arc42 Template

January 2023

About arc42

arc42, the template for documentation of software and system architecture.

Template Version 8.2 EN. (based upon AsciiDoc version), January 2023

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

AI Stockbroker

Final Project Report

Cloud and Distributed Computing SoSe 25

Team Members:
Kevin Garrison
Ilef Kalboussi
Pardis Ebrahimi
Lukas Hauser

Supervisor: Prof. Roy Oberhauser
Date: July 2025

1 Introduction and Goals

This section was authored by K.G., I.K., P.E., and L.H.

The goal of this project is to build an AI-powered stock analysis platform, called **AI Stockbroker**, that provides real-time financial insights and investment recommendations *(Buy / Hold / Sell)* for listed companies. The system leverages real-time market data, SEC filings, and financial news sentiment to generate actionable analytics for users.

Main functional requirements include:

- Aggregation of financial data from multiple sources (APIs, SEC, news)
- AI-based analysis using large language models and forecasting tools
- Real-time recommendation engine
- Interactive visual dashboards and chatbot interface

1.1 Goals defined using the SMART criteria

1. **Specific:** Design and implement an AI-powered stock broker agent capable of analyzing SEC filings, forecasting short-term stock price movements, and generating actionable Buy, Sell, or Hold recommendations using a Large Language Model (LLM).
2. **Measurable** - Success will be evaluated based on:
   - Forecasting accuracy
   - Usability of recommendations via user feedback
   - Completion of defined project milestones
3. **Achievable:** Our team combines key technical skills—NLP, LLMs, time series forecasting, web scraping, and front-end development—along with robust tools like the EDGAR API and Nixtla's NeuralForecast library.
4. **Relevant:** The project addresses a real-world gap: retail investors lack access to timely, AI-enhanced analysis of financial documents. Our solution bridges data and decision-making.
5. **Time-bound:** The project spans 16 weeks (March 15 – July 4, 2025), with deliverables and sprint milestones every 2 weeks to ensure iterative progress and continuous validation.

1.2 Requirements Overview

1.2.1 Contents

This system is an AI-powered stock analysis platform that delivers intelligent investment recommendations (Buy / Hold / Sell) by aggregating and analyzing multiple financial data sources. It is designed to support investors in making informed decisions based on real-time market dynamics, official financial disclosures (SEC filings), and sentiment analysis of financial news.

1.2.2 Driving Forces

The platform addresses the growing need for:

- Automated, intelligent financial analysis powered by Large Language Models (LLMs)
- Real-time market monitoring and alerting capabilities
- User-friendly visualization of financial trends and predictions
- Aggregation of diverse data sources into a coherent recommendation engine
- Scalable and modular architecture for integrating new data sources or models over time

1.2.3 Motivation

From the perspective of the end users; primarily retail investors, financial analysts, or portfolio managers ; the system is built to:

- Automate the time-consuming task of reading and interpreting market data
- Provide consistent, explainable AI-driven recommendations
- Centralize fragmented data (filings, news, prices) into a single unified interface
- Improve both decision speed and decision quality in stock trading

1.2.4 Form

Table 1: Form (Functional Requirements Overview)

Feature ID	Feature Name	Description
FR1	Real-time market data aggregation	Collect and store up-to-date stock prices and volume using yfinance API and similar services.
FR2	SEC Filings Analysis	Fetch and parse quarterly and annual company filings (e.g., 10-Q, 10-K) to extract relevant financial insights.
FR3	Financial News Sentiment Processing	Analyze sentiment of financial news related to tracked companies using NLP models.
FR4	AI-based Recommendation Generation	Generate Buy/Hold/Sell decisions using LLMs based on multi-source data fusion.
FR5	Time-Series Forecasting	Predict short-term stock price trends using ML models and historical data.
FR6	Natural Language User Interface (Chatbot)	A smart chatbot interface for custom queries and analysis.
FR7	Interactive Data Visualizations	Present dynamic financial charts and metrics through React and Grafana dashboards.
FR8	Modular Microservice Architecture	Ensure modularity and scalability through decoupled microservices orchestrated via Docker Compose.
FR9	Centralized Logging & Monitoring	Aggregate and visualize runtime logs and container activity using Fluent Bit and Grafana.

2 Architecture Constraints

This section was authored by K.G., I.K., P.E., and L.H.

2.0.1 Constraints

The design and implementation of the system are subject to several technical, operational, and data-related constraints. These are outlined below:

- **Data Source Limitations:** External APIs such as SEC.gov and YFinance impose rate limits and availability constraints, affecting the freshness and completeness of fetched financial data.
- **Real-time Performance:** Due to the involvement of forecasting models and external data fetching, achieving low-latency responses for all queries is challenging. Caching strategies (e.g., Redis) and asynchronous data prefetching are used to mitigate this.
- **Scalability:** The system must handle concurrent requests from multiple users, especially in peak trading hours. The API-Gateway and backend services must scale horizontally to meet demand.
- **Storage Requirements:** Time-series data and vector embeddings require efficient and scalable storage solutions. InfluxDB and Qdrant have been chosen for their performance, but they require careful indexing and maintenance.

- **Security and Compliance:** Financial data must be handled securely. All components interacting with external APIs or storing sensitive data must comply with relevant data protection regulations (e.g., GDPR).
- **Model Interpretability:** Forecasting models and recommendation outputs must remain interpretable to build user trust. This can be challenging when overly complex or black-box models are used.
- **Monitoring Overhead:** Continuous logging and real-time visualization via Grafana can introduce performance overhead. Efficient log management is necessary to avoid system slowdowns.

2.0.2 Motivation

The financial decision-making process is increasingly reliant on real-time data, intelligent forecasting, and trustworthy information sources. However, investors and analysts often face challenges such as fragmented data sources, lack of automation in interpreting reports, and limited insight into stock trends.

This system aims to address these issues by integrating multiple data providers (e.g., SEC.gov, Yahoo Finance) into a unified, interactive platform. It offers automated recommendations, document retrieval, and time-series forecasting, all accessible via an intuitive user interface.

Additionally, the inclusion of explainable AI components such as a retrieval-augmented chatbot and vector-based search enhances transparency and user trust. Real-time monitoring and time-series visualization support analysts in making timely, data-driven investment decisions.

Ultimately, this system bridges the gap between raw financial data and actionable intelligence, streamlining the workflow from data acquisition to decision-making.

2.0.3 Form of the Work

This work is realized as a modular software system composed of multiple microservices and data processing components. The architecture is implemented using a service-oriented approach, enabling scalability, reusability, and maintainability (see Figure 1).

The core components include:

- A web-based user interface for stock selection, detail inspection, and recommendation viewing.
- An API gateway that orchestrates requests between the frontend and backend services.
- Backend services for data fetching, document retrieval, forecasting, and monitoring.
- Integration with external data sources (SEC.gov, YFinance) and internal databases (InfluxDB, Redis, Qdrant).
- Visualization tools such as Grafana for monitoring system performance and displaying time-series data.

The system is implemented and hosted on AWS using modern web technologies and deployed in a containerized environment to facilitate testing and scaling.

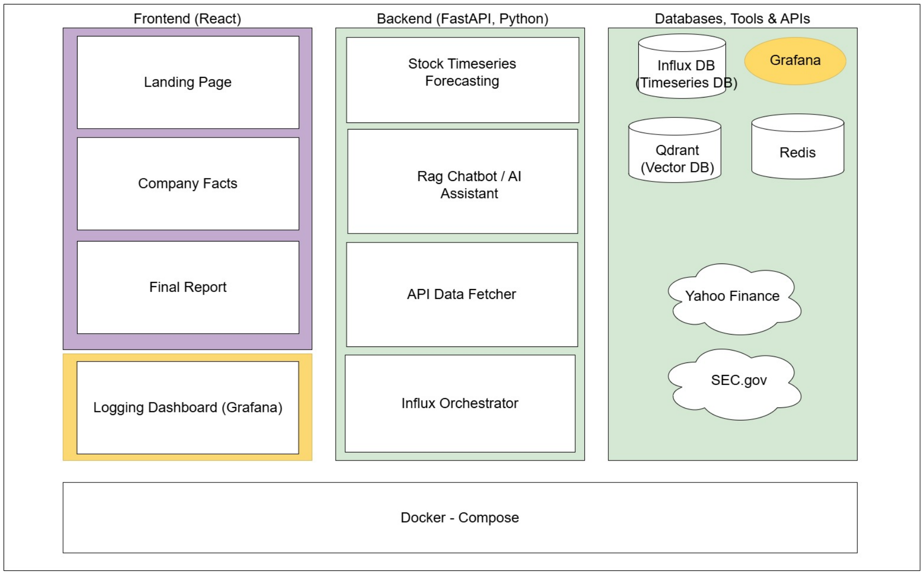


Figure 1: Modular system architecture showing frontend, backend services, and integrated databases/tools.

3 Context and Scope

This section was authored by I.K.and P.E.

3.1 Business Context

In today's high-velocity financial markets, accurate, timely, and actionable information is a key enabler of competitive investment strategies. Financial analysts, brokers, and institutional investors face mounting pressure to process massive volumes of heterogeneous data—ranging from market prices and earnings reports to regulatory disclosures and real-time news feeds.

Despite the abundance of available data, many firms encounter critical operational challenges:

- Data fragmentation across multiple disconnected platforms.
- Manual interpretation of complex financial and regulatory documents.
- Absence of real-time forecasts or AI-enhanced investment suggestions.
- Limited observability across data pipelines and decision workflows.

This project proposes an integrated and intelligent data platform that:

- Consolidates structured and unstructured financial data sources.
- Automates analysis of regulatory filings using NLP and LLMs.
- Delivers explainable, forecast-driven insights via a chatbot interface.
- Supports real-time monitoring of data ingestion, processing, and output.

The platform is strategically aligned to support:

- Faster, more accurate decision-making for institutional and retail investors.
- Cost reduction through automation of manual analysis tasks.
- Enhanced trust through transparent and explainable AI.
- Scalable infrastructure supporting long-term growth.

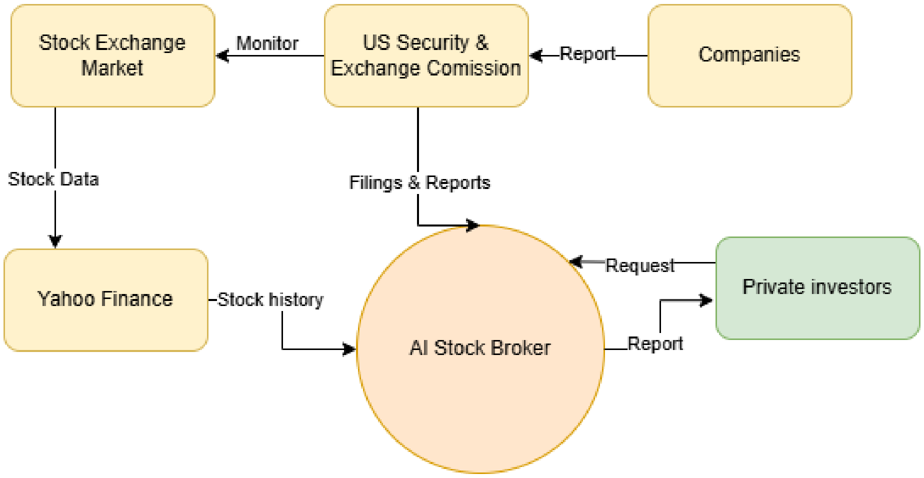


Figure 2: The AI Stock Broker platform within the broader financial data ecosystem.

Figure 2 illustrates the platform's position in the investment ecosystem, showing how it consolidates diverse data sources and provides intelligent recommendations to decision-makers.

3.2 Technical Context

This system operates in the domain of financial data processing and intelligent decision support, leveraging a service-oriented architecture composed of loosely coupled microservices. It integrates multiple external data sources and modern machine learning techniques to deliver real-time, data-driven recommendations to end users.

The architecture is cloud-native and containerized, facilitating scalability, modular development, and deployment through tools such as Docker. Backend services are developed using Python and Node.js, while the frontend is implemented as a single-page application (SPA) using a modern JavaScript framework (e.g., React).

The system communicates over REST APIs via a centralized API Gateway. Logging, time-series storage, and monitoring are handled using the ELK stack, InfluxDB, and Grafana, enabling real-time observability of both data and system performance.

3.3 Scope of Technical Implementation

The scope of the technical work includes:

- **Frontend Interface:** A modular web interface allowing users to select stocks, view company data, receive recommendations, and access financial documents.
- **API Gateway:** Acts as a centralized router and security layer for incoming frontend requests, managing interactions between services.
- **Backend Microservices:**
  - *API Fetcher:* Retrieves data from public APIs (e.g., Yahoo Finance, SEC.gov).
  - *Forecasting Service:* Generates time-series predictions for selected stocks or financial indicators.
  - *RAG Chatbot:* Provides document-based answers and explanations using vector embeddings and similarity search.
- **Datastores:**
  - *InfluxDB:* Stores time-series and monitoring data.
  - *Redis:* Caches frequently accessed documents and forecasts.
  - *Qdrant:* Serves as a vector database for document retrieval and semantic search.
- **Monitoring and Visualization:** Integration with Grafana for dashboarding and observability, based on logs and InfluxDB metrics.
- **Security and Data Integrity:** Basic input validation, HTTPS communication, and appropriate API access controls.

Out-of-scope elements include transaction execution, detailed portfolio management features, and deep financial risk modeling. These may be considered in future extensions of the system.

4 Solution Strategy

This section was authored by K.G. and L.H.

Contents

This section outlines the fundamental architectural decisions and strategies that shape the overall structure and behavior of the AI Stockbroker system. These decisions include the selected technologies, decomposition approach, quality-driven trade-offs, and organizational considerations that influence implementation and scalability.

Technology Decisions

The system is built upon a modern, modular technology stack to meet real-time performance and scalability needs:

- **Backend:** Python is used for data collection, processing, and AI-based analytics due to its mature ecosystem in finance and machine learning (e.g., yfinance, neuralforecast, OpenAI GPT-4o).
- **Frontend:** ReactJS provides a responsive, interactive user interface for data visualization and chatbot interaction.
- **Databases:**
  - InfluxDB is chosen for efficient handling of time-series market data.
  - Qdrant is used for vector similarity search, powering semantic queries and RAG-based chatbot interaction.
  - Redis enables caching and quick data retrieval to improve performance.
- **Visualization:** Grafana is used to create dynamic, real-time financial dashboards.
- **Infrastructure:** Docker and Docker Compose manage containerized microservices. Nginx serves as a reverse proxy to unify access points across services.

System Decomposition

The architecture follows a **microservices pattern** to ensure modularity, ease of maintenance, and independent scalability. The key components include:

- **API Gateway:** Manages request routing and centralizes service exposure.
- **Data Fetchers:** Collect real-time stock data, SEC filings, and news articles.
- **AI Engine:** Hosts large language models (GPT-4o) for NLP analysis and investment recommendations with retrieval augmented generation (RAG).
- **Forecasting Module:** Performs machine learning-based market predictions.
- **Frontend:** Displays data visualizations, user input forms via UI.

Quality-Driven Architectural Strategies

To meet non-functional requirements such as performance, reliability, and observability, the following strategies were applied:

- **Performance:** Redis is used for caching frequently accessed data; async processing is employed where applicable.
- **Scalability:** Each microservice can be independently scaled using Docker Compose.
- **Maintainability:** Clear service boundaries, RESTful APIs, and modular code promote easy updates and team collaboration.
- **Observability:** Fluent Bit collects logs from all containers, which are centralized and visualized in Grafana for monitoring and debugging.

Organizational Decisions

- **Development Process:** The team follows an agile approach with weekly sprints, using Trello for task tracking and Git for version control.
- **Team Distribution:** Responsibilities are divided among team members by component (e.g., data ingestion, AI module, frontend, infrastructure).
- **External Services:** Open-source tools and APIs are heavily leveraged (e.g., yfinance, SEC EDGAR API).

These strategic decisions form the backbone of the AI Stockbroker system. They enable modular development, ensure performance and scalability, and provide a solid foundation for AI-driven financial analysis.

5 Building Block View

This section was authored by K.G., I.K., P.E., and L.H.

This section describes the static structure of the system based on its decomposition into building blocks. It focuses on components that collaborate to process financial data, perform forecasts, deliver intelligent recommendations, and ensure system observability.

5.1 Whitebox Overall System (Level 1)

The overall system is structured into three main domains:

- **Frontend (User Interface)** – Enables users to interact with the system, view stock information, get AI-driven recommendations, and access relevant documents.
- **Backend (Microservices)** – A set of independent services responsible for data fetching, forecasting, document processing, and data storage.
- **Monitoring/Logging (Developer Tools)** – Provides observability through logs and time-series dashboards using Grafana and InfluxDB.

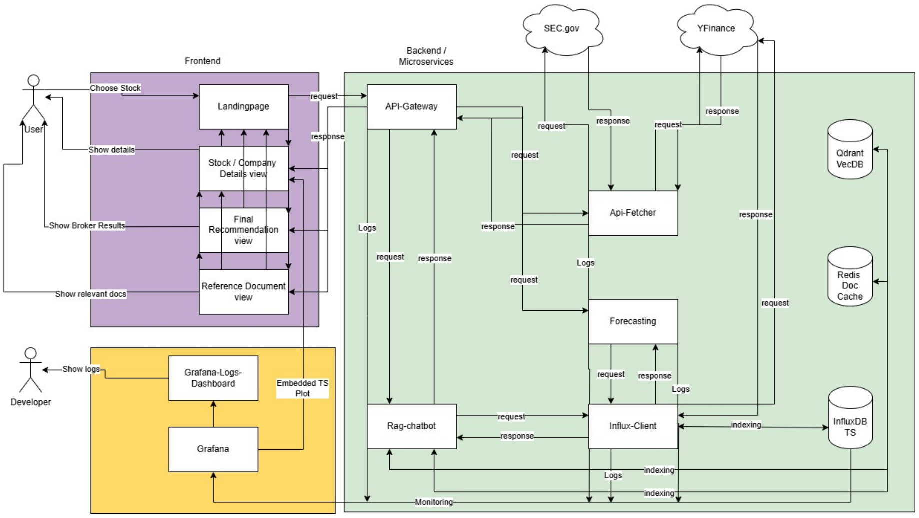


Figure 3: System Building Block Architecture illustrated by a Data Flow Diagram

Figure 3 provides an overview of the system's key modules and how they interact through data flows.

5.2 Level 2: Internal Structure of Main Building Blocks

Frontend (Purple Section)

- **Landing Page** – Entry point for users to choose a stock.
- **Stock/Company Details View** – Displays market data for selected stocks.
- **Final Recommendation View** – Shows the AI's stock recommendation output.
- **Reference Document View** – Displays relevant regulatory or financial documents.

API Gateway

Acts as a router and security layer for frontend requests. It forwards calls to respective backend services and handles logging.

Backend Services (Green Section)

- **API-Fetcher** – Collects real-time stock data from `Yahoo Finance` and official financial documents from `SEC.gov`.
- **Forecasting Service** – Generates time-series stock predictions and performance estimates.
- **Influx-Client** – Interfaces with `InfluxDB` to write and retrieve monitoring/forecasting time-series data.
- **RAG-Chatbot** – Processes document-based queries using vector search from `Qdrant` and document cache in `Redis`.

Datastores

- **Qdrant VectorDB** – Used for semantic search and similarity-based document retrieval.
- **Redis Cache** – Caches documents and recommendation results.
- **InfluxDB** – Stores time-series data from logs and predictions.

Monitoring (Orange Section)

- **Grafana Logs Dashboard** – Displays logs and metrics for developers.
- **Grafana** – Provides dashboards for real-time system monitoring using data from `InfluxDB`.

Actors

- **User** – Uses the UI to interact with the stock broker system.
- **Developer** – Monitors system health and performance via logs and dashboards.

5.3 Motivation and Justification

The architecture is designed using a modular, microservice-based approach to maximize scalability, maintainability, and real-time responsiveness. By clearly separating frontend, backend, and monitoring concerns, the system remains loosely coupled and easily extensible.

5.4 Interfaces and Communication

All services communicate via RESTful APIs. Logs and metrics flow from services to the InfluxDB time-series database and are visualized in Grafana for developer monitoring.

6 Runtime View

This section was authored by K.G., I.K., P.E., and L.H.

Contents

The runtime view describes the dynamic behavior and interactions between the building blocks of the system, based on typical scenarios. The selected runtime scenarios focus on:

- Key functional use cases executed by end-users.
- Integration with external systems such as financial APIs.
- Operational monitoring and developer workflows.

Motivation

Understanding the runtime behavior of the system is essential for developers, architects, and stakeholders who want to grasp how the system components interact during execution. The following scenarios represent the most architecturally relevant workflows of the system.

Form

Each scenario is described as a numbered sequence of events in natural language. Corresponding runtime diagrams can be found below and should be interpreted as logical flows (not strict timing-based interactions).

6.1 Runtime Scenario 1: Stock Recommendation Flow

Description: A user selects a stock to analyze. The system fetches data from external sources, forecasts future stock values, and provides relevant supporting documents.

- User selects a stock on the frontend interface.
- The frontend sends a request to the **API Gateway**.
- The gateway forwards the request to the **API-Fetcher**, which retrieves financial data from **Yahoo Finance** and **SEC.gov**.
- After receiving the data, the gateway sends a request to the **Forecasting Service**.
- The Forecasting service generates time-series predictions and logs them via the **Influx-Client**.
- The combined result — stock details, forecasts, and contextual documents — is returned to the frontend and shown to the user.

Notable Aspects:

- This flow illustrates the core functionality of the platform.
- Highlights integration of multiple backend services.

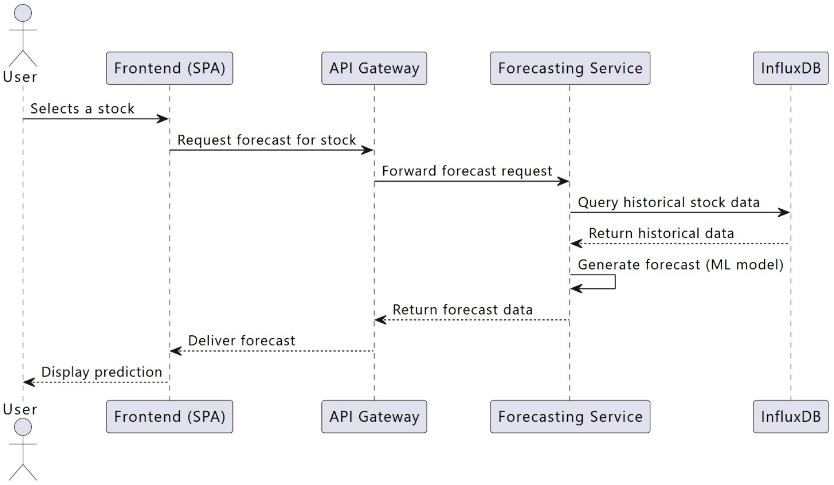


Figure 4: Stock Recommendation Flow

6.2 Runtime Scenario 2: Monitoring via Grafana

Description: A developer wants to inspect system performance and logs through Grafana dashboards.

- A backend service (e.g., **Forecasting**) generates logs during runtime.
- These logs are forwarded to the **Influx-Client**, which indexes and stores them in **InfluxDB**.
- The **Grafana** dashboard is configured to visualize this data.
- The developer accesses the Grafana UI and inspects time-series metrics or embedded logs.

Notable Aspects:

- This scenario shows how observability is achieved.
- Demonstrates backend-to-database-to-UI flow.
- Supports quick debugging and performance insights.

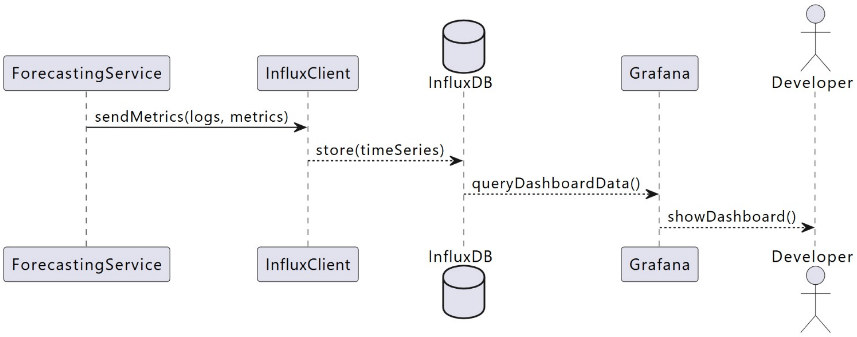


Figure 5: Monitoring via Grafana

6.3 Runtime Scenario 3: External API Fetching and Caching

Description: The system periodically fetches financial data from external sources and updates the cache.

- The **API Gateway** triggers the **API-Fetcher**.
- The API-Fetcher sends data requests to external APIs: **Yahoo Finance** and **SEC.gov**.
- Responses are parsed and transformed into usable formats.
- The resulting data is stored in the **Redis** cache for quick future access.
- Optionally, logs from the operation are sent to **InfluxDB**.

Notable Aspects:

- Demonstrates external system communication.
- Shows how caching improves system responsiveness.
- Supports real-time data availability for user queries.

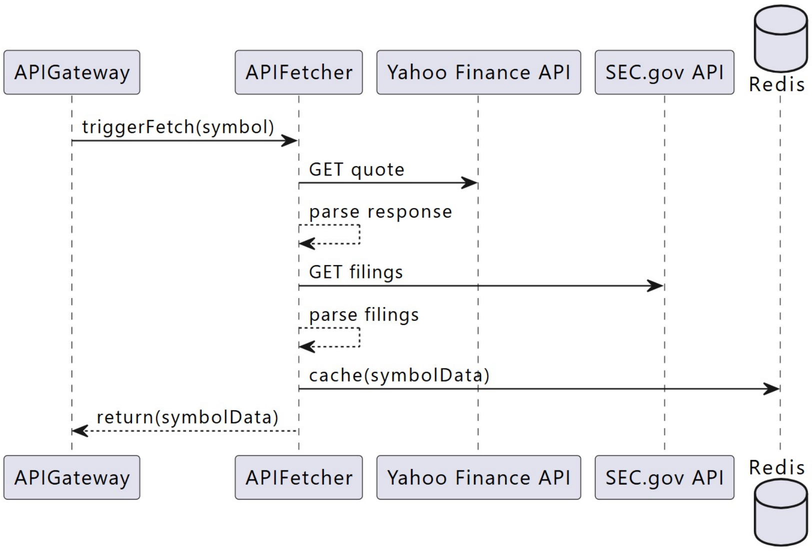
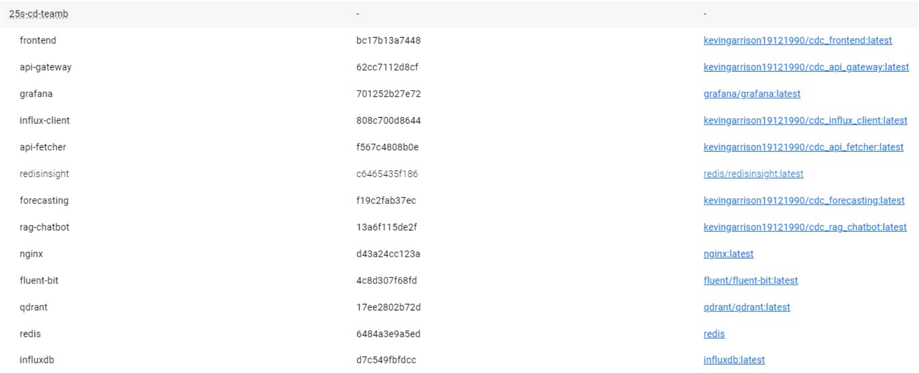


Figure 6: External API Fetching and Caching

7 Deployment View

This section was authored by K.G., I.K., P.E., and L.H.

7.1 Infrastructure Level 1: Docker Compose Overview



25s-cd-teamb	-	-
frontend	bc17b13a7448	kevingarrison19121990/cdc_frontend:latest
api-gateway	62cc7112d8cf	kevingarrison19121990/cdc_api_gateway:latest
grafana	701252b27e72	grafana/grafana:latest
influx-client	808c700d8644	kevingarrison19121990/cdc_influx_client:latest
api-fetcher	f567c4808b0e	kevingarrison19121990/cdc_api_fetcher:latest
redisinsight	c6465435f186	redis/redisinsight:latest
forecasting	f19c2fab37ec	kevingarrison19121990/cdc_forecasting:latest
rag-chatbot	13a6f115de2f	kevingarrison19121990/cdc_rag_chatbot:latest
nginx	d43a24cc123a	nginx:latest
fluent-bit	4c8d307f68fd	fluent/fluent-bit:latest
qdrant	17ee2802b72d	qdrant/qdrant:latest
redis	6484a3e9a5ed	redis
influxdb	d7c549fbfdcc	influxdb:latest

Figure 7: Overview of running Docker containers for AI Stockbroker

Figure 7 shows all the Docker containers that compose the AI Stockbroker system in our production environment. Each container encapsulates one microservice or infrastructure component, ensuring isolation and ease of deployment.

7.1.1 Brief Explanation

- **Frontend (React)** serves the user interface and static assets.
- **API Gateway** handles all incoming HTTP requests and routes them to the appropriate backend service.
- **Grafana** visualizes metrics and logs.
- **Fetcher Service** collects market data, SEC filings, and news in parallel.
- **InfluxDB**, **Qdrant**, and **Redis** provide time-series storage, vector search, and caching, respectively.
- **Forecasting** runs AI models to generate buy/hold/sell recommendations.
- **rag-chatbot** formats explanations via a Retrieval-Augmented Generation (RAG) pattern.
- **Nginx** reverse-proxies and terminates TLS for web traffic.
- **Api-fetcher** uses as a data fetcher for external data.
- **fluent-bit** collects the docker logs and metrics.

7.1.2 Motivation

We use Docker Compose to package each component into a standalone container. This approach provides:

- **Isolation:** Faults in one service do not directly affect others.
- **Scalability:** Containers can be replicated horizontally (e.g., multiple Fetcher replicas).
- **Consistency:** Identical environments across development, testing, and production.
- **Maintainability:** Easy updates and rollbacks by replacing container images.

7.2 Infrastructure Level 2: Container Details

In this subsection, we describe the internal structure and responsibilities of each Docker container.

7.2.1 API Gateway Container

- **Ports:** 8000
- **Responsibilities:**
  - Routing incoming requests to microservices.
  - Acts as an API aggregator.

7.2.2 API Fetcher Container

- **Ports:** 8001
- **External APIs:**
  - Yahoo Finance
  - SEC.gov
- **Data Flow:**
  - Fetch Report Data from SEC.
  - Fetch Time-series from Yahoo Finance

7.2.3 Forecasting Container

- **Ports:** 8003
- **Components:**
  - `RNN` (Neuralforecast)
  - `Nixtla`
- **Workflow:** Query data from InfluxDB, invoke AI/forecasting model, and persist recommendations back to the database.

7.2.4 RAG-Chatbot Container

- **Ports:** 8002
- **Functionality:**
  - Stores document as vectors.
  - Retrieval of important context vectors from Qdrant.
  - Generation of natural-language explanations via RAG.
  - Makes investment decisions via GPT-4o.

7.2.5 Frontend (React) Container

- **Ports:** 4000
- **Description:**
  - Serves a multipage Application.
  - Connects to API Gateway for data.

7.2.6 Grafana

- **Port:** 3000
- **Usage:**
  - Dashboards for monitoring metrics and logs.
  - Dashboards to visualize Time-series raw Data and Forecasting Data.

7.2.7 Database & Infrastructure Containers

InfluxDB

- **Port:** 8086
- **Data:**
  - Time-series storage for raw data.
  - Forecasting Data Storage.
  - Docker Logs and metrics storage.

Qdrant

- **Port:** 6333
- **Data:** Vector embeddings for RAG.

Redis

- **Port:** 6379
- **Role:** Caching layer for quick data reuse.

Nginx (Reverse Proxy)

- **Ports:** 80
- **Configuration:**
  - TLS termination.
  - Load balancing between frontend and API Gateway.

8 Cross-cutting Concepts

This section was authored by K.G., I.K., P.E., and L.H.

This section describes overarching rules, patterns, and principles that apply across multiple components of the AI Stockbroker system. They ensure conceptual integrity and consistency.

8.1 Domain Concepts

- **StockSymbol**
  - Attributes: `ticker`, `companyName`, `exchange`
  - Invariants: `ticker` must be unique; `exchange` code follows ISO standard.
- **Recommendation**
  - Attributes: `symbol`, `decision` (Buy/Hold/Sell), `timestamp`
  - Invariants: `timestamp` in UTC.

8.2 User Experience (UX) Concepts

- **Responsive Design:** Dashboards and chatbot UI must render correctly on desktop (≥1024 px) and mobile (<768 px).
- **Visual Consistency:** All pages in the project use the same color scheme, typography, and button styles, making navigation intuitive for users.
- **Error Feedback:** Display friendly messages on failures.

8.3 Security Concepts

- **Authentication:** All API calls must go through the API Gateway. No direct access to backend services is allowed from outside the network.
- **Authorization:** Role-Based Access Control is implemented at the API Gateway and/or backend services. Distinguish between "admin" and "standard" user permissions for sensitive operations (e.g., admin can manage users, standard can only view data).
- **Network Security:**
  - Only expose necessary ports (e.g., 80 for nginx, 3000 for Grafana, 8086 for InfluxDB).
  - Use Docker networks to isolate internal services.
  - Restrict inter-service communication to only what is necessary.
- **Transport Security:**
  - Use HTTPS for all external endpoints (nginx should terminate TLS).
  - Internal traffic can be encrypted if required by compliance.
- **Logging & Monitoring:**
  - Use Fluent Bit for centralized logging.
  - Monitor authentication/authorization failures and alert on suspicious activity.
- **Image Security:** Use official or trusted images.
- **Least Privilege Principle:**
  - Services run with only the permissions they need.
  - Avoid running containers as root unless absolutely necessary.

8.4 Architecture and Design Patterns

- **Microservices:** Each domain is an independent service.
- **API Gateway:** Single entry point for routing and aggregation.
- **Retrieval-Augmented Generation (RAG):** Chatbot augments LLM prompts with real context vectors from Qdrant.
- **Circuit Breaker:** Fetcher Service wraps external API calls to avoid cascading failures.

8.5 REST APIs List

8.5.1 API Gateway (Port 8000)

- **GET /api/data-ingest-yf-to-influx**
- **GET /api/companies-for-dropdown**
- **GET /api/screeners**
- **GET /api/filter-ticker-from-screener/**
- **GET /api/second-filter-market-cap/**
- **GET /api/companies-df**
- **GET /api/company-facts/{ticker}**
- **GET /api/company-news/{ticker}**
- **GET /api/stock-history/{ticker}**
- **GET /api/stock-broker-analysis/{ticker}**
- **GET /api/forecast/{ticker}**
- **GET /api/get-logo/{ticker}**
- **GET /api/reference-docs**

- **GET /api/download-reference-doc**
- **GET /api/download-broker-pdf/{ticker}**
- **GET /api/delete-cache**
- **GET /api/delete-forecasts**

8.5.2 API Fetcher (Port 8001)

- `GET /filter-market-cap/`
- `GET /companies`
- `GET /stock-history/{ticker}`
- `GET /yfinance-company-facts/{ticker}`
- `GET /company-context/{ticker}`
- `GET /company-news/{ticker}`
- `GET /get-logo/{ticker}`

8.5.3 Forecasting Service (Port 8003)

- `GET /forecast/{ticker}`

8.5.4 Influx Client (Port 8004)

- `GET /collect-stock-history-from-yfinance`
- `POST /write-forecast-to-influx/{ticker}`
- `GET /history/{ticker}`
- `GET /forecast/{ticker}`
- `GET /delete-forecasts`

8.5.5 RAG Chatbot (Port 8002)

- POST /relevant-sec-files/{ticker}
- GET /reference-docs-from-analysis
- GET /download-refdoc-redis
- GET /download-broker-pdf/{ticker}
- GET /delete-cached-docs

8.6 "Under-the-hood" Implementation Rules

- **Logging:** Structured JSON format with fields `serviceName`, `timestamp`, and `traceId`.
- **Error Handling:** Wrap all exceptions in a custom `ServiceException` containing an error code.

8.7 Development Concepts

- **Branching Strategy:** Git Flow with `feature/`, `develop`, and `main` branches.
- **Code Reviews:** All pull requests require one approving review.
- **Testing:** Manual testing was used during the sprints.

8.8 Operational Concepts

- **Container restart:** Docker Compose is configured to automatically restart containers if they become unhealthy.
- **Logging & Monitoring:** Fluent Bit collects and forwards structured logs from all containers. Grafana is used for dashboards and monitoring.
- **Configuration Management:** All services are configured via environment variables, managed securely outside of source code and injected at runtime through Docker Compose.
- **Secrets Management:** Sensitive credentials and API keys are injected as environment variables and never stored in source code.

9 Architecture Decisions

This section was authored by K.G., I.K., P.E., and L.H.

Document the most significant architectural choices, with context, alternatives, and rationale.

9.1 ADR-01: Choice of Vector Database

Status: Accepted
Context: Need a fast, scalable vector store to support RAG-based chatbot.
Decision: Use Qdrant.
Alternatives Considered:

- FAISS (requires manual shard management)
- Milvus (higher resource footprint)

Consequences:

+ Easy Docker deployment and Python client.
− Additional monitoring for index growth required.

Decision Summary Table

Table 2: Key Architecture Decisions

ID	Decision	Alternatives	Rationale
AD1	Vector DB: Qdrant	FAISS, Milvus	Ease of deployment, Python support, low ops overhead
AD2	Forecasting Model: RNN	Prophet, LSTM	Superior financial forecasting accuracy
AD3	Containerization: Docker	VMs, Bare Code	Lightweight, consistent environments, easy local development
AD4	Frontend Framework: React	Angular, Vue.js	Rich ecosystem and component reusability

10 Risks and Technical Debts

This section was authored by K.G., I.K., P.E., and L.H.

This section lists the main technical risks and accumulated technical debt in the AI Stockbroker system, ordered by priority. For each item, suggested mitigation or remediation measures are provided.

Table 3: Technical Risks and Debts

ID	Risk / Technical Debt	Priority	Mitigation / Remediation
R1	Dependency on free-tier APIs (yfinance, SEC.gov) may hit rate limits or change terms without notice.	High	Plan for paid API tiers or mirror critical data; implement backoff and caching strategies.
R2	Model interpretability: LLM–based recommendations can be not enough transparent to end users and regulators.	High	Integrate context files for audit
R3	Monolithic app creates a single point of failure.	High	Build microservices using Docker and insure a good Orchestration.
R4	Accumulated technical debt in ad-hoc error handling across services.	Low	Standardize on a common error-handling framework and custom exception hierarchy.

Note: **Priority levels** are assessed based on potential impact on availability, correctness, and maintainability.

10.1 Evaluation of RNN Model

This section was authored by P.E.

For the stock price forecasting task, we employed a Recurrent Neural Network (RNN) model from the Nixtla `NeuralForecast` library. RNNs are well-suited for modeling sequential data, making them particularly appropriate for financial time series. Their architecture enables them to learn dependencies across time steps, effectively capturing temporal dynamics such as trends, cycles, and seasonality.

To assess model performance, we selected the Mean Absolute Percentage Error (MAPE) as the evaluation metric. MAPE is widely adopted in financial forecasting due to its interpretability: it quantifies the average deviation between predicted and actual values as a percentage. For instance, a MAPE of 10% indicates that the model's predictions deviate from actual prices by an average of 10%.

The evaluation was conducted using historical stock data spanning from 2015 to 2025. To simulate a realistic forecasting scenario, the final five months of data (January to May 2025) were withheld during training. The model was then tasked with predicting this unseen period, and the forecasted values were compared against the actual stock prices for validation.

Figure 8 presents the forecast accuracy achieved across 20 stock tickers. The results demonstrate that the model performed consistently well, achieving an average accuracy of approximately 85%, indicating its effectiveness in capturing stock price behavior across different assets.

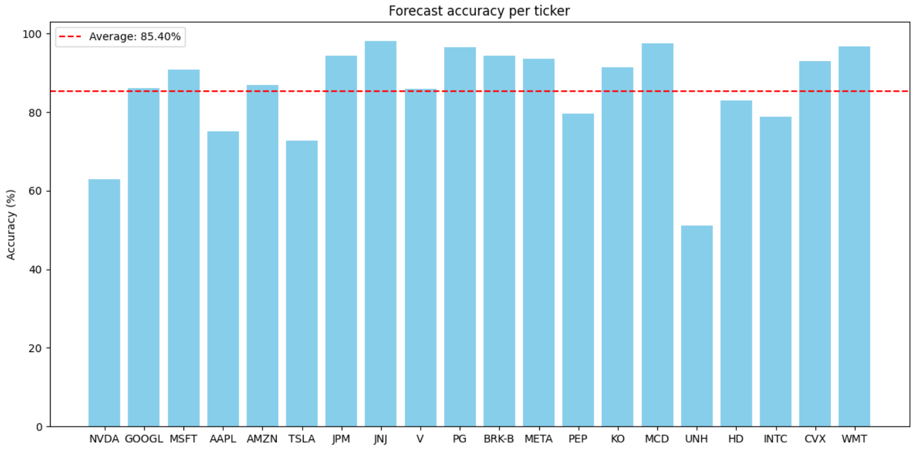


Figure 8: Forecast accuracy across 20 stock tickers.

A Appendix: Sprint Review Presentations

This appendix includes the PowerPoint slides used during the project reviews.

Aalen University

AI Stock Broker

CDC - Project

CDC, TeamB

Aalen University

Exploratory Sprint

CDC - Project

CDC, TeamB

SMART – Stock Broker Agent Project

Specific:

Design and implement an AI-powered stock broker agent capable of analyzing SEC filings, forecasting short-term stock price movements, and generating actionable Buy, Sell, or Hold recommendations using a Large Language Model (LLM).

Measurable (Work in Progress):

Success will be evaluated based on:

- Forecasting accuracy (e.g., MAPE or directional accuracy > X%)
- Usability of recommendations via user feedback
- Completion of defined project milestones

Achievable:

Our team combines key technical skills—NLP, LLMs, time series forecasting, web scraping, and front-end development—along with robust tools like the EDGAR API and Nixtla's NeuralForecast library.

SMART – Stock Broker Agent Project

RELEVANT

Relevant:

The project addresses a real-world gap: retail investors lack access to timely, AI-enhanced analysis of financial documents. Our solution bridges data and decision-making.

TIMED

Time-bound:

The project spans 16 weeks (March 15 – July 4, 2025), with deliverables and sprint milestones every 2 weeks to ensure iterative progress and continuous validation.

Components

Databases:

- InfluxDB
- Redis
- Qdrant VectorDB

Monitoring:

- Grafana
- Python logging
- GUI

Languages:

- Python 13
- Flux
- Html5
- Css
- Javascript

Frameworks:

- Frontend (React)
- Backend (FastAPI)

Timeseries Framework:

- Nixtla

APIs:

- SEC.gov
- Yahoo Finance

Deployment:

- Docker
- Docker-Compose
- AWS

LLM:

- OpenAI Gpt4o
- Deepseek R1

Versioning & Tracking:

- Bitbucket
- Trello

Communication:

- Whatsapp
- Teams

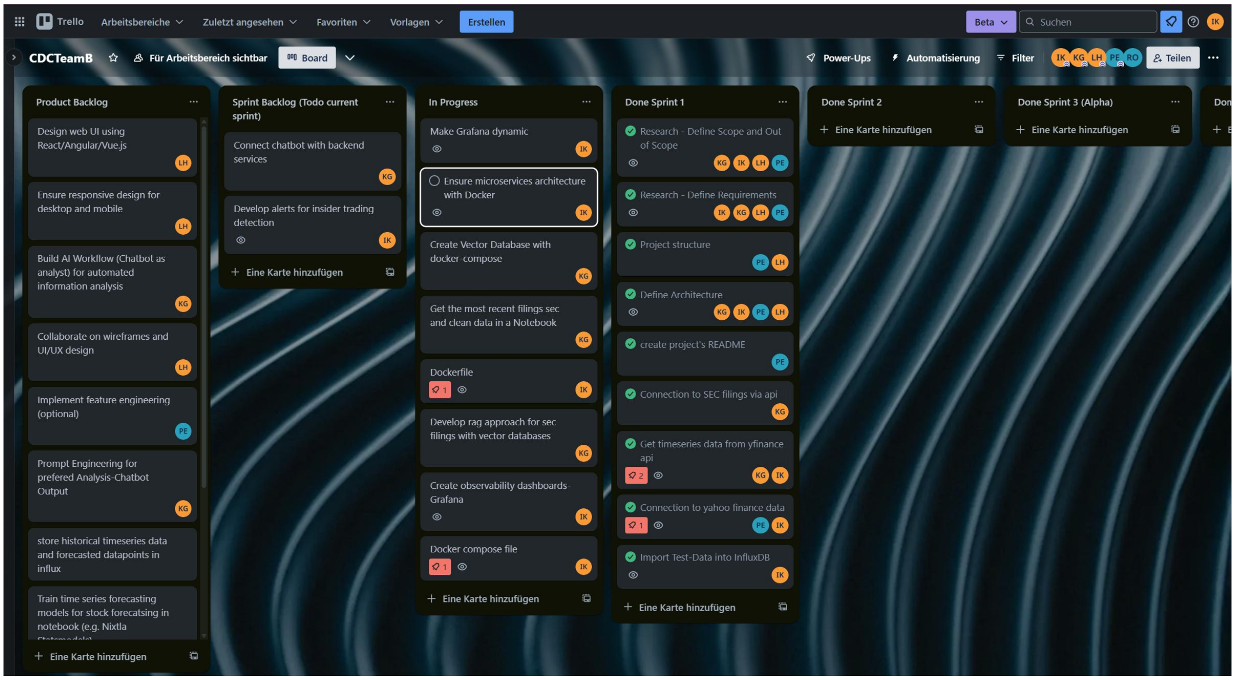
Risks:

- Data Quality
- Runtime
- Model Accuracy
- Hallucination LLM

Impediments:

- High Frequency Data
- Caching
- Merge Conflicts

Trello – Screenshot



Product Backlog

- Design web UI using React/Angular/Vue.js LH
- Ensure responsive design for desktop and mobile LH
- Build AI Workflow (Chatbot as analyst) for automated information analysis KG
- Collaborate on wireframes and UI/UX design LH
- Implement feature engineering (optional) PE
- Prompt Engineering for prefered Analysis-Chatbot Output KG
- store historical timeseries data and forecasted datapoints in influx
- Train time series forecasting models for stock forecatsing in notebook (e.g. Nixtla Statsmodels)

+ Eine Karte hinzufügen

Sprint Backlog (Todo current sprint)

- Connect chatbot with backend services KG
- Develop alerts for insider trading detection IK

+ Eine Karte hinzufügen

In Progress

- Make Grafana dynamic IK
- ○ Ensure microservices architecture with Docker IK
- Create Vector Database with docker-compose KG
- Get the most recent filings sec and clean data in a Notebook KG
- Dockerfile 1 IK
- Develop rag approach for sec filings with vector databases KG
- Create observability dashboards-Grafana IK
- Docker compose file 1 IK

+ Eine Karte hinzufügen

Done Sprint 1

- ✓ Research - Define Scope and Out of Scope KG IK LH PE
- ✓ Research - Define Requirements IK KG LH PE
- ✓ Project structure PE LH
- ✓ Define Architecture KG IK PE LH
- ✓ create project's README PE
- ✓ Connection to SEC filings via api KG
- ✓ Get timeseries data from yfinance api 2 KG IK
- ✓ Connection to yahoo finance data 1 PE IK
- ✓ Import Test-Data into InfluxDB IK

+ Eine Karte hinzufügen

Done Sprint 2

+ Eine Karte hinzufügen

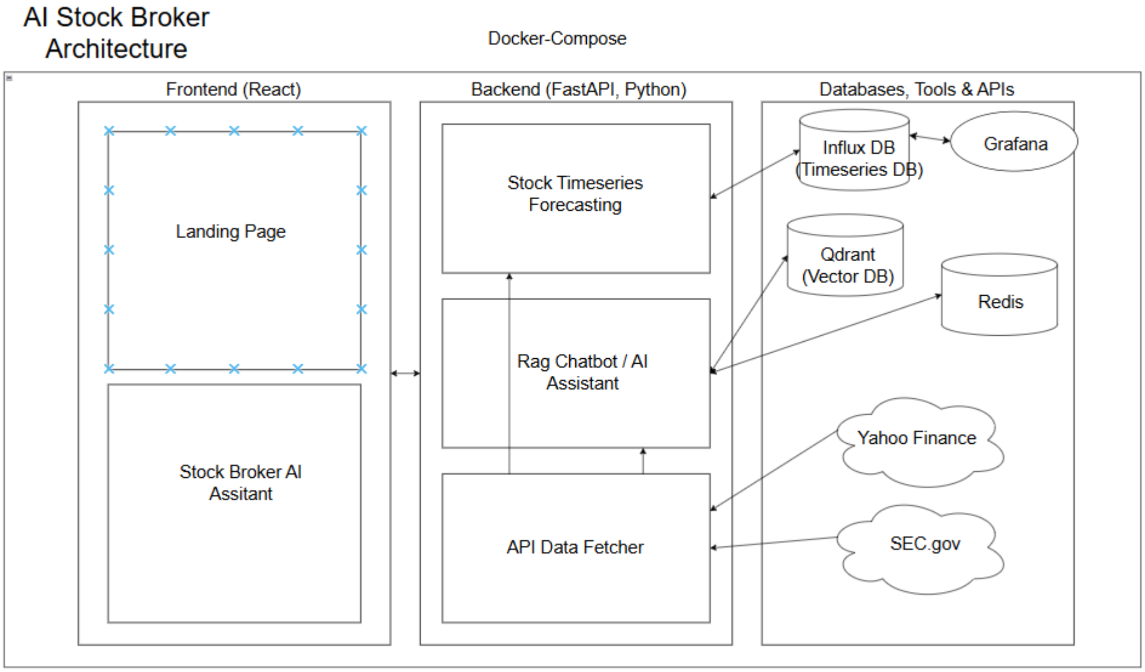
Done Sprint 3 (Alpha)

+ Eine Karte hinzufügen

Don

+ E

Project Architecture



Aalen University

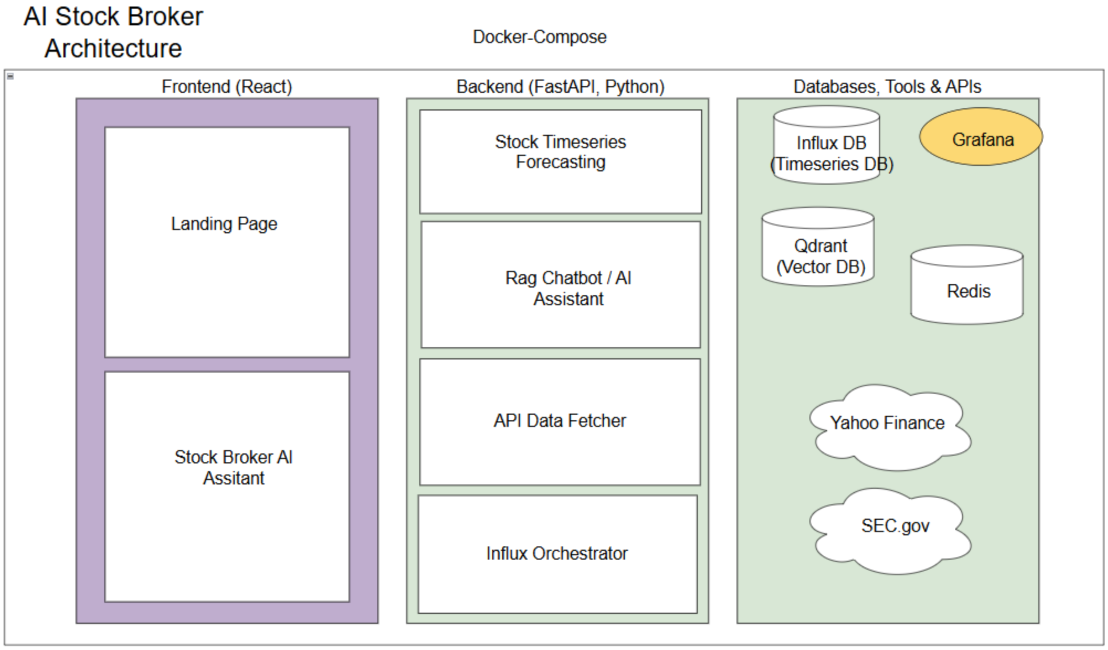
Architectual Spike Sprint

CDC - Project

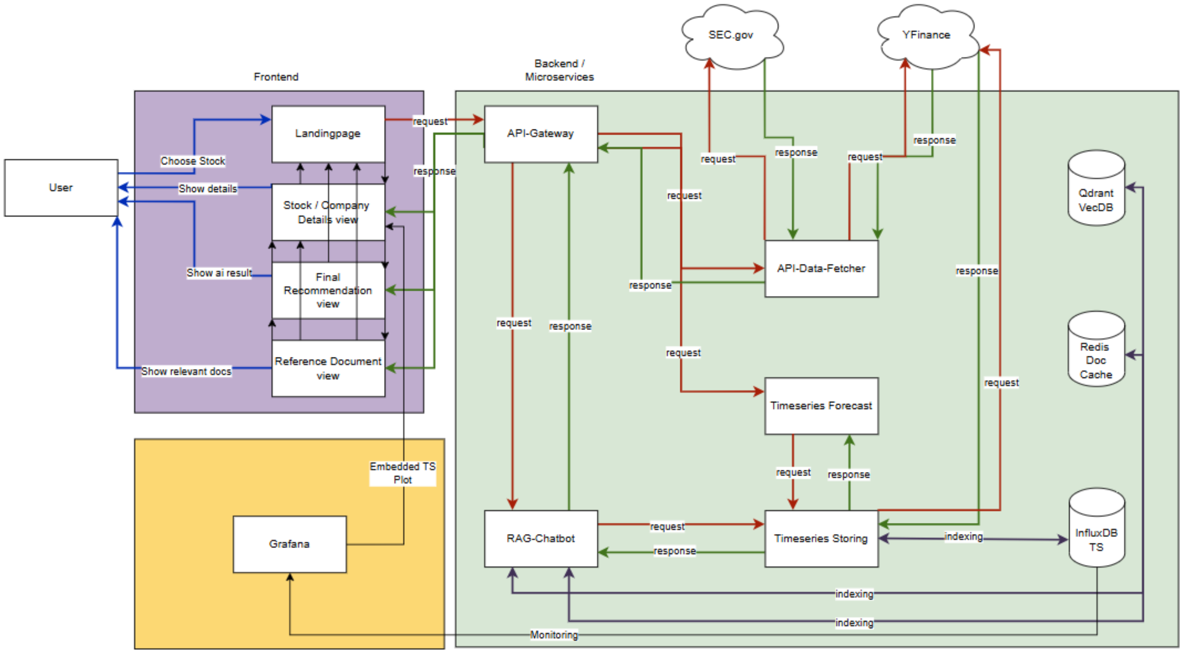
CDC, TeamB

Aalen University

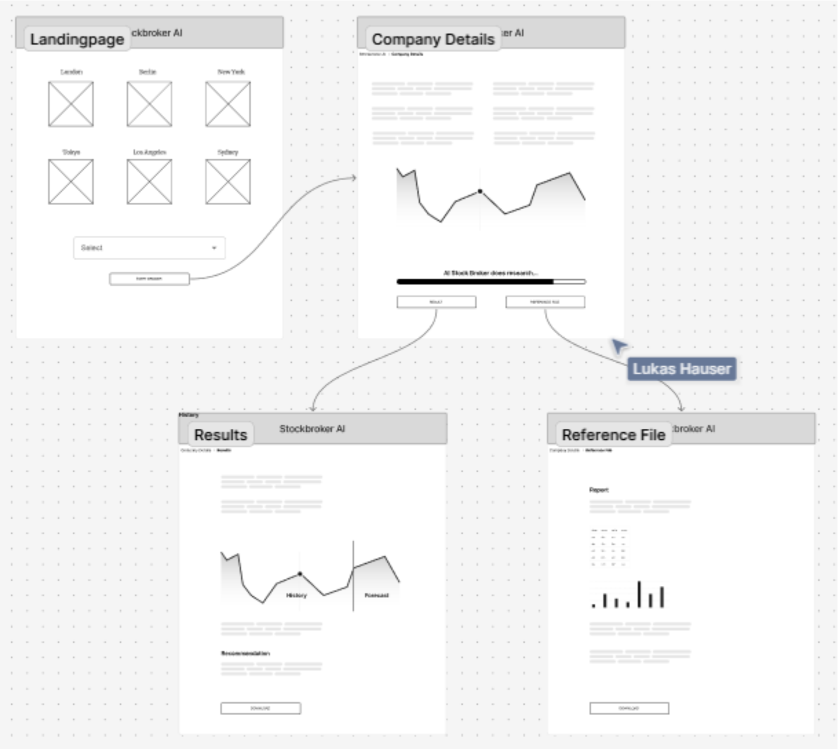
Information architecture diagram



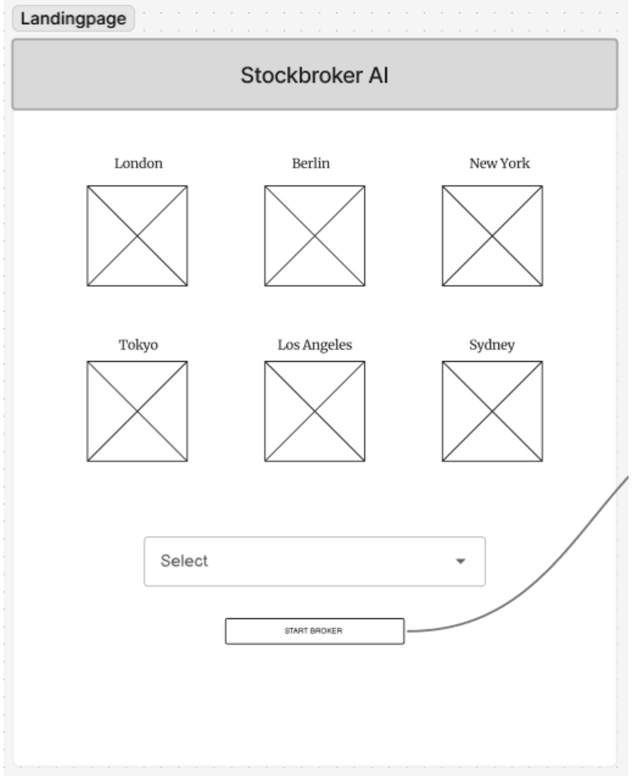
System architecture diagram



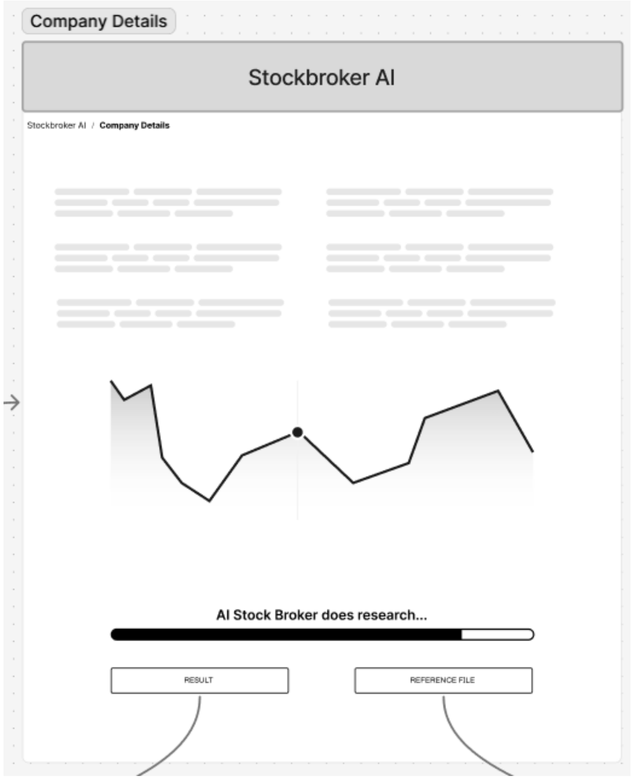
UI Wireframe



UI Wireframe

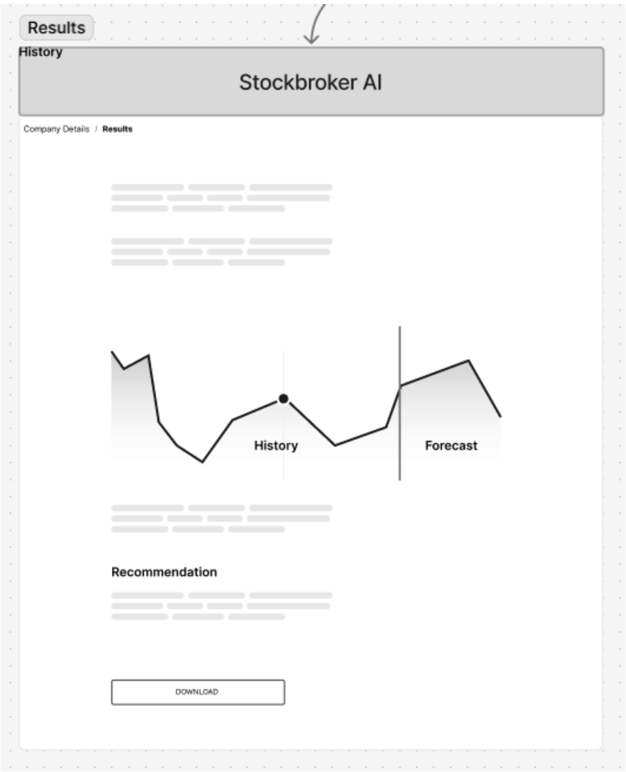


UI Wireframe

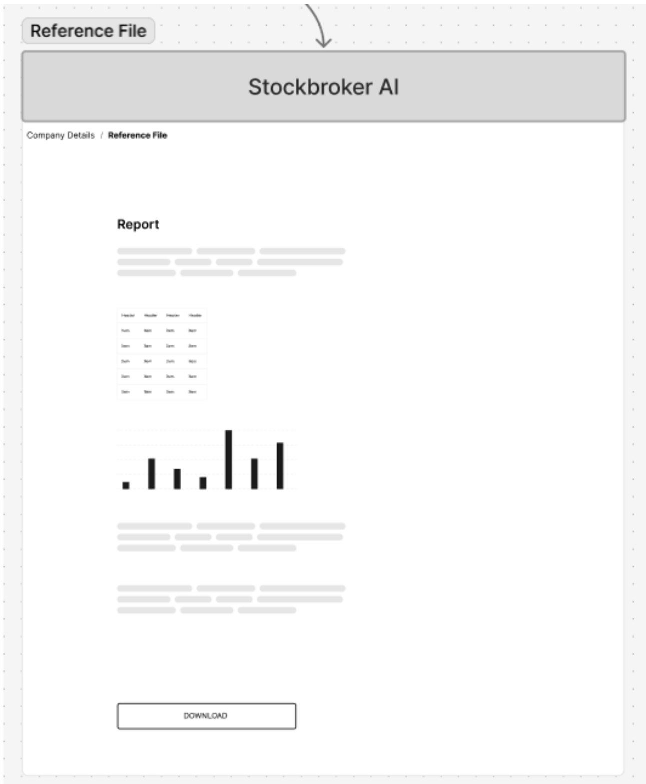


AI Stock Broker does research...

UI Wireframe

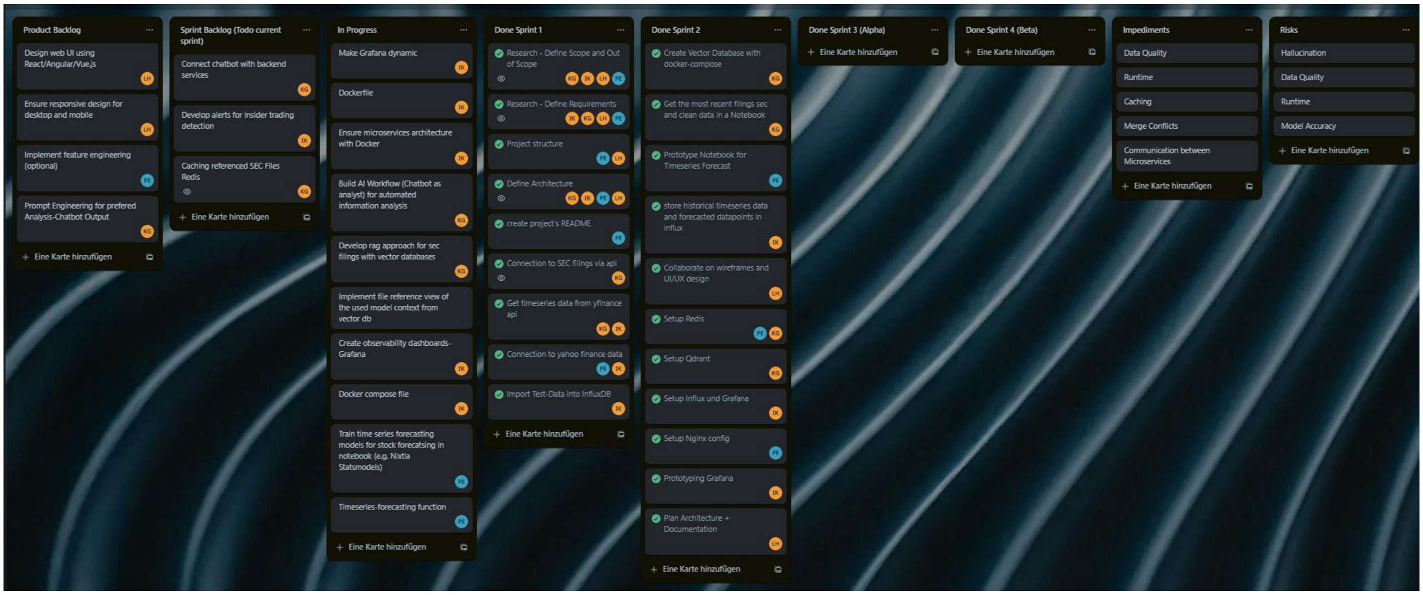


UI Wireframe



Header	Header	Header	Header
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Trello Screenshot



Product Backlog

- Design web UI using React/Angular/Vue.js LH
- Ensure responsive design for desktop and mobile LH
- Implement feature engineering (optional) PE
- Prompt Engineering for prefered Analysis-Chatbot Output KG

Sprint Backlog (Todo current sprint)

- Connect chatbot with backend services KG
- Develop alerts for insider trading detection IK
- Caching referenced SEC Files Redis KG

In Progress

- Make Grafana dynamic IK
- Dockerfile IK
- Ensure microservices architecture with Docker IK
- Build AI Workflow (Chatbot as analyst) for automated information analysis KG
- Develop rag approach for sec filings with vector databases KG
- Implement file reference view of the used model context from vector db
- Create observability dashboards-Grafana IK
- Docker compose file IK
- Train time series forecasting models for stock forecatsing in notebook (e.g. Nixtla Statsmodels) PE
- Timeseries-forecasting function PE

Done Sprint 1

- ✓ Research - Define Scope and Out of Scope KG IK LH PE
- ✓ Research - Define Requirements IK KG LH PE
- ✓ Project structure PE LH
- ✓ Define Architecture KG IK PE LH
- ✓ create project's README PE
- ✓ Connection to SEC filings via api KG
- ✓ Get timeseries data from yfinance api KG IK
- ✓ Connection to yahoo finance data PE IK
- ✓ Import Test-Data into InfluxDB IK

Done Sprint 2

- ✓ Create Vector Database with docker-compose KG
- ✓ Get the most recent filings sec and clean data in a Notebook KG
- ✓ Prototype Notebook for Timeseries Forecast PE
- ✓ store historical timeseries data and forecasted datapoints in influx IK
- ✓ Collaborate on wireframes and UI/UX design LH
- ✓ Setup Redis PE KG
- ✓ Setup Qdrant KG
- ✓ Setup Influx und Grafana IK
- ✓ Setup Nginx config PE
- ✓ Prototyping Grafana IK
- ✓ Plan Architecture + Documentation LH

Done Sprint 3 (Alpha)

Done Sprint 4 (Beta)

Impediments

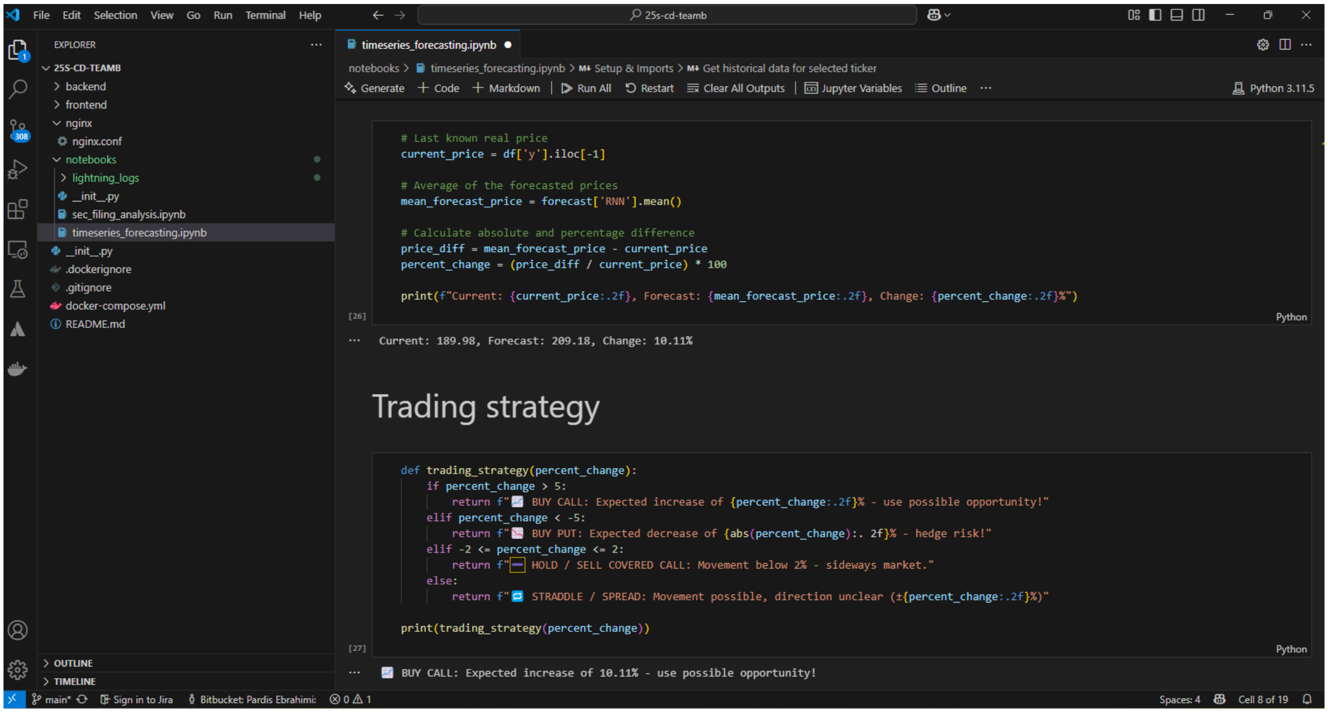
- Data Quality
- Runtime
- Caching
- Merge Conflicts
- Communication between Microservices

Risks

- Hallucination
- Data Quality
- Runtime
- Model Accuracy

Aalen University

Screenshots from Demo: Forecasting



```python
# Last known real price
current_price = df['y'].iloc[-1]

# Average of the forecasted prices
mean_forecast_price = forecast['RNN'].mean()

# Calculate absolute and percentage difference
price_diff = mean_forecast_price - current_price
percent_change = (price_diff / current_price) * 100

print(f"Current: {current_price:.2f}, Forecast: {mean_forecast_price:.2f}, Change: {percent_change:.2f}%")
```

[26] Python

```
Current: 189.98, Forecast: 209.18, Change: 10.11%
```

Trading strategy

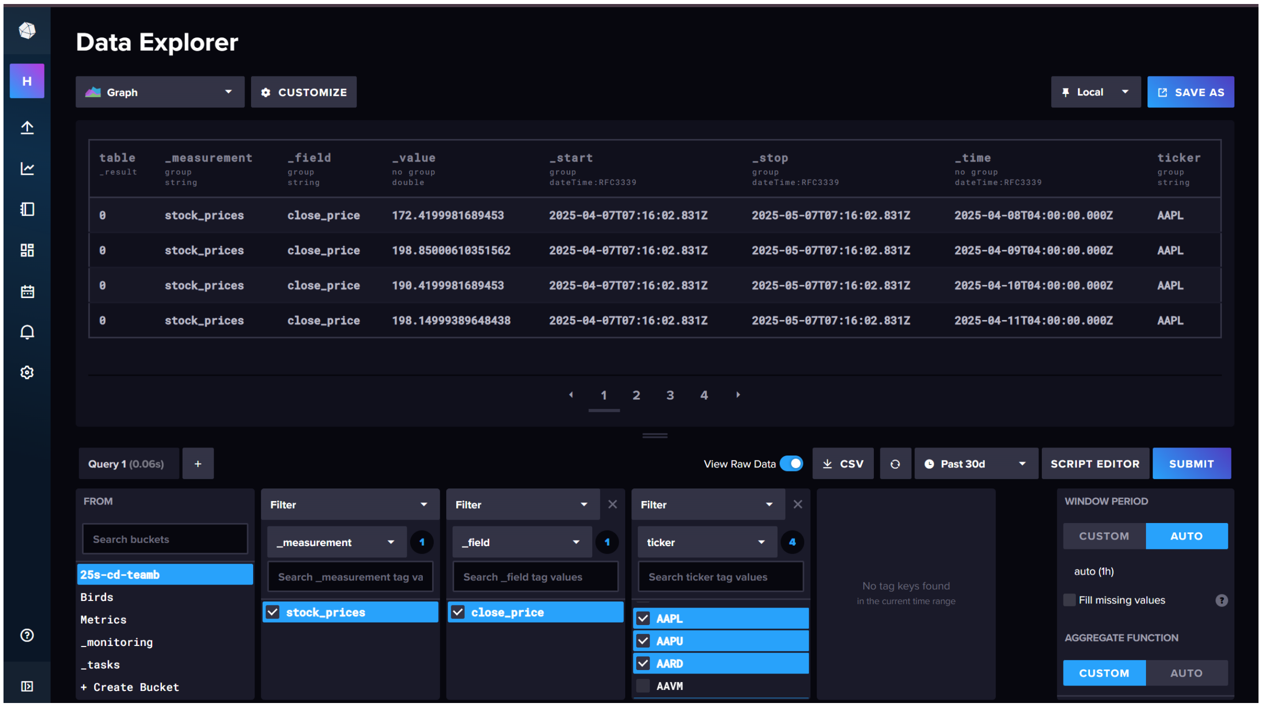
```python
def trading_strategy(percent_change):
    if percent_change > 5:
        return f"📈 BUY CALL: Expected increase of {percent_change:.2f}% - use possible opportunity!"
    elif percent_change < -5:
        return f"📉 BUY PUT: Expected decrease of {abs(percent_change):. 2f}% - hedge risk!"
    elif -2 <= percent_change <= 2:
        return f"➖ HOLD / SELL COVERED CALL: Movement below 2% - sideways market."
    else:
        return f"🔁 STRADDLE / SPREAD: Movement possible, direction unclear (±{percent_change:.2f}%)"

print(trading_strategy(percent_change))
```

[27] Python

```
📈 BUY CALL: Expected increase of 10.11% - use possible opportunity!
```

Screenshots from Demo: InfluxDB



Data Explorer

table _result	_measurement group string	_field group string	_value no group double	_start group dateTime:RFC3339	_stop group dateTime:RFC3339	_time no group dateTime:RFC3339	ticker group string
0	stock_prices	close_price	172.4199981689453	2025-04-07T07:16:02.831Z	2025-05-07T07:16:02.831Z	2025-04-08T04:00:00.000Z	AAPL
0	stock_prices	close_price	198.85000610351562	2025-04-07T07:16:02.831Z	2025-05-07T07:16:02.831Z	2025-04-09T04:00:00.000Z	AAPL
0	stock_prices	close_price	190.4199981689453	2025-04-07T07:16:02.831Z	2025-05-07T07:16:02.831Z	2025-04-10T04:00:00.000Z	AAPL
0	stock_prices	close_price	198.14999389648438	2025-04-07T07:16:02.831Z	2025-05-07T07:16:02.831Z	2025-04-11T04:00:00.000Z	AAPL

FROM

Search buckets

25s-cd-teamb
Birds
Metrics
_monitoring
_tasks
+ Create Bucket

Filter: _measurement 1

Search _measurement tag va

☑ stock_prices

Filter: _field 1

Search _field tag values

☑ close_price

Filter: ticker 4

Search ticker tag values

☑ AAPL
☑ AAPU
☑ AARD
□ AAVM

No tag keys found
in the current time range

WINDOW PERIOD

CUSTOM [✓ AUTO]

auto (1h)

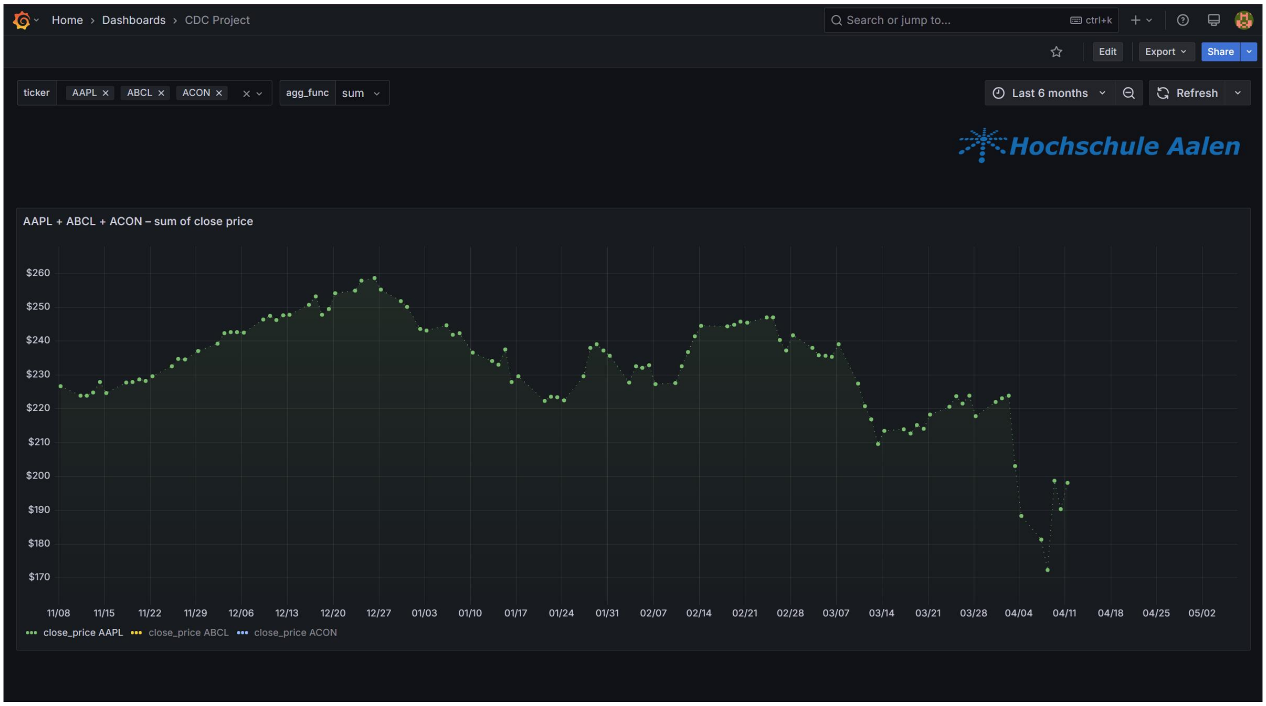
□ Fill missing values

AGGREGATE FUNCTION

[✓ CUSTOM] AUTO

Aalen University

Screenshots from Demo: Grafana



Home › Dashboards › CDC Project

ticker: AAPL, ABCL, ACON

agg_func: sum

Last 6 months

Hochschule Aalen

AAPL + ABCL + ACON – sum of close price

Text Section

- Completed (in last Sprint):
  - Create Vector Database with docker-compose
  - Get the most recent filings sec and clean data in a Notebook
  - Prototype Notebook for Timeseries Forecast
  - store historical timeseries data and forecasted datapoints in influx
  - Collaborate on wireframes and UI/UX design
  - Setup Redis
  - Setup Qdrant
  - Setup Influx und Grafana
  - Setup Nginx config
  - Prototyping Grafana
  - Plan Architecture + Documentation

- Planned (for next Sprint):
  - Dockerfile + Docker compose files
  - Ensure microservices architecture with Docker
  - Develop RAG approach for sec filings with vector databases
  - Implement file reference view of the used model context from vector db
  - Create observability dashboards- Grafana
  - Train time series forecasting models for stock forecatsing in notebook and as a function
  - Connect chatbot with backend services
  - Caching referenced SEC Files Redis

- Impediments & Risks:
  - Data Quality
  - Runtime
  - Caching
  - Merge conflicts
  - Communication between Microservices
  - Hallucination
  - Model Accuracy

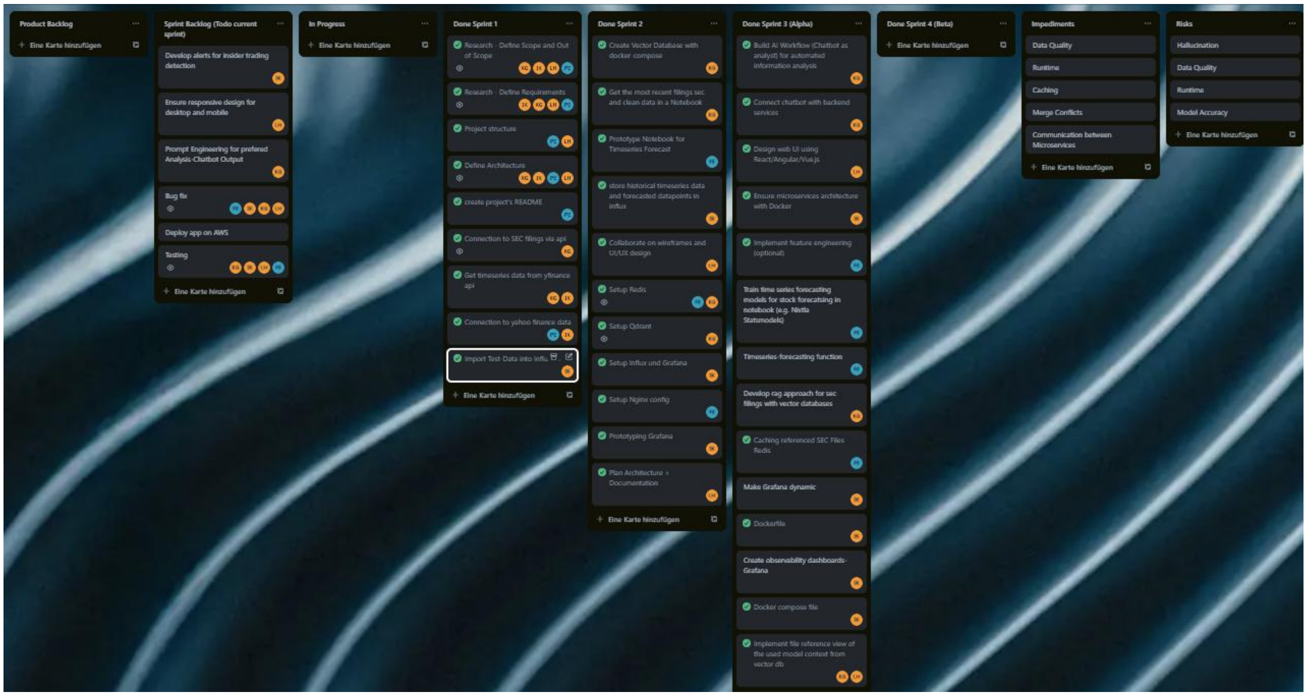
Aalen University

Alpha Sprint

CDC - Project

CDC, TeamB

Trello Screenshot



Product Backlog

- Eine Karte hinzufügen

Sprint Backlog (Todo current sprint)

- Develop alerts for insider trading detection
- Ensure responsive design for desktop and mobile
- Prompt Engineering for prefered Analysis-Chatbot Output
- Bug fix
- Deploy app on AWS
- Testing
- Eine Karte hinzufügen

In Progress

- Eine Karte hinzufügen

Done Sprint 1

- ✓ Research - Define Scope and Out of Scope
- ✓ Research - Define Requirements
- ✓ Project structure
- ✓ Define Architecture
- ✓ create project's README
- ✓ Connection to SEC filings via api
- ✓ Get timeseries data from yfinance api
- ✓ Connection to yahoo finance data
- ✓ Import Test-Data into Influ...
- Eine Karte hinzufügen

Done Sprint 2

- ✓ Create Vector Database with docker-compose
- ✓ Get the most recent filings sec and clean data in a Notebook
- ✓ Prototype Notebook for Timeseries Forecast
- ✓ store historical timeseries data and forecasted datapoints in influx
- ✓ Collaborate on wireframes and UI/UX design
- ✓ Setup Redis
- ✓ Setup Qdrant
- ✓ Setup Influx und Grafana
- ✓ Setup Nginx config
- ✓ Prototyping Grafana
- ✓ Plan Architecture + Documentation
- Eine Karte hinzufügen

Done Sprint 3 (Alpha)

- ✓ Build AI Workflow (Chatbot as analyst) for automated information analysis
- ✓ Connect chatbot with backend services
- ✓ Design web UI using React/Angular/Vue.js
- ✓ Ensure microservices architecture with Docker
- ✓ Implement feature engineering (optional)
- Train time series forecasting models for stock forecatsing in notebook (e.g. Nixtla Statsmodels)
- Timeseries-forecasting function
- Develop rag approach for sec filings with vector databases
- ✓ Caching referenced SEC Files Redis
- Make Grafana dynamic
- ✓ Dockerfile
- Create observability dashboards-Grafana
- ✓ Docker compose file
- ✓ Implement file reference view of the used model context from vector db

Done Sprint 4 (Beta)

- Eine Karte hinzufügen

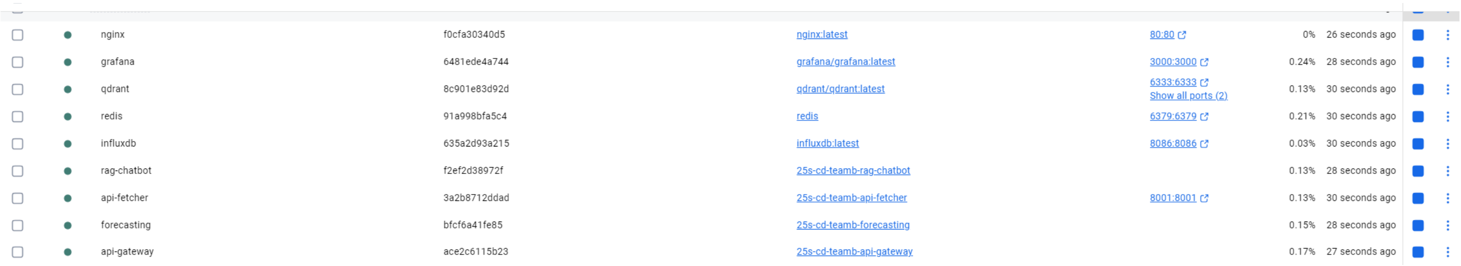
Impediments

- Data Quality
- Runtime
- Caching
- Merge Conflicts
- Communication between Microservices
- Eine Karte hinzufügen

Risks

- Hallucination
- Data Quality
- Runtime
- Model Accuracy
- Eine Karte hinzufügen

Docker



□	●	nginx	f0cfa30340d5	nginx:latest	80:80	0%	26 seconds ago	■	⋮
□	●	grafana	6481ede4a744	grafana/grafana:latest	3000:3000	0.24%	28 seconds ago	■	⋮
□	●	qdrant	8c901e83d92d	qdrant/qdrant:latest	6333:6333 Show all ports (2)	0.13%	30 seconds ago	■	⋮
□	●	redis	91a998bfa5c4	redis	6379:6379	0.21%	30 seconds ago	■	⋮
□	●	influxdb	635a2d93a215	influxdb:latest	8086:8086	0.03%	30 seconds ago	■	⋮
□	●	rag-chatbot	f2ef2d38972f	25s-cd-teamb-rag-chatbot		0.13%	28 seconds ago	■	⋮
□	●	api-fetcher	3a2b8712ddad	25s-cd-teamb-api-fetcher	8001:8001	0.13%	30 seconds ago	■	⋮
□	●	forecasting	bfcf6a41fe85	25s-cd-teamb-forecasting		0.15%	28 seconds ago	■	⋮
□	●	api-gateway	ace2c6115b23	25s-cd-teamb-api-gateway		0.17%	27 seconds ago	■	⋮

Risks/Impediments

- Bugs
- API availability
- Runtime
- Orchestration
- Poor UX (loading time)

Completed Alpha

- Microservices
- API Gateway MS
- Forecasting MS
- API MS
- Influx MS
- RAG Chatbot
- Grafana Dashboards
- Influx DB
- Redis Doc Caching
- Qdrant Vec DB
- Frontend Base

Todo Beta

- Bugs fixing
- Microservice Orchestration
- Runtime Optimization
- Forecast Optimization
- UX Features (News, Loading Bars, Doc Views, Stats ..)
- Deployment on AWS
- Switching RAG to CAG for better Runtime?!?

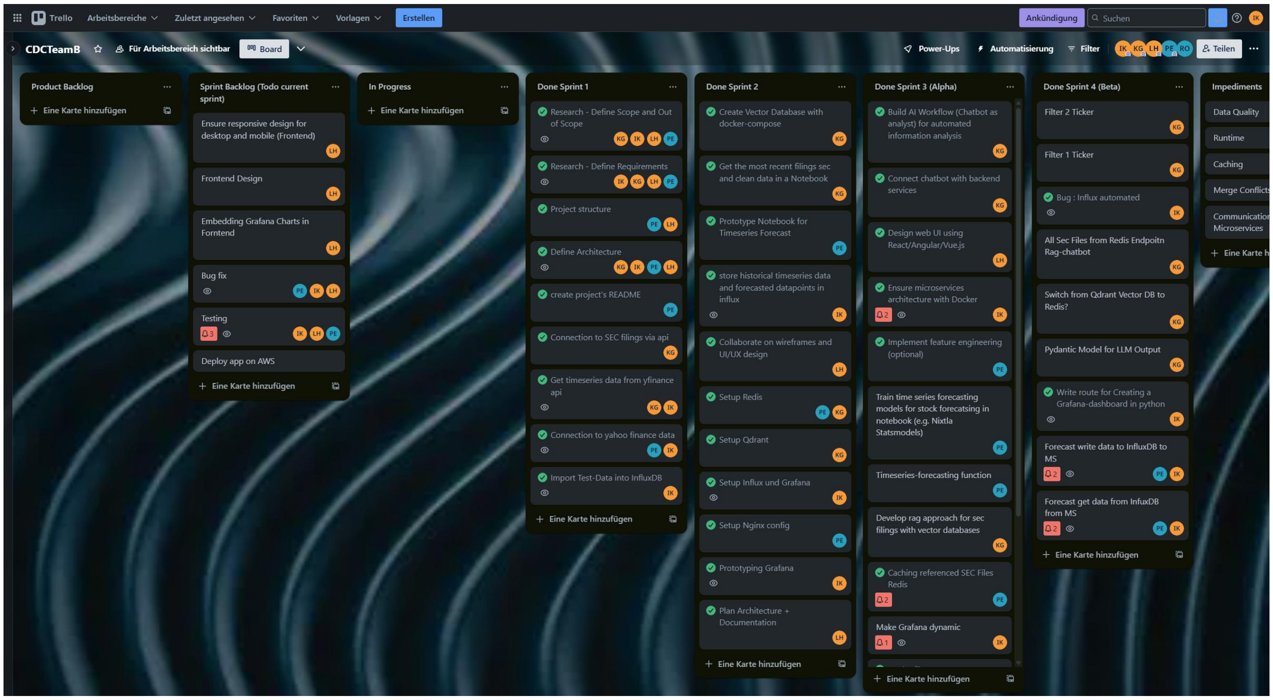
Aalen University

Beta Sprint

CDC - Project

CDC, TeamB

Trello Screenshot 1



Product Backlog

- Eine Karte hinzufügen

Sprint Backlog (Todo current sprint)

- Ensure responsive design for desktop and mobile (Frontend) — LH
- Frontend Design — LH
- Embedding Grafana Charts in Forntend — LH
- Bug fix — PE IK LH
- Testing — 3 — IK LH PE
- Deploy app on AWS
- Eine Karte hinzufügen

In Progress

- Eine Karte hinzufügen

Done Sprint 1

- ✓ Research - Define Scope and Out of Scope — KG IK LH PE
- ✓ Research - Define Requirements — IK KG LH PE
- ✓ Project structure — PE LH
- ✓ Define Architecture — KG IK PE LH
- ✓ create project's README — PE
- ✓ Connection to SEC filings via api — KG
- ✓ Get timeseries data from yfinance api — KG IK
- ✓ Connection to yahoo finance data — PE IK
- ✓ Import Test-Data into InfluxDB — IK
- Eine Karte hinzufügen

Done Sprint 2

- ✓ Create Vector Database with docker-compose — KG
- ✓ Get the most recent filings sec and clean data in a Notebook — KG
- ✓ Prototype Notebook for Timeseries Forecast — PE
- ✓ store historical timeseries data and forecasted datapoints in influx — IK
- ✓ Collaborate on wireframes and UI/UX design — LH
- ✓ Setup Redis — PE KG
- ✓ Setup Qdrant — KG
- ✓ Setup Influx und Grafana — IK
- ✓ Setup Nginx config — PE
- ✓ Prototyping Grafana — IK
- ✓ Plan Architecture + Documentation — LH
- Eine Karte hinzufügen

Done Sprint 3 (Alpha)

- ✓ Build AI Workflow (Chatbot as analyst) for automated information analysis — KG
- ✓ Connect chatbot with backend services — KG
- ✓ Design web UI using React/Angular/Vue.js — LH
- ✓ Ensure microservices architecture with Docker — 2 — IK
- ✓ Implement feature engineering (optional) — PE
- Train time series forecasting models for stock forecatsing in notebook (e.g. Nixtla Statsmodels) — PE
- Timeseries-forecasting function — PE
- Develop rag approach for sec filings with vector databases — KG
- ✓ Caching referenced SEC Files Redis — 2 — PE
- Make Grafana dynamic — 1 — IK
- Eine Karte hinzufügen

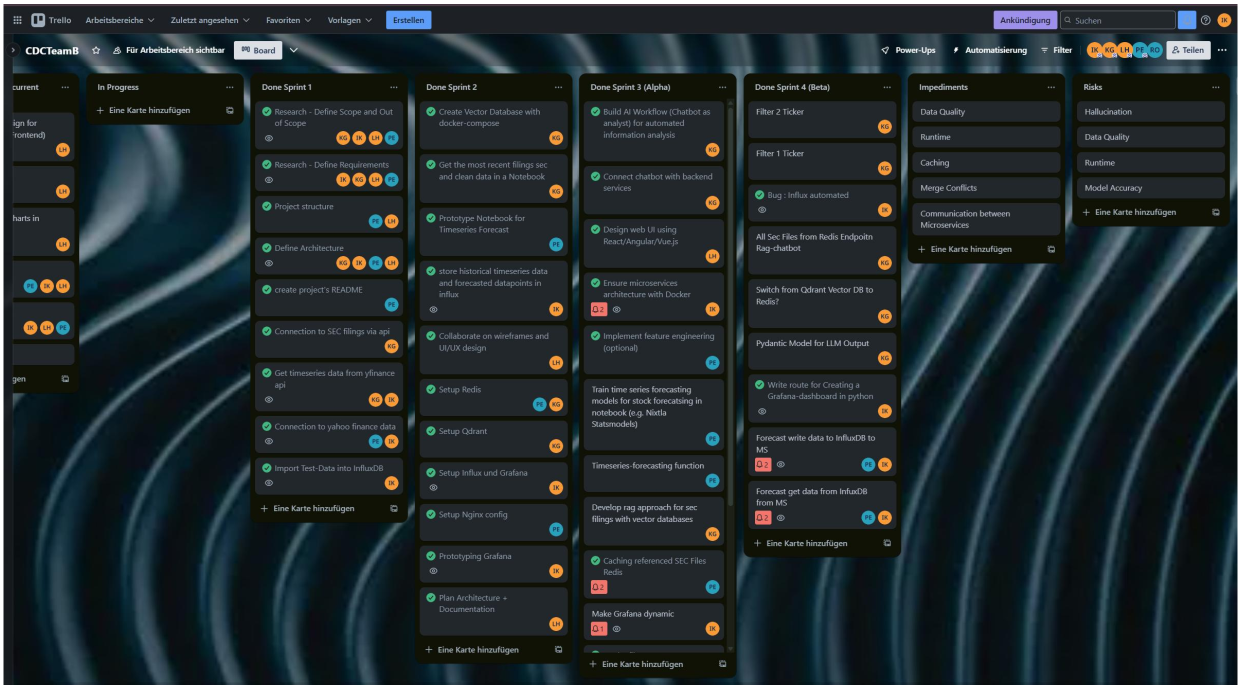
Done Sprint 4 (Beta)

- Filter 2 Ticker — KG
- Filter 1 Ticker — KG
- ✓ Bug : Influx automated — IK
- All Sec Files from Redis Endpoitn Rag-chatbot — KG
- Switch from Qdrant Vector DB to Redis? — KG
- Pydantic Model for LLM Output — KG
- ✓ Write route for Creating a Grafana-dashboard in python — IK
- Forecast write data to InfluxDB to MS — 2 — PE IK
- Forecast get data from InfuxDB from MS — 2 — PE IK
- Eine Karte hinzufügen

Impediments

- Data Quality
- Runtime
- Caching
- Merge Conflicts
- Communication Microservices
- Eine Karte h

Trello Screenshot 2



current

- ign for rontend) — LH
- LH
- harts in — LH
- PE IK LH
- IK LH PE
- gen

In Progress

- + Eine Karte hinzufügen

Done Sprint 1

- ✓ Research - Define Scope and Out of Scope — KG IK LH PE
- ✓ Research - Define Requirements — IK KG LH PE
- ✓ Project structure — PE LH
- ✓ Define Architecture — KG IK PE LH
- ✓ create project's README — PE
- ✓ Connection to SEC filings via api — KG
- ✓ Get timeseries data from yfinance api — KG IK
- ✓ Connection to yahoo finance data — PE IK
- ✓ Import Test-Data into InfluxDB — IK
- + Eine Karte hinzufügen

Done Sprint 2

- ✓ Create Vector Database with docker-compose — KG
- ✓ Get the most recent filings sec and clean data in a Notebook — KG
- ✓ Prototype Notebook for Timeseries Forecast — PE
- ✓ store historical timeseries data and forecasted datapoints in influx — IK
- ✓ Collaborate on wireframes and UI/UX design — LH
- ✓ Setup Redis — PE KG
- ✓ Setup Qdrant — KG
- ✓ Setup Influx und Grafana — IK
- ✓ Setup Nginx config — PE
- ✓ Prototyping Grafana — IK
- ✓ Plan Architecture + Documentation — LH
- + Eine Karte hinzufügen

Done Sprint 3 (Alpha)

- ✓ Build AI Workflow (Chatbot as analyst) for automated information analysis — KG
- ✓ Connect chatbot with backend services — KG
- ✓ Design web UI using React/Angular/Vue.js — LH
- ✓ Ensure microservices architecture with Docker — 2 — IK
- ✓ Implement feature engineering (optional) — PE
- Train time series forecasting models for stock forecatsing in notebook (e.g. Nixtla Statsmodels) — PE
- Timeseries-forecasting function — PE
- Develop rag approach for sec filings with vector databases — KG
- ✓ Caching referenced SEC Files Redis — 2 — PE
- Make Grafana dynamic — 1 — IK
- + Eine Karte hinzufügen

Done Sprint 4 (Beta)

- Filter 2 Ticker — KG
- Filter 1 Ticker — KG
- ✓ Bug : Influx automated — IK
- All Sec Files from Redis Endpoitn Rag-chatbot — KG
- Switch from Qdrant Vector DB to Redis? — KG
- Pydantic Model for LLM Output — KG
- ✓ Write route for Creating a Grafana-dashboard in python — IK
- Forecast write data to InfluxDB to MS — 2 — PE IK
- Forecast get data from InfuxDB from MS — 2 — PE IK
- + Eine Karte hinzufügen

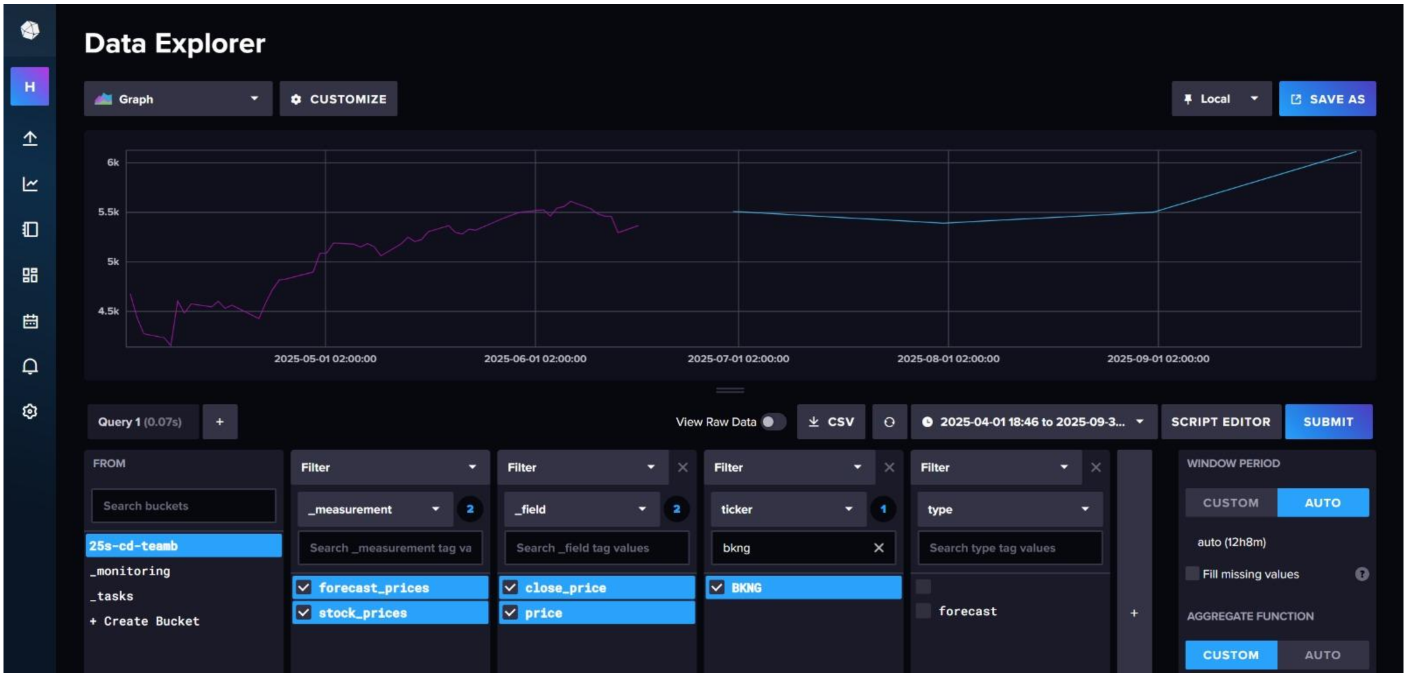
Impediments

- Data Quality
- Runtime
- Caching
- Merge Conflicts
- Communication between Microservices
- + Eine Karte hinzufügen

Risks

- Hallucination
- Data Quality
- Runtime
- Model Accuracy
- + Eine Karte hinzufügen

InfluxDB



Query 1 (0.07s)

2025-04-01 18:46 to 2025-09-3...

FROM: 25s-cd-teamb

_measurement: ☑ forecast_prices ☑ stock_prices

_field: ☑ close_price ☑ price

ticker: bkng — ☑ BKNG

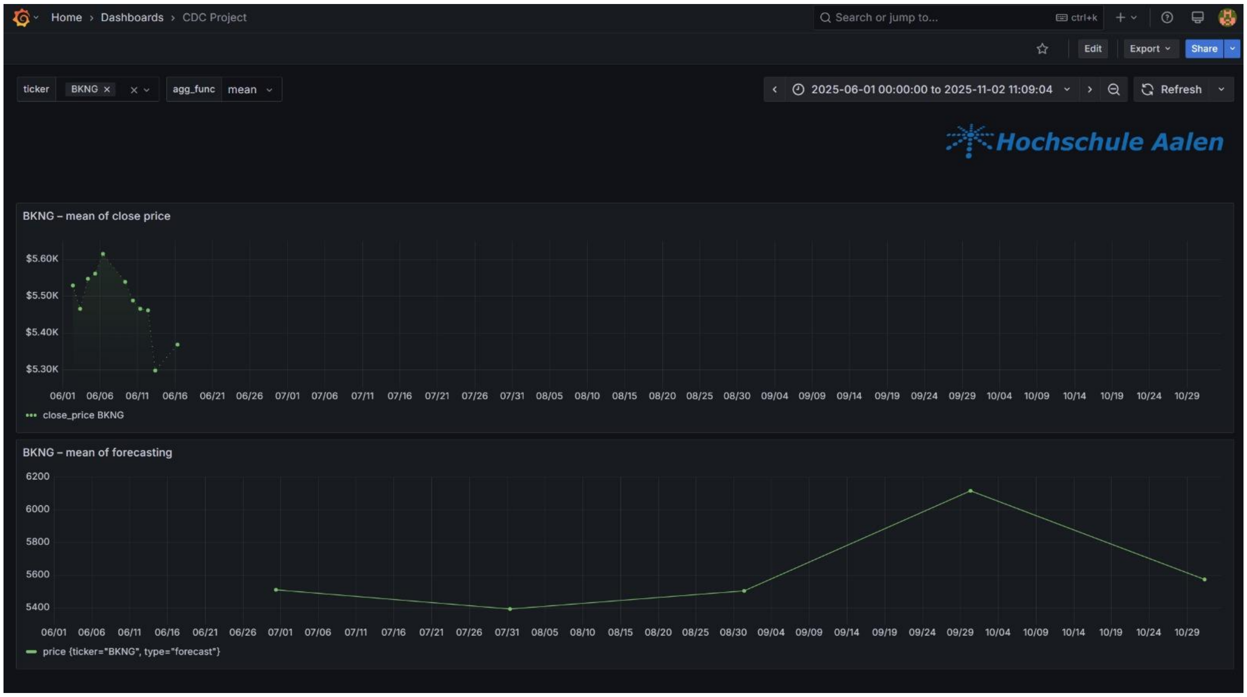
type: □ □ forecast

WINDOW PERIOD: AUTO — auto (12h8m)

□ Fill missing values

AGGREGATE FUNCTION: CUSTOM

Grafana



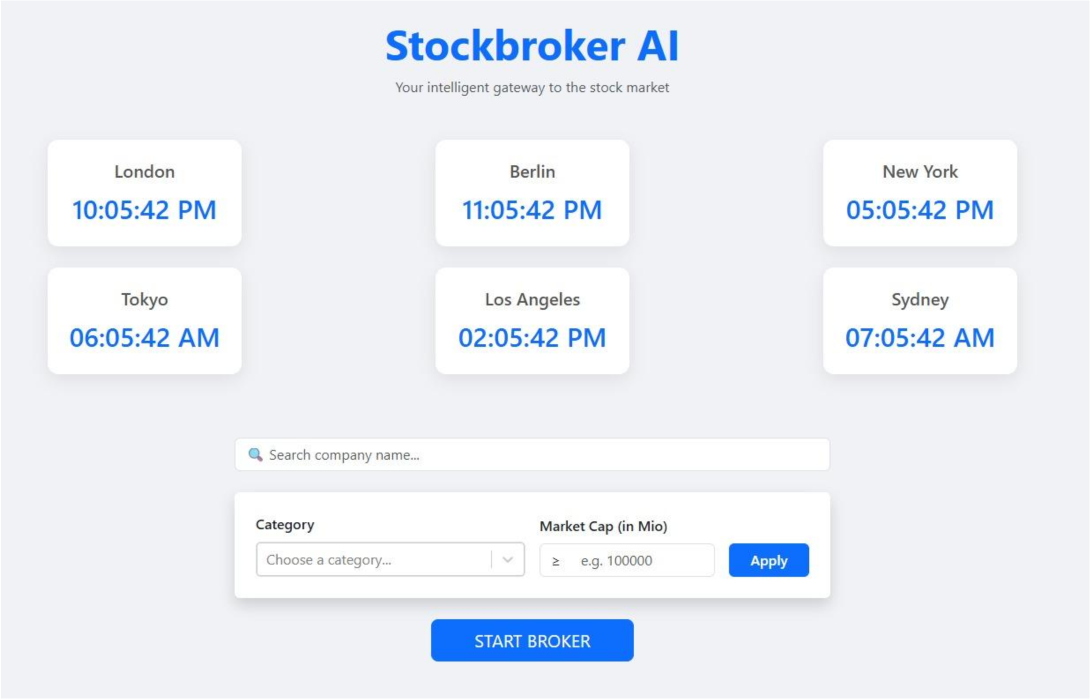
Home › Dashboards › CDC Project

ticker BKNG

agg_func mean

2025-06-01 00:00:00 to 2025-11-02 11:09:04

Landing Page



Stockbroker AI

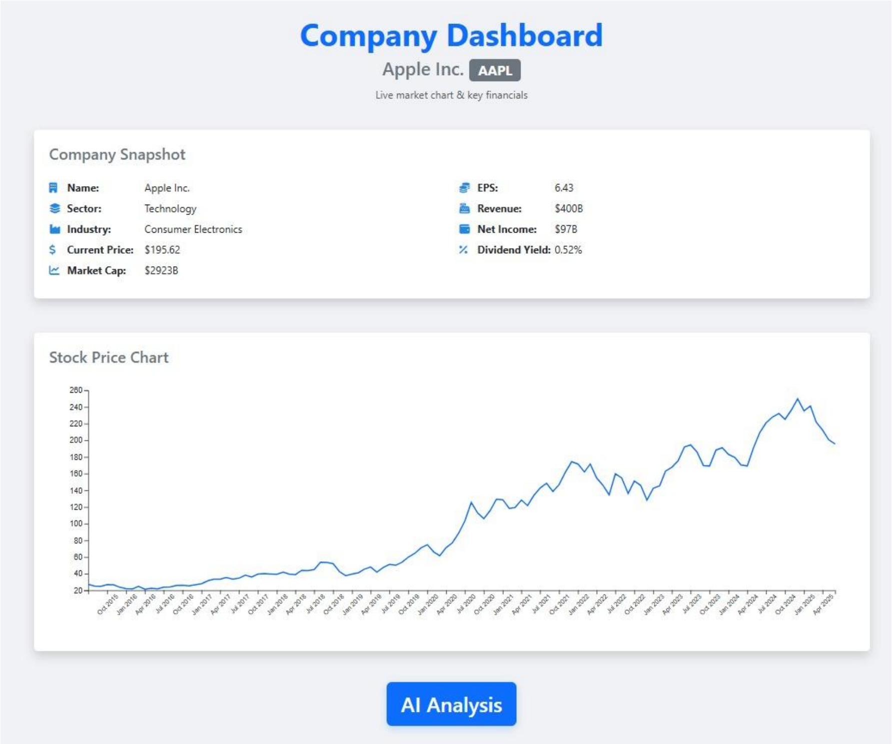
Your intelligent gateway to the stock market

London	Berlin	New York
10:05:42 PM	11:05:42 PM	05:05:42 PM

Tokyo	Los Angeles	Sydney
06:05:42 AM	02:05:42 PM	07:05:42 AM

Aalen University

Company Dashboard



Company Dashboard

Apple Inc. AAPL

Live market chart & key financials

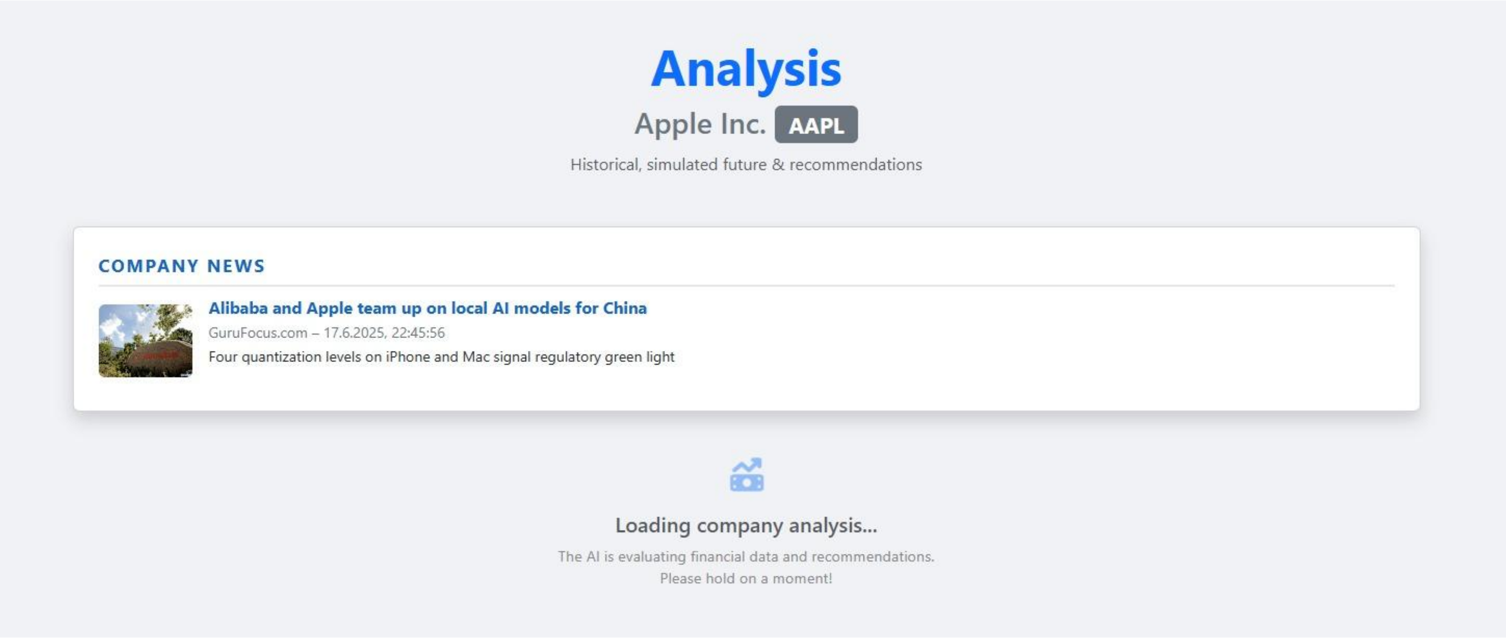
Company Snapshot

Name:	Apple Inc.	**EPS:**	6.43
Sector:	Technology	**Revenue:**	$400B
Industry:	Consumer Electronics	**Net Income:**	$97B
Current Price:	$195.62	**Dividend Yield:**	0.52%
Market Cap:	$2923B		

Stock Price Chart

Aalen University

Analysis Page 1



Analysis

Apple Inc. AAPL

Historical, simulated future & recommendations

COMPANY NEWS

Alibaba and Apple team up on local AI models for China

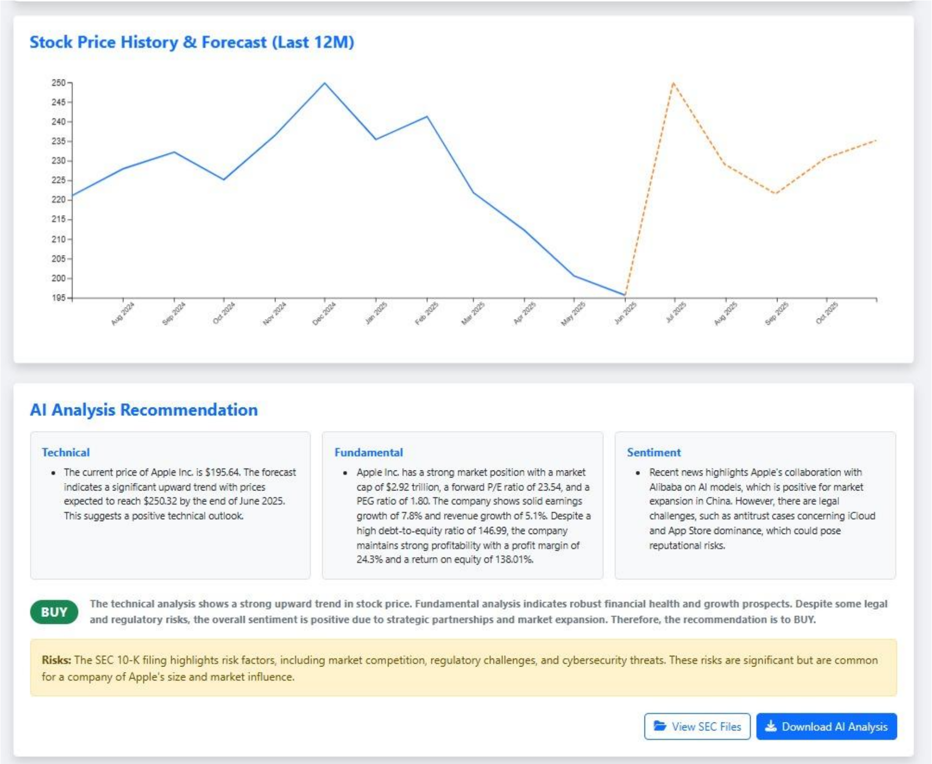
GuruFocus.com – 17.6.2025, 22:45:56

Four quantization levels on iPhone and Mac signal regulatory green light

Loading company analysis...

The AI is evaluating financial data and recommendations.
Please hold on a moment!

Analysis Page 2



Stock Price History & Forecast (Last 12M)

AI Analysis Recommendation

Technical

- The current price of Apple Inc. is $195.64. The forecast indicates a significant upward trend with prices expected to reach $250.32 by the end of June 2025. This suggests a positive technical outlook.

Fundamental

- Apple Inc. has a strong market position with a market cap of $2.92 trillion, a forward P/E ratio of 23.54, and a PEG ratio of 1.80. The company shows solid earnings growth of 7.8% and revenue growth of 5.1%. Despite a high debt-to-equity ratio of 146.99, the company maintains strong profitability with a profit margin of 24.3% and a return on equity of 138.01%.

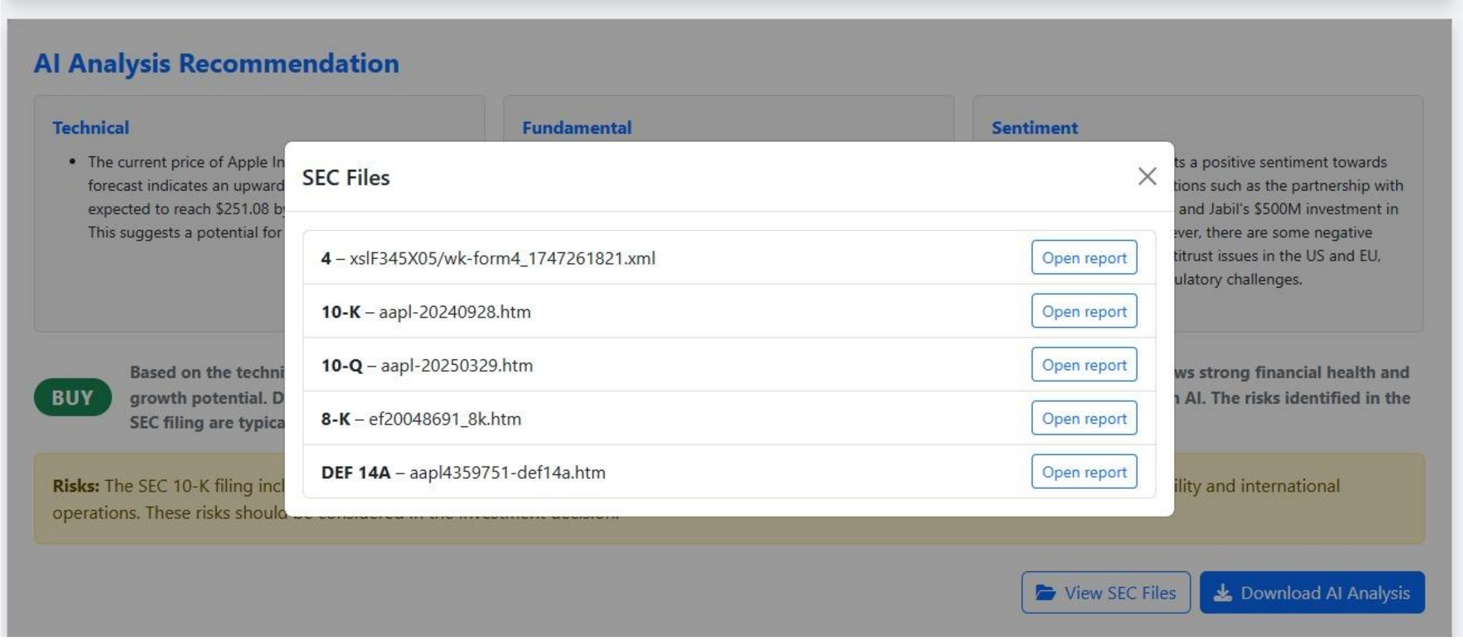
Sentiment

- Recent news highlights Apple's collaboration with Alibaba on AI models, which is positive for market expansion in China. However, there are legal challenges, such as antitrust cases concerning iCloud and App Store dominance, which could pose reputational risks.

BUY The technical analysis shows a strong upward trend in stock price. Fundamental analysis indicates robust financial health and growth prospects. Despite some legal and regulatory risks, the overall sentiment is positive due to strategic partnerships and market expansion. Therefore, the recommendation is to BUY.

Risks: The SEC 10-K filing highlights risk factors, including market competition, regulatory challenges, and cybersecurity threats. These risks are significant but are common for a company of Apple's size and market influence.

SEC File Model



AI Analysis Recommendation

Technical

- The current price of Apple In
forecast indicates an upward
expected to reach $251.08 b
This suggests a potential for

Fundamental

Sentiment

ts a positive sentiment towards
tions such as the partnership with
and Jabil's $500M investment in
ever, there are some negative
titrust issues in the US and EU,
ulatory challenges.

BUY **Based on the techni** **ws strong financial health and**
growth potential. D **AI. The risks identified in the**
SEC filing are typica

Risks: The SEC 10-K filing incl ility and international
operations. These risks should

SEC Files

4 – xslF345X05/wk-form4_1747261821.xml

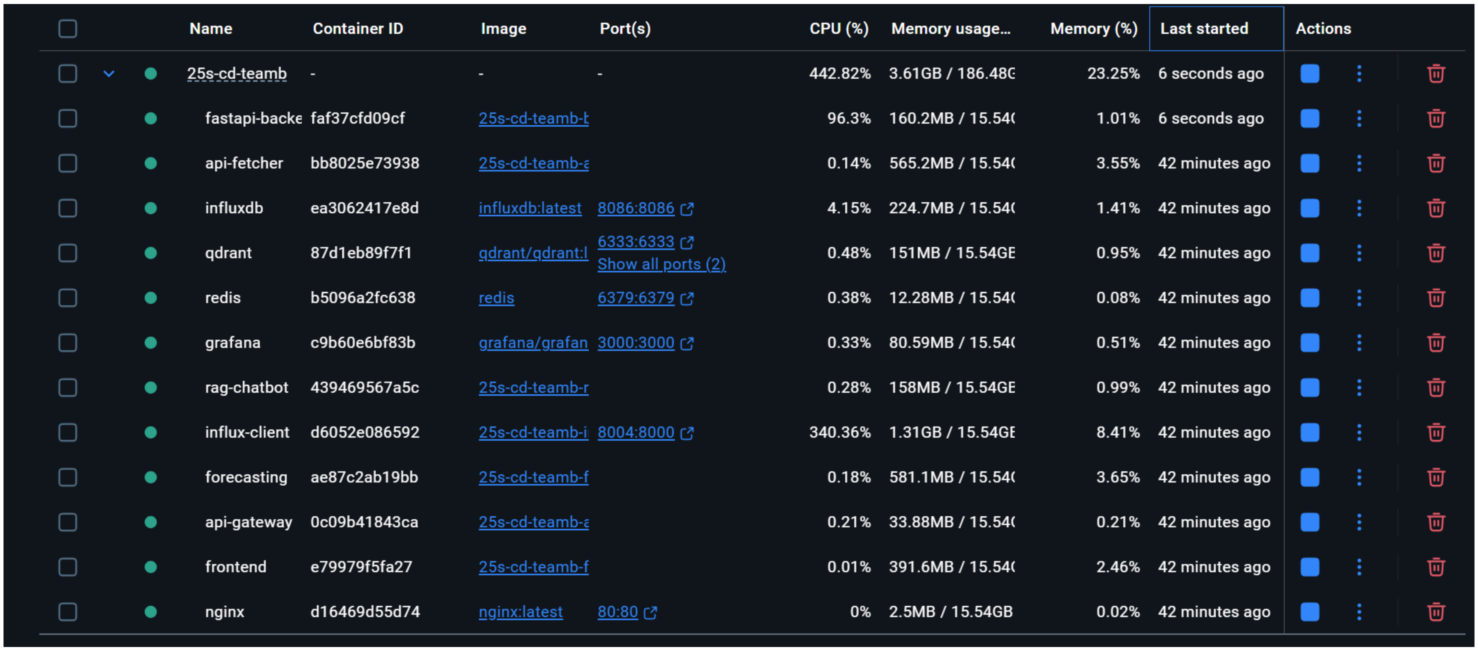
10-K – aapl-20240928.htm

10-Q – aapl-20250329.htm

8-K – ef20048691_8k.htm

DEF 14A – aapl4359751-def14a.htm

Docker



☐			Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Last started	Actions
☐	⌄	●	25s-cd-teamb	-	-	-	442.82%	3.61GB / 186.48G	23.25%	6 seconds ago	
☐		●	fastapi-backe	faf37cfd09cf	25s-cd-teamb-t		96.3%	160.2MB / 15.54(	1.01%	6 seconds ago	
☐		●	api-fetcher	bb8025e73938	25s-cd-teamb-a		0.14%	565.2MB / 15.54(	3.55%	42 minutes ago	
☐		●	influxdb	ea3062417e8d	influxdb:latest	8086:8086	4.15%	224.7MB / 15.54(	1.41%	42 minutes ago	
☐		●	qdrant	87d1eb89f7f1	qdrant/qdrant:l	6333:6333 Show all ports (2)	0.48%	151MB / 15.54GE	0.95%	42 minutes ago	
☐		●	redis	b5096a2fc638	redis	6379:6379	0.38%	12.28MB / 15.54(	0.08%	42 minutes ago	
☐		●	grafana	c9b60e6bf83b	grafana/grafan	3000:3000	0.33%	80.59MB / 15.54(	0.51%	42 minutes ago	
☐		●	rag-chatbot	439469567a5c	25s-cd-teamb-r		0.28%	158MB / 15.54GE	0.99%	42 minutes ago	
☐		●	influx-client	d6052e086592	25s-cd-teamb-i	8004:8000	340.36%	1.31GB / 15.54GE	8.41%	42 minutes ago	
☐		●	forecasting	ae87c2ab19bb	25s-cd-teamb-f		0.18%	581.1MB / 15.54(	3.65%	42 minutes ago	
☐		●	api-gateway	0c09b41843ca	25s-cd-teamb-a		0.21%	33.88MB / 15.54(	0.21%	42 minutes ago	
☐		●	frontend	e79979f5fa27	25s-cd-teamb-f		0.01%	391.6MB / 15.54(	2.46%	42 minutes ago	
☐		●	nginx	d16469d55d74	nginx:latest	80:80	0%	2.5MB / 15.54GB	0.02%	42 minutes ago	

Risks/Impediments

- Residual Bugs
- External API Reliablity (Yfinance downtime or throttling)
- Long Runtime
- Service Sync Delays (InfluxDB, container orchestration)
- UX Limitations (data too slow)

Completed Beta

- Microservices orchestration
- Filtering
- Bugs fixed
- Rag-chatbot
- Switch from Qdrant -> Redis
- Pydantic Model for LLM Output
- Route Grafana
- Forecast Data in Influx + Grafana

Todo Final submission

- Final Integration Testing across all microservices
- Deployment
- Cleaning up code and Dockerfiles for maintainability
- Ensure responsive design for Desktop -> Frontend
- Embedding Grafana charts in Frontend
- Testing