problematic for the system, because all the generated content must be clean, predictable and suitable for displaying to the user.

The Gemini models were also not selected. While some Gemini models were cost-effective, the results were not reliable enough for consistent use. The biggest issues were accessibility and response reliability, as Gemini 3 models often did not respond. Additionally, the Gemini models had the same language inconsistency issues as the Claude models, which made them even less suitable for the application.

In conclusion, GPT-5.4-mini was the best general model for most LLM tasks based on this test. It achieved the highest score for all the tested processes and could also be considered as a replacement for GPT-5.4 in the financial report summary generation process. GPT-5.4 was used for financial report extraction, which requires maximum accuracy, because it produced the highest quality score and OpenAI's documentation supports its stronger handling of ambiguous input, although GPT-5.4-mini could be a credible replacement candidate. GPT-5.4-nano performed strongly relative to its price tier and was therefore used for lighter workloads, specifically the validation steps of the article and financial report pipelines. Claude and Gemini models proved unsuitable for this application due to their cost, reliability and language consistency issues.

5.3 Feedback from user testing

User testing was conducted to validate whether the developed application helps address the information accessibility problem. The aim of the testing was not to prove that the application directly leads to better investment decisions, but to evaluate whether it helps users gain an initial overview of a company, understand the available information and decide what should be researched further.

Six feedback sessions were completed. The participants were selected to cover a range of investing experience: one experienced investor, two users with intermediate experience, one user with minimal investing experience, and two users with no prior investing experience. This variation was useful because the application is intended to support both beginner investors who may not know where to start, and more experienced users, who may want to reduce the time spent collecting information from multiple sources.

Each session followed the same structure. The participant first rated their existing knowledge of a chosen company, then performed 20 minutes of manual research using public Baltic news from predetermined sources, then took a short break and after that used the developed application for 20 minutes with the same goal. The average ratings in each assessment category, with a maximum rating of 10, are shown in Table 6.

Table 6. Average feedback ratings

Category	**After initial research**	**After using the application**	**Change**
Knowledge of the company	5.17	6.17	+1.00
Accessibility of information	6.00	8.67	+2.67
Understandability of information	5.17	7.67	+2.50
Relevance of information	6.33	8.67	+2.33
Time sufficiency	4.33	7.00	+2.67
Motivation to research further	3.50	6.00	+2.50
Ability to decide what to research next	5.00	6.00	+1.00
Confidence in making an investment decision	3.67	6.00	+2.33

In all the eight categories there was an improvement on average, but not evenly. The largest improvements were in accessibility (+2.67), time sufficiency (+2.67), understandability (+2.50) and motivation to research further (+2.50). This supports the assumption that the application reduces the effort required to form an initial understanding of a Baltic-listed company. The increase in time sufficiency is especially relevant, because one of the main problems addressed by the project is that financial information is scattered across several sources and often requires significant manual work to process.

The biggest improvements were observed for the users with no prior investing experience. Both rated time sufficiency at 1 after the manual research phase and described it as frustrating and unmotivating, since they could not find the correct information within the available time. After using the application, one of them improved across every category, with the largest increases in time sufficiency (1 to 8) and confidence (1 to 8). The same user noted in the written feedback that the application made it possible to read enough about a company in the available time. The other user saw similar gains in accessibility and understandability and described the application as compact and logical. This pattern suggests that the application provides the most value for users who do not yet know where to search for relevant information or how to interpret longer financial materials. Both noted that if they were to start investing, they would prefer the developed platform, since it consolidates the information into one place.

The user with minimal investing experience also showed clear improvement. Their rating for understandability increased from 4 to 8, time sufficiency from 6 to 9 and confidence from 5 to 8. This indicates that the summarised format helped make the information more understandable even for a user who had some prior investing experience but did not have a strong technical background in financial analysis. The same user described the application as a very useful tool for first familiarisation with a company and more efficient than manual research.

The feedback from the more experienced users was more mixed. The experienced investor gave high ratings after using the application, with motivation to research further increasing from 4 to 8, ability to decide what to research next from 8 to 10 and confidence from 7 to 10. This user also suggested more advanced filtering options

based on the personal investment strategy and more visualisation of data. Among the two users with intermediate investing experience, one gave more critical ratings, with some categories remaining unchanged or decreasing. Their written feedback still described the application as good and consolidating, while the main criticism concerned navigation clarity, filtering options and the visibility of interface elements. The other rated the application higher than manual research in most categories and described it as simple, user-friendly and useful for beginner investors, while suggesting more filters, a separate ratios page and more visual material.

The qualitative feedback supports the same overall conclusion as the numeric ratings. Users appreciated that the application collected relevant information into one place and reduced the amount of irrelevant material they had to process manually. The user with minimal investing experience specifically noted that the application did not create the same feeling of being overwhelmed by too many news articles, and that the summaries of financial reports were easier to read than the original documents.

At the same time, the feedback also revealed several usability issues. Multiple users pointed out that the summary views could be visually clearer and better separated into sections. One user described the interface as minimalist and pleasant but also mentioned that the summary text became somewhat monotonous and that different parts of the text were not always easy to distinguish. Another user suggested that risks, financial indicators or other important factors of the summary could be separated into tables or tabs.

Navigation, visibility and some confusion issues were also observed. Users mentioned that watchlist buttons were not immediately understandable, article numbers on the summary page required explanation, and the article titles in the news list should function as hyperlinks. This feedback shows that some interface elements need clearer labels or styling.

Overall, the user feedback supports the usefulness of the developed solution. The application appears to reduce the effort required to collect and understand initial company information, especially for users with less investing experience. The feedback also identifies concrete usability improvements that would further increase the application’s usefulness.

5.4 Future developments

The user feedback revealed multiple future development directions. Users suggested that the summary view could be visually clearer and better separated into sections. Therefore, a future version of the summary page should present key parts of the generated text in a more structured manner. For example, risks, important recent events, financial indicators and management commentary could be separated into clearer tables or additional tabs instead of shown as plain text alongside everything else.

Navigation and interaction clarity should be improved in the user interface. During testing, some users did not immediately understand the meaning of certain buttons, article references or numbers shown in the summary view. Future work should introduce clearer labels and styling, tooltips or short explanatory texts for interface elements that may not be self-explanatory.

The filtering system in the user interface could be improved so that users can more easily narrow down companies or articles based on the type of information they are looking for, for example by news source or financial criteria such as dividend payments.

From the technical analysis, the summary generation process should be improved and the number of data sources should be expanded. In the current version, company summaries are generated on a fixed schedule. This was chosen because attempts to use an LLM to score the importance of news articles did not produce stable enough results. A possible solution for it would be to ground the importance evaluation with the company's stored financial report summary, so that the model could judge incoming news against the company's actual financial data and existing risk factors. With that context, the LLM should not undervalue developments for smaller companies or overvalue small changes for larger ones. Combined with deterministic triggers such as the publication of a new financial report, this would allow company summaries to be generated only when meaningful new information is available.

6 Conclusion

The problem addressed in this thesis was the lack of a publicly accessible tool that consolidates and summarises relevant financial information about companies listed on the Baltic Stock Exchange. Retail investors who want to research these companies must manually search through news articles, press releases and financial reports, which can be time consuming for users with limited experience or resources.

The goal of the project was to further develop an existing internal-use prototype into a publicly usable financial analysis platform. The main objectives were to refactor the backend to a more extendable architecture, automate data processing workflows and improve the application for regular users through user-facing features and a redesigned interface. To achieve this, the project adopted hexagonal architecture, implemented automatic data pipelines, improved user management, and integrated LLM-based summarisation and validation in the automatic workflows.

As a result, the completed application provides users with company summaries, article summaries, financial report summaries, stock price charts with related news markers and a personal watchlist functionality. Overall, the project achieved its main objectives and produced a more scalable, robust and publicly usable platform, while leaving room for future improvements such as broader market coverage, additional data sources and new user-facing features.

References

[1] T. Akana, M. Drayton and A. Lee, “Why Some Americans Don’t Invest in the Stock Market,” September 2025. [Online]. Available: https://www.philadelphiafed.org/-/media/FRBP/Assets/Consumer-Finance/Briefs/Why-Some-Americans-Dont-Invest-in-the-Stock-Market.pdf. [Accessed February 2026].

[2] Blackrock, “Bridging the gap: From saving to investing,” December 2025. [Online]. Available: https://www.blackrock.com/uk/literature/whitepaper/bridging-the-gap-report.pdf. [Accessed February 2026].

[3] Blackrock, “People and Money,” November 2025. [Online]. Available: https://www.ishares.com/us/literature/presentation/people-and-money-etf-investors-2025.pdf. [Accessed February 2026].

[4] EFPA, “Response to ESMA Call for Evidence on the Retail Investor Journey,” July 2025. [Online]. Available: https://efpa-eu.org/wp-content/uploads/2025/07/EFPA-Response_ESMA-Call-for-Evidence_Retail-Investor-Journey_July-2025.pdf. [Accessed February 2026].

[5] SEB Bank, “Uuring: Eestis on investeerimine populaarsem kui mujal Baltimaades, eelistatakse kinnisvara,” May 2021. [Online]. Available: https://www.seb.ee/uudised/2021-05-11/uuring-eestis-on-investeerimine-populaarsem-kui-mujal-baltimaades-eelistatakse-k. [Accessed February 2026].

[6] Delfi Ärileht, “Uuring kinnitab: eestlased on valmis ühes kuus investeerima just nii suure summa,” June 2021. [Online]. Available: https://arileht.delfi.ee/artikkel/93865017/uuring-kinnitab-eestlased-on-valmis-uhes-kuus-investeerima-just-nii-suure-summa. [Accessed February 2026].

[7] OECD, “OECD/INFE 2023 International Survey of Adult Financial Literacy,” 2023. [Online]. Available: https://www.oecd.org/content/dam/oecd/en/publications/reports/2023/12/oecd-infe-2023-international-survey-of-adult-financial-literacy_8ce94e2c/56003a32-en.pdf. [Accessed February 2026].

[8] World Economic Forum, “The Future of Capital Markets: Democratization of Retail Investing,” August 2022. [Online]. Available: https://www.weforum.org/publications/the-future-of-capital-markets-democratization-of-retail-investing/. [Accessed February 2026].

[9] Nasdaq Baltic, “List of Markets,” [Online]. Available: https://nasdaqbaltic.com/market-information/about-the-markets/. [Accessed May 2026].

[10] A. Cockburn, “The Hexagonal (Ports & Adapters) Architecture,” 2005.

[11] D. Svrtan and S. Makagon, “Ready for changes with Hexagonal Architecture,” March 2020. [Online]. Available: https://netflixtechblog.com/ready-for-changes-with-hexagonal-architecture-b315ec967749. [Accessed February 2026].

[12] J. Palermo, “The Onion Architecture : Part 1,” July 2008. [Online]. Available: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/. [Accessed March 2026].

[13] R. C. Martin, “The Clean Architecture,” August 2012. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Accessed March 2026].

[14] M. A. Khder, “Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application,” November 2021. [Online]. Available: https://www.i-csrs.org/Volumes/ijasca/2021.3.11.pdf. [Accessed February 2026].

[15] Georgetown University, “Web Scraping a Corpus,” March 2026. [Online]. Available: https://guides.library.georgetown.edu/c.php?g=1201872&p=10272362. [Accessed March 2026].

[16] IAPP, “The state of web scraping in the EU,” July 2024. [Online]. Available: https://iapp.org/news/a/the-state-of-web-scraping-in-the-eu. [Accessed March 2026].

[17] A. Vaswani, “Attention Is All You Need,” August 2023. [Online]. Available: https://arxiv.org/pdf/1706.03762. [Accessed March 2026].

[18] J. Lee, N. Stevens, S. C. Han and M. Song, “A Survey of Large Language Models in Finance (FinLLMs),” February 2024. [Online]. Available: https://arxiv.org/pdf/2402.02315. [Accessed March 2026].

[19] Y. Nie, “A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges,” June 2024. [Online]. Available: https://arxiv.org/pdf/2406.11903. [Accessed March 2026].

[20] A. Kim, M. Muhn and V. Nikolaev, “Financial Statement Analysis with Large Language Models,” May 2024. [Online]. Available: https://bfi.uchicago.edu/wp-

content/uploads/2024/05/Financial-Statement-Analysis-with-Large-Language-Models.pdf. [Accessed March 2026].

[21] D. Anh-Hoang, V. Tran and L. M. Nguyen, “Survey and analysis of hallucinations in large language models: attribution to prompting strategies or model behavior,” September 2025. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC12518350/. [Accessed March 2026].

[22] S. Hiriyanna and W. Zhao, “Multi-Layered Framework for LLM Hallucination Mitigation in High-Stakes Applications: A Tutorial,” August 2025. [Online]. Available: https://www.mdpi.com/2073-431X/14/8/332. [Accessed April 2026].

[23] OpenAI, “GPT-5.4 prompting guide,” [Online]. Available: https://developers.openai.com/api/docs/guides/prompt-guidance?model=gpt-5.4. [Accessed May 2026].

[24] OpenAI, “Structured model outputs,” [Online]. Available: https://developers.openai.com/api/docs/guides/structured-outputs. [Accessed May 2026].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis

Authors Rene Väljaots and Steven Ernits

1 Grant Tallinn University of Technology free licence (non-exclusive licence) for the thesis “Publicly Accessible Platform for Company Financial Analysis Using Large Language Models”, supervised by Karl-Erik Karu

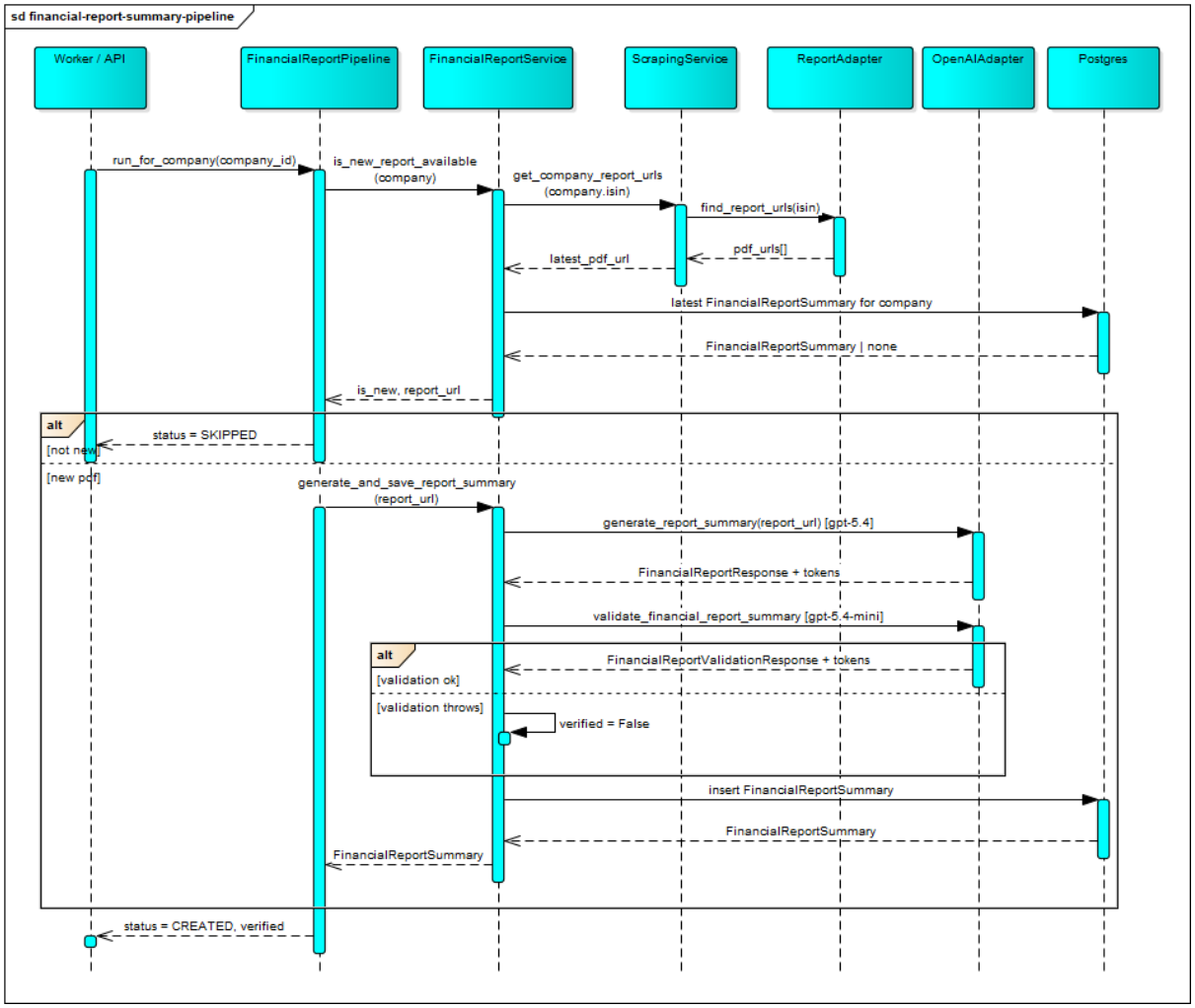
1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

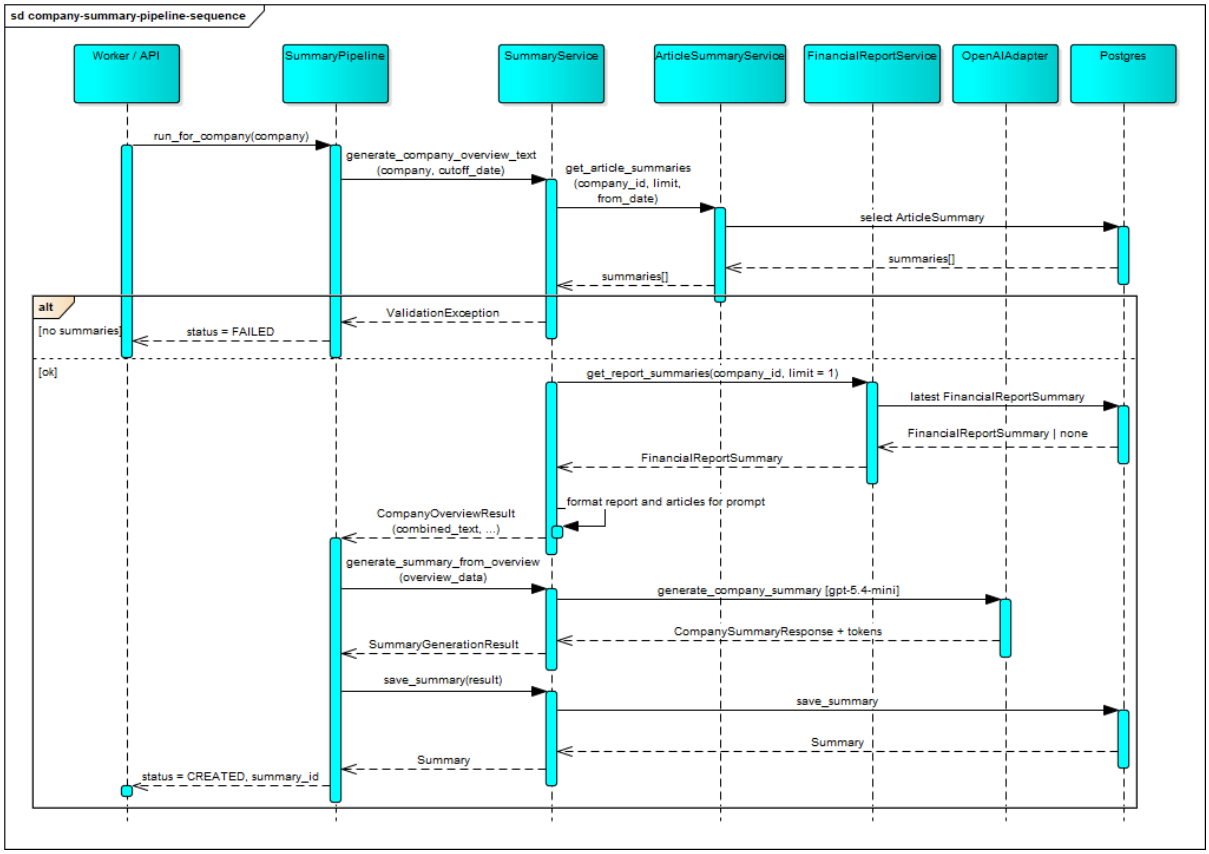
2 The authors are aware that the authors also retain the rights specified in clause 1 of the non-exclusive licence.

3 The authors confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

Appendix 2 – Financial Report Pipeline



Appendix 3 – Company Summary Pipeline



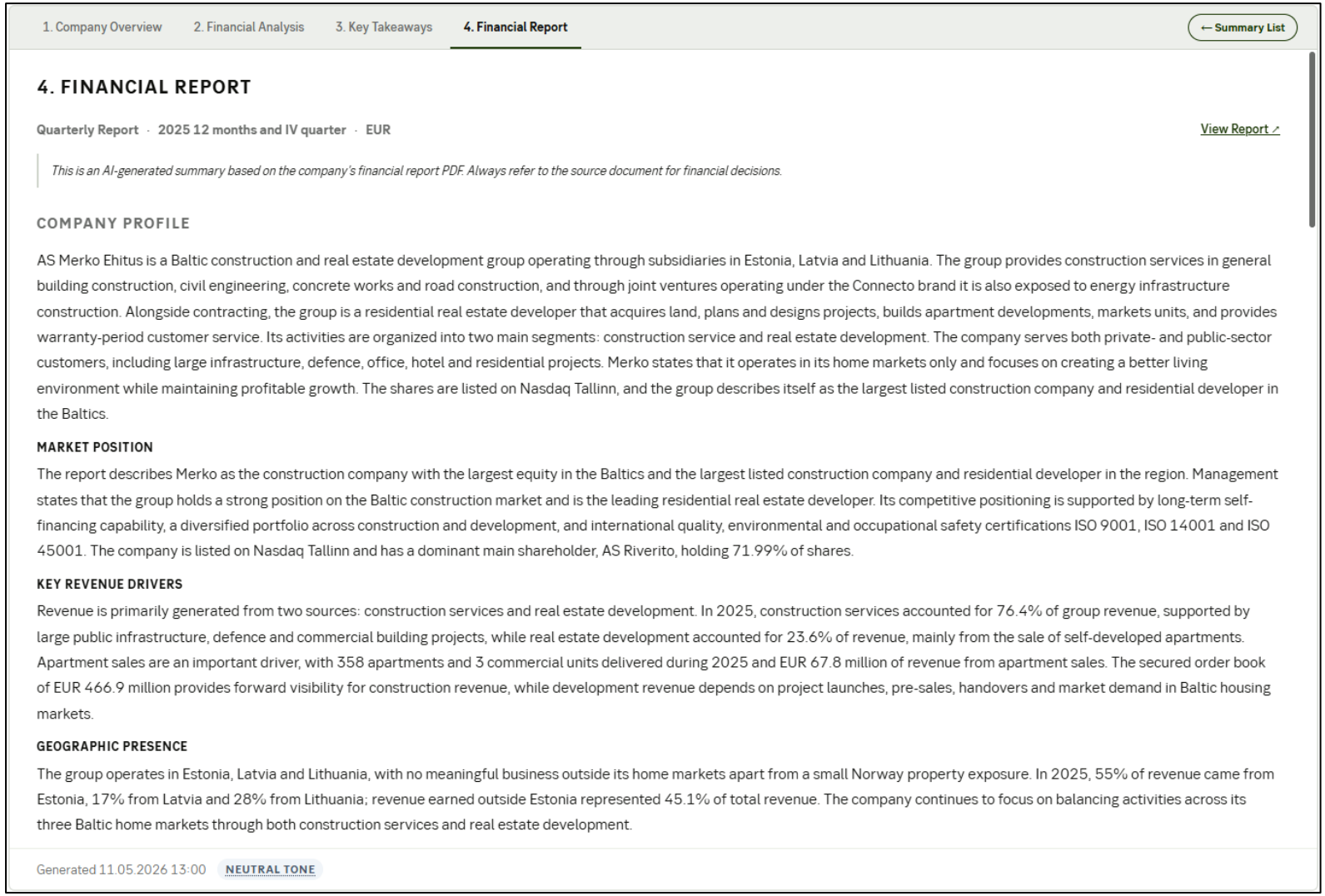
Appendix 4 – User accessible endpoints

Method	Endpoint	Description	Access
POST	/api/v1/login/google	Google OAuth log in	Public
GET	/api/v1/users/me	User data for session	User
GET	/api/v1/companies	Fetch all companies	User
GET	/api/v1/companies/{id}	Fetches details for a company by id	User
GET	/api/v1/summaries	Fetches summaries with optional filtering	User
GET	/api/v1/summaries/ history	Fetches stored summaries as minimal objects for company	User
GET	/api/v1/summaries/ {summary_id}	Fetches selected summary by id	User
GET	/api/v1/financial-report-summaries	Fetches financial report summaries by company id with optional filtering	User
GET	/api/v1/financial-report-summaries/by-report-url	Fetches financial report summary by report URL	User
GET	/api/v1/articles	Fetches articles, filterable by company id	User
GET	/api/v1/articles/{id}	Fetches article by id	User
GET	/api/v1/article-summaries	Fetches article summaries with optional filtering by company id or article id	User
GET	/api/v1/article-summaries/{id}	Fetches article summary by id	User
POST	/api/v1/article-summaries/batch	Fetches a batch of article summaries by ids to display as references for a summary	User
GET	/api/v1/watchlist	Gets user watchlist	User
POST	/api/v1/watchlist/ {company_id}	Adds a company to users watchlist	User
DELETE	/api/v1/watchlist/ {company_id}	Deletes a company from users watchlist	User

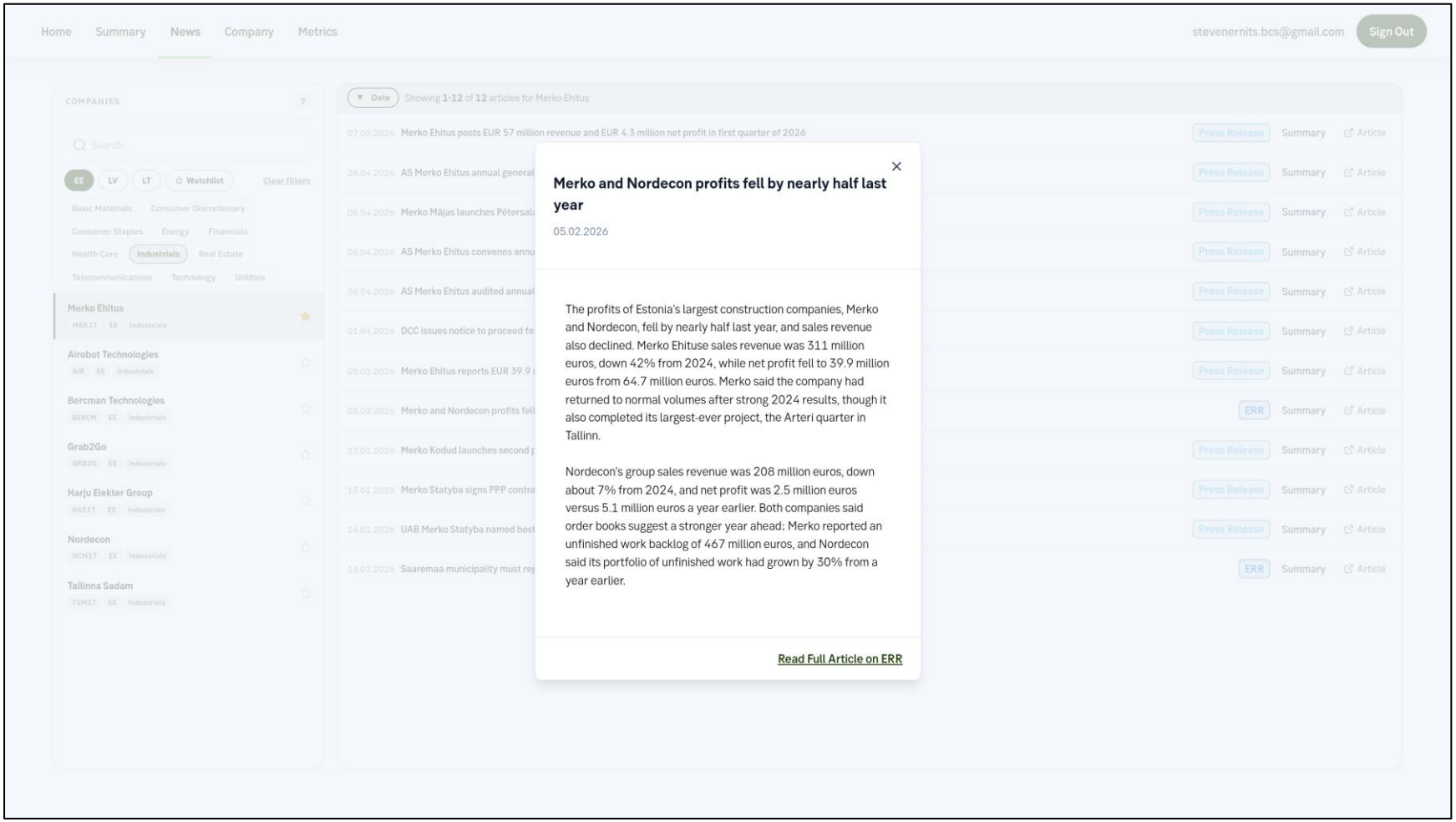
Appendix 5 – Admin accessible endpoints

Method	Endpoint	Description	Access
POST	/api/v1/login/access-token	Log in for admin	Admin
POST	/api/v1/summaries/{company_id}	Generates summary by company id	Admin
GET	/api/v1/summaries/prompt-data	Gets formatted text block that would be sent to the LLM for summary generation	Admin
POST	/api/v1/article-summaries/{article_id}	Generates a summary of an article by article id	Admin
POST	/api/v1/scrapers/…	Different scraping testing endpoints for used sources (err, dblv, lrt)	Admin
GET	/api/v1/metrics/token-usage	Fetches token usage for all LLM processes, filterable by date range	Admin
POST	/api/v1/pipelines/data-collection/{company_id}	Runs the data collection pipeline for the specified company	Rootadmin
POST	/api/v1/pipelines/article-summary/{company_id}	Runs the article summarisation pipeline for the specified company	Rootadmin
POST	/api/v1/pipelines/financial-report/{company_id}	Runs the report summarisation pipeline for the specified company	Rootadmin
POST	/api/v1/pipelines/summary/{company_id}	Runs the summary generation pipeline for the specified company	Rootadmin

Appendix 6 – Summary Page Financial Report



Appendix 7 – Article Summary Modal



Appendix 8 – Contribution and self-analysis of Rene Väljaots

My primary fields of work for this project were the backend architecture, data infrastructure, LLM integration and a significant portion of frontend development. I was responsible for designing and building the automation layer of the application, refactoring to hexagonal architecture and adding the task queue infrastructure. I was also responsible for the deployment of the application.

Contribution to the project

My main contributions to the project are listed as follows:

- Refactored the entire backend to the hexagonal architecture, implementing ports and adapters and integrating each into the services.
- Redesigned the database schema to support the new pipeline architecture and other features added.
- Built four automated pipelines and daily orchestration tasks.
- Implemented the structured output feature for financial reports, article summaries and company summaries.
- Built backend API endpoints and added many sorting and filtering options.
- Set up and implemented the queue and worker logic and architecture.
- Developed some frontend features: added the stock price Pinia store with caching, the news page pagination, the batch article summary fetching, the company view and the company chart logic.
- Styled some pages and components, made mobile UI improvements.
- Deployed the project to production and resolved production issues.
- Performed the continued LLM quality assessment using structured outputs.
- Proofread the thesis and wrote large parts of sections 4 and 5.
- Created architectural system diagrams in D2 and Mermaid.
- Conducted 3 feedback interviews.

Self-analysis

The further development of this project turned out to be significantly larger in scope than I had anticipated. What I initially expected to be a small refactor ended up being closer to building an entirely new product. The most challenging aspect was the use of abstractions in Python, which I had not done before, but which were essential to making the project extensible. I am satisfied with the end result, and it was encouraging to hear from the feedback interviews that users found the platform genuinely useful. The features I am most proud of are the structured LLM outputs and the financial report extraction, which opened up considerably more possibilities for future development than we had before. I am also pleased with how the company view page came together, combining a stock price chart, a small company overview and article summaries into a single view. It was a view I personally wanted to see on the page and I think it worked well. The created architecture supports adding additional logic and makes the life of future developers easier. The system should be scalable and allows adding more countries in the future. This project was successful in my eyes.

Total time I contributed to the project's development and thesis writing: 510 hours.

Appendix 9 – Contribution and self-analysis of Steven Ernits

My primary fields of work for this project were related to the frontend, authentication, and helping to plan the architecture of the application, as well as creating the system diagrams and writing the theoretical analysis. I also structured and wrote much of the thesis paper.

Contribution to the project

My main contributions to the project are listed as follows:

- Helping to plan the restructuring of the backend architecture to use the hexagonal pattern.
- Redesigning the frontend UI and authentication system, adding Google OAuth as well as sealed and encrypted session tokens, working on screen size responsiveness.
- Optimising endpoint calls on the frontend as the backend API evolved.
- Implementing the company watchlist feature, including the backend service, API endpoints, database model, and frontend integration.
- Implementing the Celery task for fetching company stock prices daily.
- Rewriting the metrics service to query token usage from PostgreSQL instead of Redis.
- Adding the financial report summary tab to the company view.
- Creating the system's entity relation and sequence diagrams.
- Performing initial quality assessment on different LLM models.
- Structuring and formatting the thesis paper.
- Collecting material and writing the project's theoretical background as well as contributing to the other chapters.
- Designing the user validation and feedback structure, conducting 3 feedback interviews.

Self-analysis

Much of the tech stack we used was completely new to me, the only familiar parts were PostgreSQL and Nuxt, so there was a lot to learn from this project. The restructuring of the initial solution brought its own hurdles, most notably the switch to hexagonal architecture. I consider the architectural change and implementation of the automatic pipelines the most valuable lessons from this project, as I finally understood the importance of clear and well-structured architecture, and I learned how complicated processes containing multiple steps can be automated. I also developed my frontend design and code skills, which I had been postponing for a while.

I am very pleased with the end result of the project and proud of our work as a team. We discussed every new idea together and often iterated over them countless times, every disagreement we had always resulted in an even better outcome than initially expected. I feel that we have completed a strong project that was well thought out through the entire development process, which was only possible with clear and frequent communication and desire to not just build something but also make it actually useful and scalable.

From this project I will take with me numerous new skills regarding programming, system design, LLMs, data scraping, UI design and working as a team on a software project.

Total time I contributed to the project’s development and thesis writing: 350 hours.