volue smartPulse

FORECAST SERVICE

System Architecture

Version 1.0 ● NOVEMBER 15, 2025

Anar Mehdiyev

Vienna, 2025

Summary

This document describes the requested microservice solution according to the Technical Interview Assessment requirements. It consists of the following major chapters:

1. **Architecture** of the Forecast Service using the C4 Model, a hierarchical approach to software architecture visualization
2. **Decision Log** and rationale of used technology stack
3. **API Specification** of the three API endpoints required for the Forecast Service
4. **Instructions** to set up and execute the service

System Architecture

FORECAST SERVICE

Volue | smartPulse

Technical Interview Assessment

VERSION: 1.0 REVISION DATE: 15.11.2025

Name	Title	Notes	Date
Murat Yilmaz	Reviewer		
Burak Serhat	Reviewer		
Jörg Lienhardt	Reviewer		
Michael Moll	Reviewer		
Anar Mehdiyev	Author	Initial version	15.11.2025

Contents

1. C4 Model System Architecture

This chapter describes the architecture of the Forecast Service using the C4 Model, a hierarchical approach to software architecture visualization consisting of four levels of abstraction:

Level	Description	Details
1	**Context -** System context and external systems	Diagram and description present main business processes
2	**Container -** Major containers and their relationships	Relationships of FastAPI, PostgreSQL, Apache Kafka, Docker Compose
3	**Component -** Internal components within containers	Detailed description of components of FastAPI routers, Service classes, Repository pattern, SQLModel
4	**Code -** Key code structures and relationships	Overview of Python classes,SQLModel, kafka client

Layered Architecture

- **API Layer:** HTTP handling and request validation
- **Service Layer:** Business logic orchestration
- **Repository Layer:** Data persistence abstraction
- **Model Layer:** Domain entity definitions
- **Configuration Layer:** Config, DB, messaging

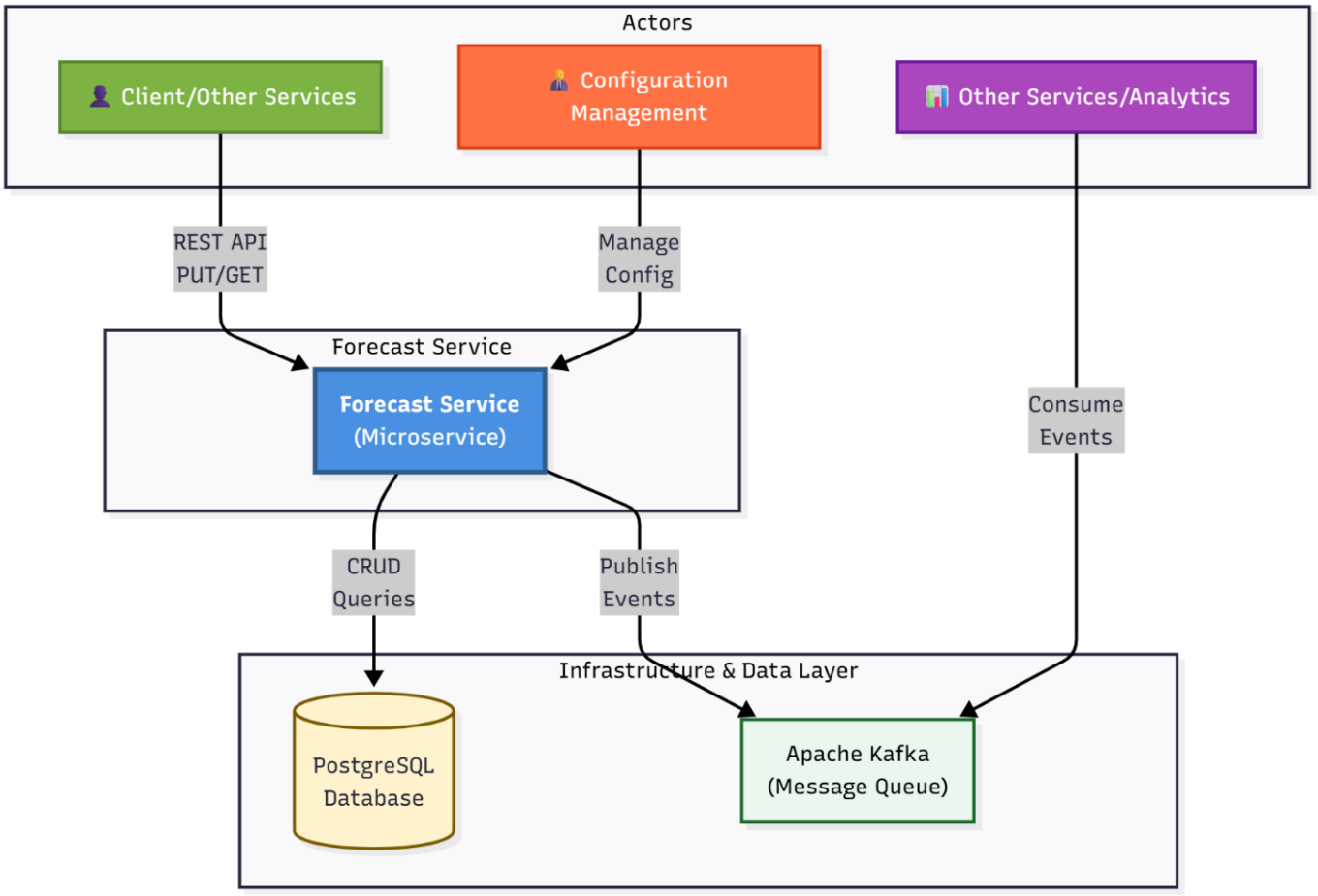
1.2. Context

1.2.1 Purpose

The **Context Diagram** provides the “big picture” view of the system’s architecture. The system context diagram shows the Forecast Service as a “black box” at the center, depicting interactions with external systems, actors, and dependencies.

1.2.2 Diagram

.



1.2.3 Description

The **Forecast Service** is a RESTful microservice that:

- **Receives requests** from clients (applications) via HTTP REST API
- **Stores data** persistently in a PostgreSQL database
- **Publishes events** to Apache Kafka for consumer systems (analytics, other services)

Key Interactions:

- **Clients / Other Services** → **Forecast Service**: Create, update, and retrieve forecast data via REST endpoints
- **Forecast Service** → **PostgreSQL**: Persist forecast estimates and plant information
- **Forecast Service** → **Kafka**: Emit position-change events when forecasts are created/updated
- **Other Services /Analytics** ← **Kafka**: Consume events for other services and analytics

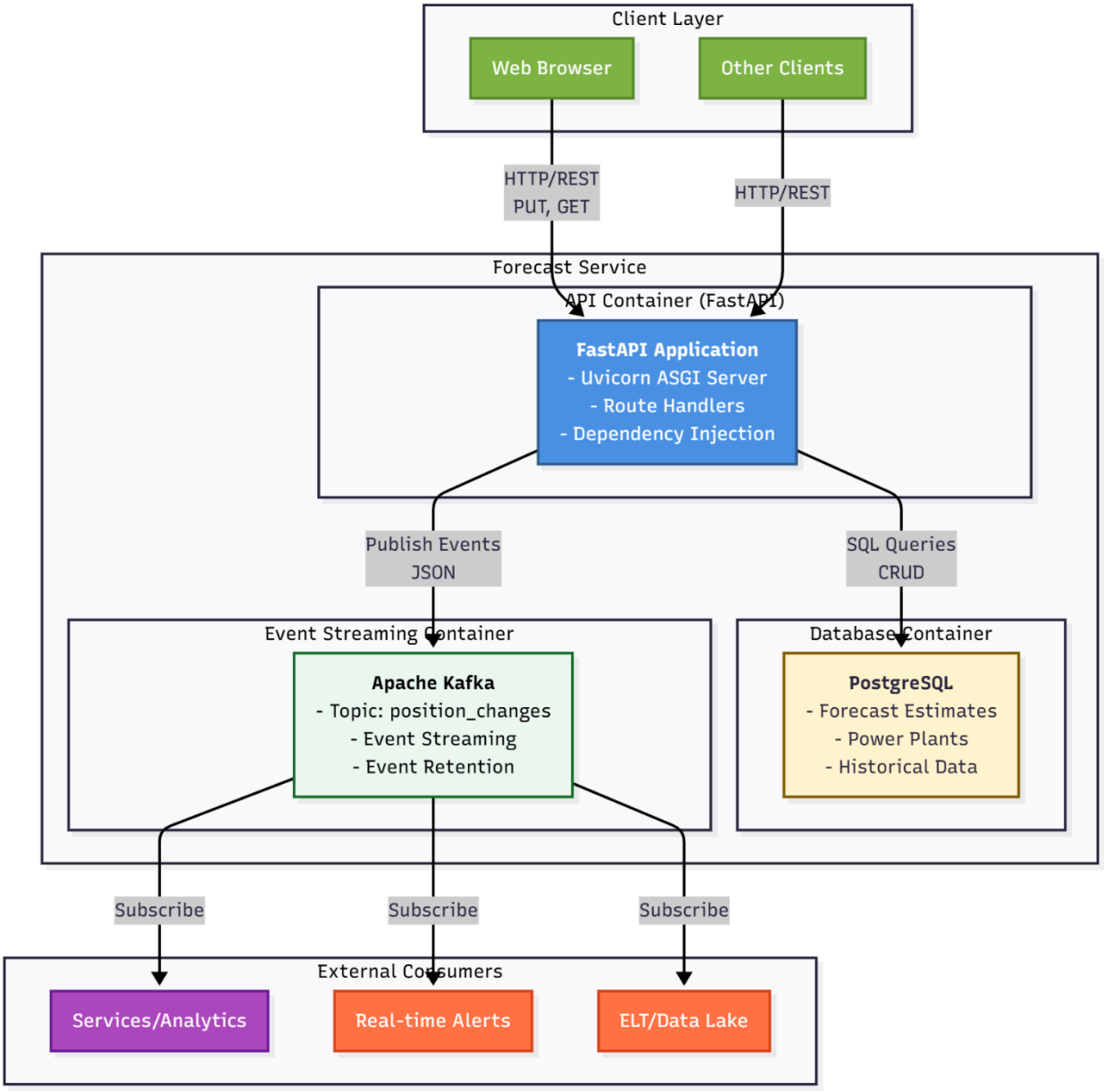
1.3. Container

1.3.1 Purpose

The container diagram zooms into the Forecast Service, showing its major building blocks (containers) and their communication patterns. A “container” here refers to a deployable unit (service, database, queue, etc.).

1.3.2 Diagram

.



1.3.3 Description

The Forecast Service is composed of three primary containers:

API Container (FastAPI)

Technology: Python FastAPI framework

Responsibilities:

- Handle HTTP REST requests from clients
- Execute business logic
- Manage database queries through repositories
- Emit Kafka events for state changes

Runs on: Port 8000

Database Container (PostgreSQL)

Technology: PostgreSQL relational database

Responsibilities:

- Persist forecast estimates and time-series data
- Store power plant master data
- Support complex aggregation queries
- Maintain data integrity with foreign keys

Runs on: Port 5432

Key Tables:

- powerplant — Plant master data
- forecastestimate — Hourly forecast estimates

Apache Kafka

Technology: Apache Kafka

Responsibilities:

- Publish position-change events from the API
- Decouple API from downstream consumers
- Enable real-time event distribution

Runs on: Port 9092 (internal), 29092 (external)

Key Topic: position_changes — Emitted when forecasts change

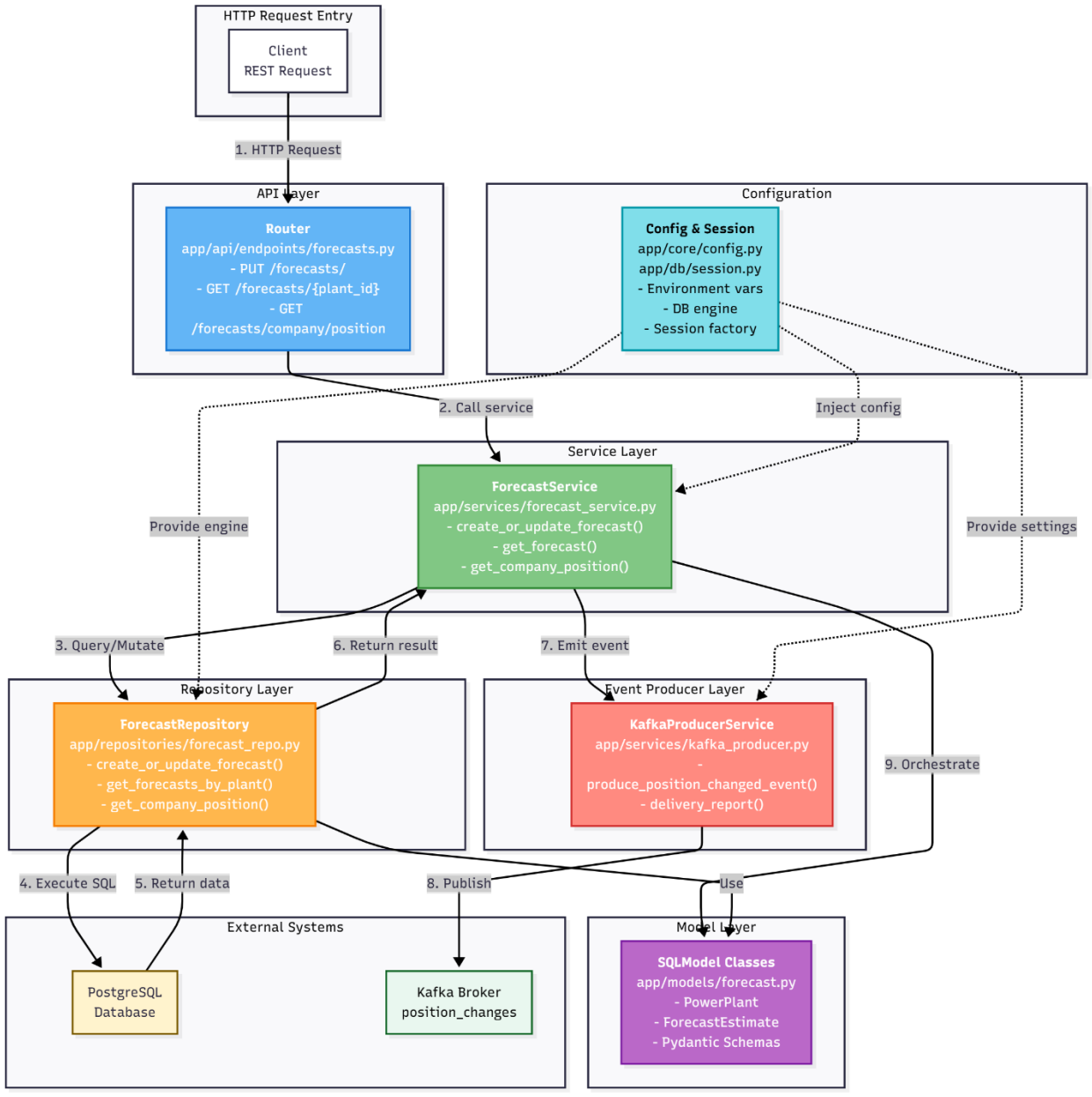
1.4. Component

1.4.1 Purpose

The component diagram zooms into the API container, showing its internal components, layers, and their interactions. This level reveals the internal structure and responsibilities.

1.4.2 Diagram

.



Data Flow for a Forecast Update (PUT /forecasts/):

1. Client sends HTTP PUT request with forecast payload
2. Router validates input schema and calls ForecastService.create_or_update_forecast()
3. Service delegates to Repository.create_or_update_forecast() to persist to DB
4. Service calls Producer.produce_position_changed_event() to emit Kafka event
5. Service returns the saved forecast to the client

1.4.3 Description

Note: Detailed description and responsibilities of components of each layer are out of scope of this assessment project. Can be provided upon request.

API Layer (Router)

Module: app/api/endpoints/forecasts.py

Service Layer (ForecastService)

Module: app/services/forecast_service.py

Repository Layer (ForecastRepository)

Module: app/repositories/forecast_repo.py

Event Producer Layer (KafkaProducerService)

Module: app/services/kafka_producer.py

Model Layer

Module: app/models/forecast.py

Configuration Layer

Modules: app/core/config.py, app/db/session.py

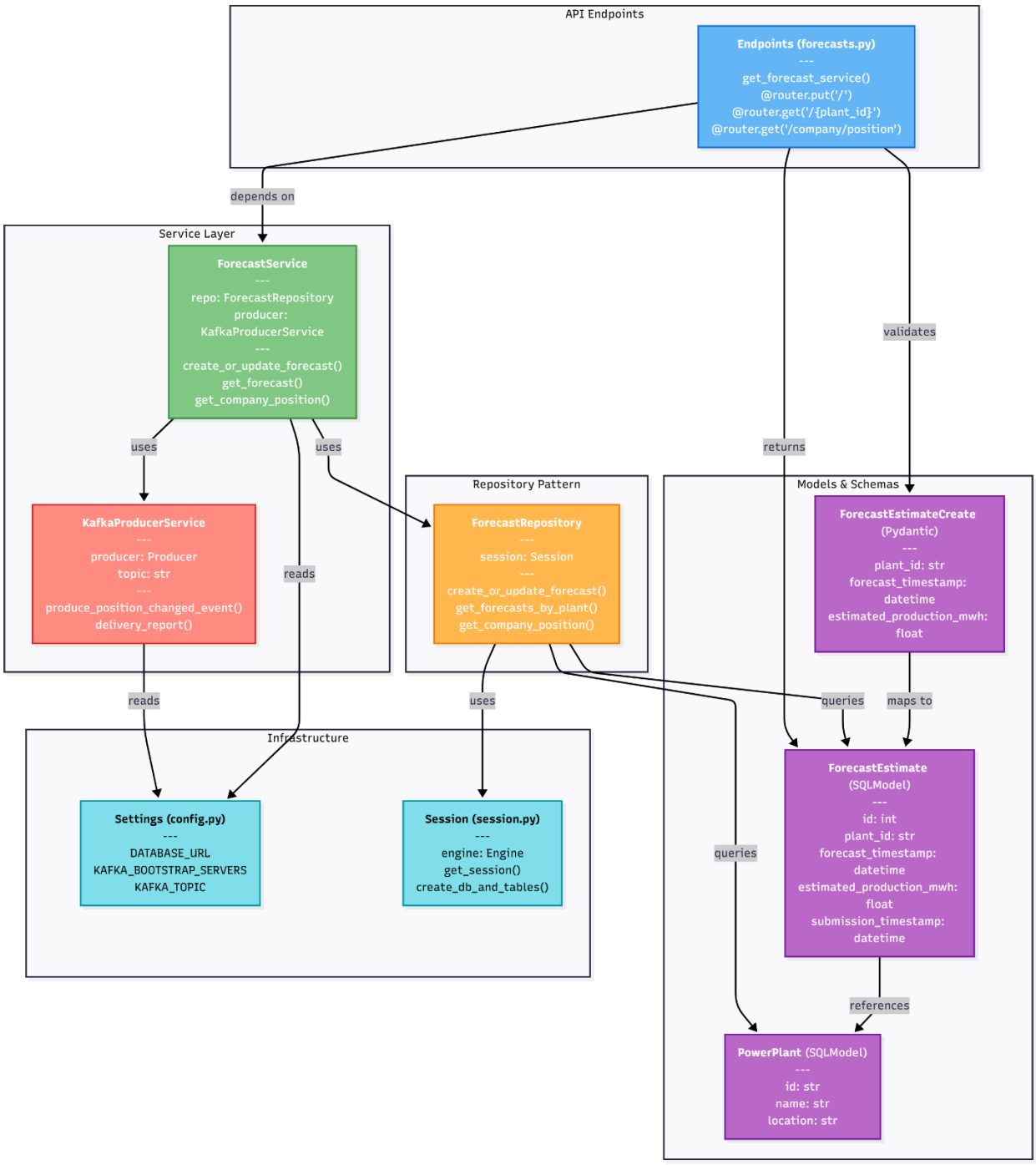
1.5. Code

1.5.1 Purpose

The code-level diagram shows key classes, attributes, and methods with their relationships. This level is most detailed and useful for developers.

1.5.2 Diagram

.



1.5.3 Description

Descriptions of classes, routes, abstractions, attributes, parameters, methods and logic are not provided, as are out of scope of this assessment project. Can be provided upon request.

2. API Specification

This chapter describes the three API endpoints required for the Forecast Service.

Base URL: /api/v1/

The microservice exposes the endpoints below:

Method	Path	Description
PUT	/api/v1/forecasts	Creates a new hourly estimate or updates an existing one. Emits Kafka event.
GET	/api/v1/forecasts/{plant_id}	Retrieves a list of raw hourly estimates for a single plant within a selected date range.
GET	/api/v1/company/position	Calculates and returns the total aggregated mWh forecast across all locations (Turkey, Bulgaria, Spain).

2.1. Create or Update forecast estimates

2.1.1 Description

This endpoint is used to submit a new hourly production estimate or update an existing one for the same plant and forecast hour. This request triggers the 'PositionChanged' event to Kafka.

Detail	Specification
Path	/api/v1/forecasts
Method	PUT
Response (200 OK)	Returns the created/updated ForecastEstimate object.

2.1.2 Request Body

```
{
 "plant_id": "TR_001",
 "forecast_timestamp": "2025-11-15T14:00:00Z",
 "estimated_production_mwh": 42.5
}
```

Field	Type	Description	Example
plant_id	string	Unique identifier for the power plant: Turkey: TR_001 Bulgaria: BG_001 Spain: ES_001	"TR_001"
forecast_timestamp	datetime	The specific hour for which the production is estimated.	"2026-01-01T10:00:00Z"
estimated_production_mwh	number	The estimated production value for that hour, in mWh.	15.5

2.1.3 Response Example


```
{
 "id": 123,
 "plant_id": "TR_001",
 "forecast_timestamp": "2025-11-15T14:00:00Z",
 "estimated_production_mwh": 42.5,
 "submission_timestamp": "2025-11-15T10:30:15.123456"
}
```


2.2. Get hourly forecasts by plant

2.2.1 Description

This endpoint retrieves the raw hourly production estimates for a specific power plant for selected period.

Detail	Specification
Path	/api/v1/forecasts/{**plant_id**}
Method	GET
Response (200 OK)	Returns a list of ForecastEstimate objects.

2.2.2 Query Parameters

Parameter	Type	Description	Format
plant_id (Path)	String	Plant identifiers: Turkey: TR_001	TR_001 BG_001

		Bulgaria: BG_001 Spain: ES_001	ES_001
start_date	datetime	Start of the forecast period (inclusive).	2026-01-01T00:00:00Z
end_date	datetime	End of the forecast period (exclusive).	2026-01-02T00:00:00Z

2.2.3 Response Example


```
{
  "id": 123,
  "plant_id": "TR_001",
  "forecast_timestamp": "2025-11-15T14:00:00Z",
  "estimated_production_mwh": 42.5,
  "submission_timestamp": "2025-11-15T10:30:15.123456"
 }
```


2.3. Get company total forecast position

2.3.1 Description

This endpoint calculates and returns the aggregated total forecast across all locations (Turkey, Bulgaria and Spain) for a selected date and time range.

Detail	Specification
Path	/api/v1/company/position
Method	GET
Response (200 OK)	Returns the total aggregated and by location.

2.3.2 Query Parameters

Parameter	Type	Description	Format
start_date	datetime	Start of the forecast period (inclusive).	2026-01-01T00:00:00Z
end_date	datetime	End of the forecast period (exclusive).	2026-01-02T00:00:00Z

2.3.3 Response Example


```
{
 "start_date": "2025-11-15T00:00:00+01:00",
 "end_date": "2025-11-16T00:00:00+01:00",
 "total_forecast_mwh": 1250.75,
 "by_location": {
  "Turkey": 450.25,
  "Bulgaria": 380.5,
  "Spain": 420.0
 }
}
```


3. Decision Log

This project implements a basic Forecast Service for an energy trading platform, as specified in the Technical Assessment document. The designed service manages hourly production estimates from three power plants (Turkey, Bulgaria, Spain) and provides aggregated results to display the company's total position.

Key Features

- **Layered Architecture**: Controller, Service, and Repository layers.
- **Asynchronous APIs**: Built using FastAPI for high performance and concurrency.
- **Persistent Storage**: Uses PostgreSQL for reliable, structured data storage.
- **Event-Driven**: Implemented the requirement to emit a 'PositionChanged' event to Kafka on every forecast update.
- **Containerization**: Fully deployable and isolated using Docker Compose.

Technology Stack & Decisions

Component	Technology	Rationale
Language/Framework	**Python/ FastAPI**	Chosen for rapid development and excellent asynchronous (async/await) performance.
Database	**PostgreSQL/ SQLModel**	Optimal for structured data and complex aggregation queries. SQLModel simplifies the data model.
Event	**Apache Kafka**	Provides a durable and scalable log of all forecast updates, and delivers easy access to consumers
Deployment	**Docker/ Docker Compose**	Fulfills the requirement for deployability. Provides an isolated, portable, and repeatable environment.

4. Instructions

Prerequisites
Docker
Docker Compose
Git

Running with Docker Compose
Docker compose builds the application, pulls and configures PostgreSQL, Kafka and connects them all via a Docker network. On first startup, the service automatically loads the PostgreSQL database with the PowerPlant definitions:

plant_id	Location
TR_001	Turkey
BG_001	Bulgaria
ES_001	Spain

The helper script posts 72 sample hourly forecasts (24h × 3 plants) to the API.

4.1. Setup and Execution

Clone the Repository
git clone https://github.com/anar4sm/forecast-service.git
cd forecast-service

Start Services:
docker compose up --build -d

Verify Services:
docker compose ps

Access the API:
The service is available at http://localhost:8000
Navigate to the automatically generated, interactive OpenAPI documentation (Swagger UI):
http://localhost:8000/api/v1/docs

Stop Services:
docker compose down

5. Next steps

To be added:

- Deployment Architecture:

    - Local Deployment
    - Cloud (Azure) Deployment
- Testing
- Security
- CI/CD Pipeline
- Monitoring
- Feature enhancements