Microservice Architecture and Programming

Assignment

**Helios RTB Engine**

Real-Time Bidding System

**Technical Documentation & Architecture Report**

Project By:

**22BCE041 – Jigar Bhoye**

**22BCE020 – Anup Patel**

**22BCE064 – Priyang Desai**

GitHub URL: [jigarbhoye04/devops:](https://github.com/jigarbhoye04/devops)

# Abstract

The Helios RTB Engine is a production-grade, distributed real-time bidding system de- signed to simulate and demonstrate the core components of modern programmatic adver- tising platforms. Built using microservices architecture, the system processes ad place- ment opportunities in real-time, enriches them with user profile data, executes sophisti- cated bidding algorithms, and provides comprehensive analytics through a modern web dashboard.

This report presents a comprehensive technical analysis of the Helios RTB Engine, covering its architecture, implementation details, deployment strategies, and operational characteristics. The system demonstrates mastery of contemporary software engineer- ing practices including polyglot programming, event-driven architecture, containerization, and cloud-native deployment.

##### Key Highlights:

* **Sub-100ms Processing:** Real-time bid processing with ultra-low latency requirements
* **Polyglot Microservices:** Six independent services written in Go, Python, and Node.js/TypeScript
* **Event-Driven Architecture:** Apache Kafka-based asynchronous message processing
* **Production-Ready:** Comprehensive observability, health checks, and secu- rity measures
* **Scalable Design:** Kubernetes-ready deployment with horizontal scaling ca- pabilities

1

**Chapter 1 Introduction**

## Project Overview

The Helios RTB (Real-Time Bidding) Engine represents a sophisticated implementation of a Demand-Side Platform (DSP) within the AdTech ecosystem. When a user visits a website with available ad space, an auction occurs in milliseconds to determine which advertisement will be displayed. Helios simulates this entire end-to-end process, from receiving bid requests to executing auctions and providing detailed analytics.

### Real-World Context

Real-time bidding is the foundation of modern digital advertising, powering platforms such as:

* + - * **Google Ads:** Processing billions of auctions daily
      * **Facebook Ads Platform:** Targeting users based on detailed profiles
      * **The Trade Desk:** Independent demand-side platform for advertisers
      * **Amazon Advertising:** E-commerce integrated ad platform These systems operate under stringent performance requirements:

|  |  |
| --- | --- |
| **Metric** | **Requirement** |
| Latency per Auction | *<* 100 milliseconds |
| Daily Auction Volume | Billions of requests |
| Concurrent Users | Millions simultaneously |
| Revenue Scale | Hundreds of billions annually |
| Availability | 99.99% uptime |

Table 1.1: Industry Standard Performance Metrics

3

##### Helios RTB Engine 4

## System Objectives

### Functional Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Bid Request Handling | Accept and validate incoming HTTP POST requests  containing bid opportunities |
| User Enrichment | Retrieve user profile data including interests and de-  mographics in real-time |
| Bid Calculation | Execute algorithmic bidding logic based on user pro-  files and campaign parameters |
| Auction Execution | Simulate ad exchange auction with threshold checks  and probability-based winner selection |
| Data Persistence | Store all auction outcomes permanently for analytics  and reporting |
| Analytics Dashboard | Provide web-based interface for viewing performance  metrics and trends |

Table 1.2: Functional Requirements

Real-Time Bidding System - Technical Report

##### Helios RTB Engine 5

### Non-Functional Requirements

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| Performance | Process bid requests in under 100ms end-to-end |
| Scalability | Support horizontal scaling of stateless services via Ku-  bernetes |
| Reliability | Implement health checks and automatic service recov-  ery |
| Security | Run all containers as non-root users; manage secrets  securely |
| Maintainability | Structured logging, comprehensive documentation,  code quality standards |
| Observability | Prometheus metrics, health endpoints, request tracing |

Table 1.3: Non-Functional Requirements

## Technology Stack

The Helios RTB Engine leverages a carefully selected technology stack optimized for performance, scalability, and developer productivity.

### Programming Languages & Frameworks

|  |  |  |
| --- | --- | --- |
| lightgray **Language** | **Framework** | **Use Case** |
| Go | Native (net/http) | High-concurrency HTTP bid inges-  tion |
| Python | FastAPI | Bidding logic service |
| Python | Django + DRF | Analytics API and data persistence |
| TypeScript | Express.js | gRPC user profile service |
| JavaScript | Node.js | Auction simulator |
| TypeScript | Next.js + React | Advertiser dashboard frontend |

Table 1.4: Language and Framework Selection

Real-Time Bidding System - Technical Report

##### Helios RTB Engine 6

### Infrastructure Components

|  |  |  |
| --- | --- | --- |
| lightgray **Component** | **Version** | **Purpose** |
| Apache Kafka | 3.x | Event streaming and message bus |
| Apache Zookeeper | 3.x | Kafka cluster coordination |
| Redis | 7.x | In-memory cache for user profiles |
| PostgreSQL | 15.x | Relational database for analytics |
| Docker | 24.x+ | Container runtime |
| Docker Compose | 2.x+ | Local orchestration |
| Kubernetes | 1.28+ | Production container orchestration |

Table 1.5: Infrastructure Technology Stack

### Communication Protocols

|  |  |
| --- | --- |
| **Protocol** | **Usage** |
| HTTP/REST | External API endpoints, dashboard communication, health  checks |
| gRPC (HTTP/2) | Low-latency user profile lookups with Protocol Buffers serial-  ization |
| Kafka Protocol | Asynchronous inter-service communication via message topics |

Table 1.6: Communication Protocols

Real-Time Bidding System - Technical Report

**Chapter 2**

**System Architecture**

## Architectural Overview

The Helios RTB Engine follows a microservices architecture pattern with event-driven communication. Services are loosely coupled through Apache Kafka message topics, en- abling independent scaling, deployment, and development.

### High-Level Architecture Diagram

**Information**

The architecture implements the Database Per Service pattern, where each mi- croservice maintains its own data store. This ensures loose coupling and allows services to choose the optimal database technology for their specific requirements.

### Architecture Diagram

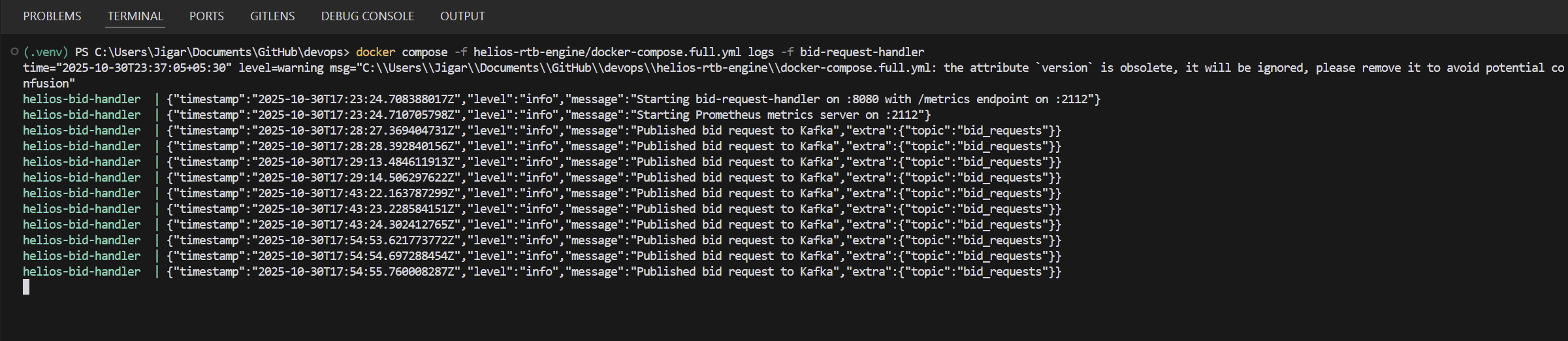
##### Helios RTB Engine 8

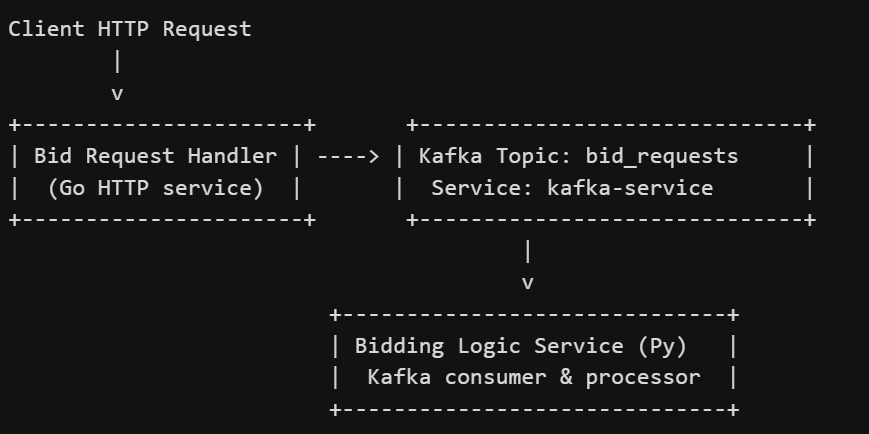
## Core Microservices

### Bid Request Handler (Go)

##### Overview

The Bid Request Handler serves as the system’s entry point, responsible for receiving HTTP POST requests containing bid opportunities and publishing them to Kafka for asynchronous processing.





Real-Time Bidding System - Technical Report

##### Helios RTB Engine 9

##### Technical Specifications

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| Language | Go 1.21+ |
| Primary Port | 8080 (HTTP API) |
| Metrics Port | 2112 (Prometheus) |
| Kafka Library | github.com/segmentio/kafka-go |
| Container Image Size | ∼20 MB (Alpine-based) |

Table 2.1: Bid Request Handler Specifications

##### Key Responsibilities

* + - 1. Accept HTTP POST requests at /bid endpoint
      2. Validate incoming JSON payload structure
      3. Generate unique request identifiers
      4. Publish validated requests to bid requests Kafka topic
      5. Return HTTP 202 Accepted for valid requests
      6. Expose /healthz endpoint for health checks
      7. Emit Prometheus metrics for monitoring

##### Sample Request Format

1 {

2 " request\_id ": " req -12345" ,

3 " user\_id ": " user -001" ,

4 " timestamp ": "2025 -10 -30 T12 :00:00 Z",

5 " site ": {

6 " domain ": " news. example . com ",

7 " page\_url ": "/ articles/ technology "

8 },

9 " device ": {

10 " ip ": "192.0.2.1" ,

11 " user\_agent ": " Mozilla /5.0..." ,

12 " device\_type ": " mobile "

13 }

14 }

Listing 2.2: Bid Request JSON Schema

Real-Time Bidding System - Technical Report

##### Helios RTB Engine 10

##### Why Go?

* + - * + **Concurrency:** Goroutines enable handling 10,000+ concurrent connections
        + **Performance:** Native compilation produces highly optimized binaries
        + **Simplicity:** Minimal runtime dependencies and fast startup times
        + **Network I/O:** Excellent built-in support for HTTP servers

### User Profile Service (Node.js/TypeScript)

##### Overview

The User Profile Service provides real-time access to user demographic and behavioral data through a gRPC interface. It maintains user profiles in Redis for sub-5ms response times.

##### Technical Specifications

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| Language | Node.js 20+ with TypeScript |
| gRPC Port | 50051 |
| Protocol | gRPC with Protocol Buffers |
| Data Store | Redis (in-memory cache) |
| Average Response Time | *<* 5 milliseconds |

Table 2.2: User Profile Service Specifications

##### Data Model

1 {

2 " user\_id ": " user -001" ,

3 " demographics ": {

4 " age ": 28 ,

5 " gender ": " M",

6 " location ": " US - CA - San Francisco "

7 },

8 " interests ": {

9 " technology ": 0.95 ,

10 " sports ": 0.70 ,

11 " travel ": 0.65 ,

12 " cooking ": 0.40 ,

13 " automotive ": 0.30

14 },

15 " engagement\_score ": 0.85 ,

16 " last\_seen ": "2025 -10 -30 T11 :45:00 Z"

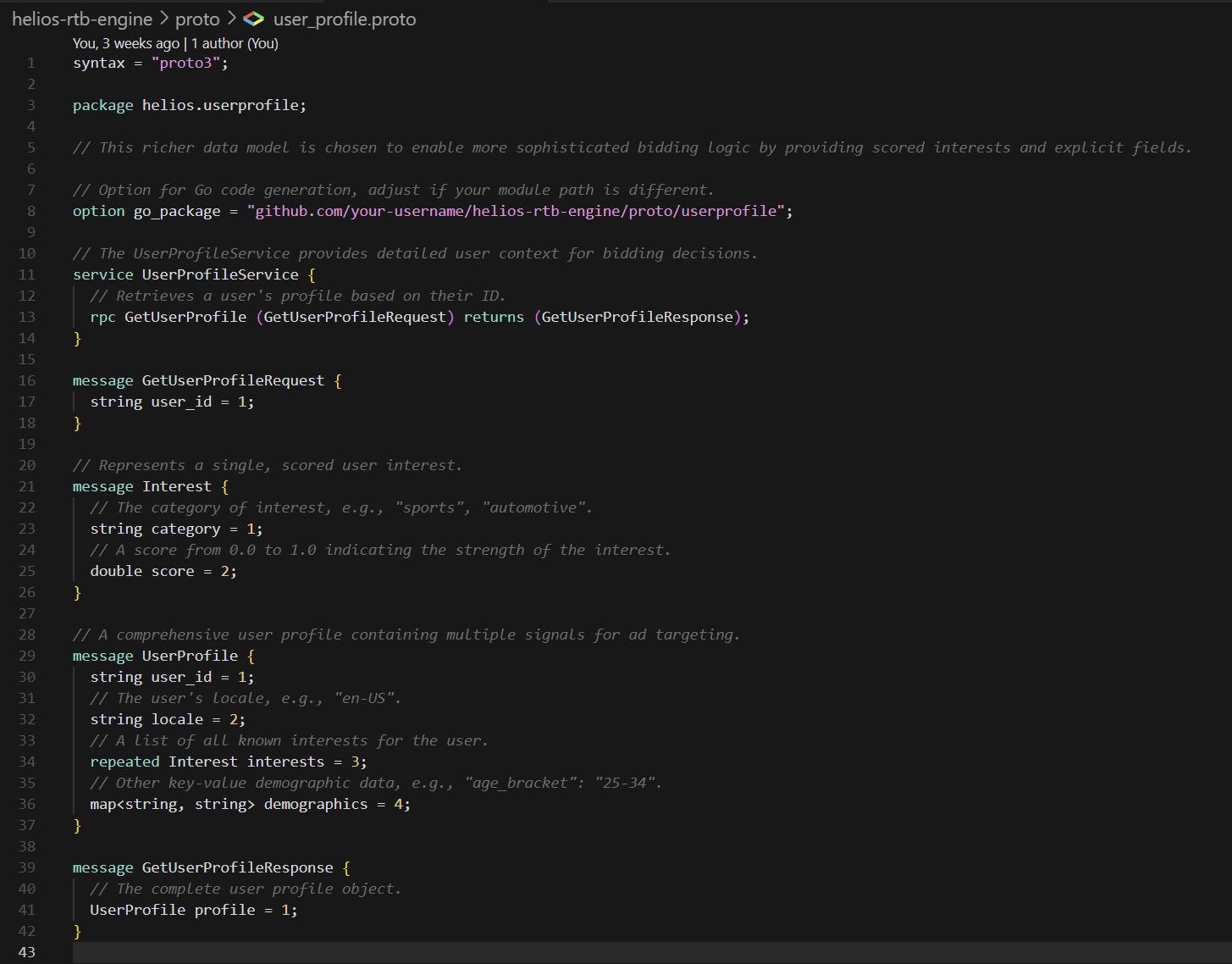
Real-Time Bidding System - Technical Report

##### Helios RTB Engine 11

Listing 2.3: User Profile Data Structure

17 }

##### gRPC Protocol Definition

****

Listing 2.4: user profile.proto (excerpt)

##### Why gRPC?

|  |  |
| --- | --- |
| **Advantage** | **Explanation** |
| Performance | HTTP/2 multiplexing and binary Protocol Buffers reduce  overhead |
| Type Safety | Strongly-typed contracts prevent integration errors |
| Code Generation | Auto-generate client/server code from .proto definitions |
| Efficiency | Compact binary serialization (smaller than JSON) |
| Streaming | Support for bidirectional streaming (future enhancement) |

Table 2.3: gRPC Advantages Over REST

### Bidding Logic Service (Python)

Helios RTB Engine

##### Overview

The Bidding Logic Service represents the core intelligence of the RTB system. It consumes bid requests from Kafka, enriches them with user profile data via gRPC calls, executes bidding algorithms, and publishes bid responses.

##### Technical Specifications

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| Language | Python 3.11+ |
| Framework | FastAPI (for metrics endpoint) |
| Kafka Library | kafka-python |
| gRPC Library | grpcio |
| Metrics Port | 8001 |
| Processing Mode | Asynchronous consumer |

Table 2.4: Bidding Logic Service Specifications

##### Bidding Algorithm

The service implements an interest-based bidding algorithm:

Listing 2.5: Bidding Algorithm Logic

1 def calculate\_bid\_price ( user\_interests: dict) -> float:

2 """

3 Calculate bid price based on user interest scores.

4

5 Interest scores range from 0.0 to 1.0

6 Returns bid price in USD

7 """

8 max\_interest = max( user\_interests. values ())

9 if user\_interests else 0.0

10

11 if max\_interest >= 0.9:

12 return 1.20 # High interest - premium bid

13 elif max\_interest >= 0.7:

14 return 0.85 # Medium interest - standard bid

15 elif max\_interest >= 0.5:

16

return 0.60

else :

return 0.35

# Low interest - minimum bid

17

18

# Fallback bid

##### Processing Flow

* + - 1. Consume message from bid requests Kafka topic

##### Helios RTB Engine 13

* + - 1. Extract user id from bid request
      2. Make gRPC call to User Profile Service: GetUserProfile(user id)
      3. Receive user interest scores
      4. Execute bidding algorithm to determine price
      5. Construct bid response with enrichment metadata
      6. Publish to bid responses Kafka topic
      7. Emit processing metrics

##### Bid Response Format

1 {

2 " bid\_request\_id ": " req -12345" ,

3 " user\_id ": " user -001" ,

4 " bid\_price ": 1.20 ,

5 " currency ": " USD ",

6 " timestamp ": "2025 -10 -30 T12 :00:01 Z",

7 " enriched ": true ,

8 " user\_interests ": {

9 " technology ": 0.95 ,

10 " sports ": 0.70

11 },

12 " winning\_interest ": " technology "

13 }

Listing 2.6: Bid Response Schema

### Auction Simulator (Node.js)

##### Overview

The Auction Simulator emulates an ad exchange’s auction mechanism, determining win- ners based on bid prices, minimum thresholds, and probabilistic selection.

##### Technical Specifications

|  |  |
| --- | --- |
| lightgray **Attribute** | **Value** |
| Language | Node.js 20+ (JavaScript) |
| Kafka Library | kafkajs |
| Port | 9001 (health check) |
| Auction Model | Second-price auction with probability |

Table 2.5: Auction Simulator Specifications

Real-Time Bidding System - Technical Report

##### Helios RTB Engine 14

##### Auction Logic

1 def determine\_auction\_winner ( bid\_price : float) -> dict:

2 """

3 Simulate auction outcome .

4

5 Parameters:

6 - bid\_price : The bid amount in USD

7

8 Returns dictionary with win status and price

9 """

10 MINIMUM\_BID\_THRESHOLD = 0.30

11 WIN\_PROBABILITY = 0.70 # 70% chance of winning

12

13 if bid\_price < MINIMUM\_BID\_THRESHOLD :

14 return {

15 " win\_status": False ,

16 " win\_price ": 0.0 ,

17 " reason ": " Below minimum threshold "

18 }

19

20 # Simulate competitive auction

21 if random . random () < WIN\_PROBABILITY :

22 return {

23 " win\_status": True ,

24 " win\_price ": bid\_price ,

25 " reason ": " Won auction "

26 }

27 else :

28 return {

29 " win\_status": False ,

30 " win\_price ": 0.0 ,

31 " reason ": " Outbid by competitor"

32 }

Listing 2.7: Auction Decision Algorithm

##### Auction Outcome Format

1 {

2 " bid\_request\_id ": " req -12345" ,

3 " user\_id ": " user -001" ,

4 " bid\_price ": 1.20 ,

5 " currency ": " USD ",

6 " timestamp ": "2025 -10 -30 T12 :00:01 Z",

7 " enriched ": true ,

8 " user\_interests ": {

Real-Time Bidding System - Technical Report

##### Helios RTB Engine 15

9 " technology ": 0.95

10 },

11 " win\_status ": true ,

12 " win\_price ": 1.20 ,

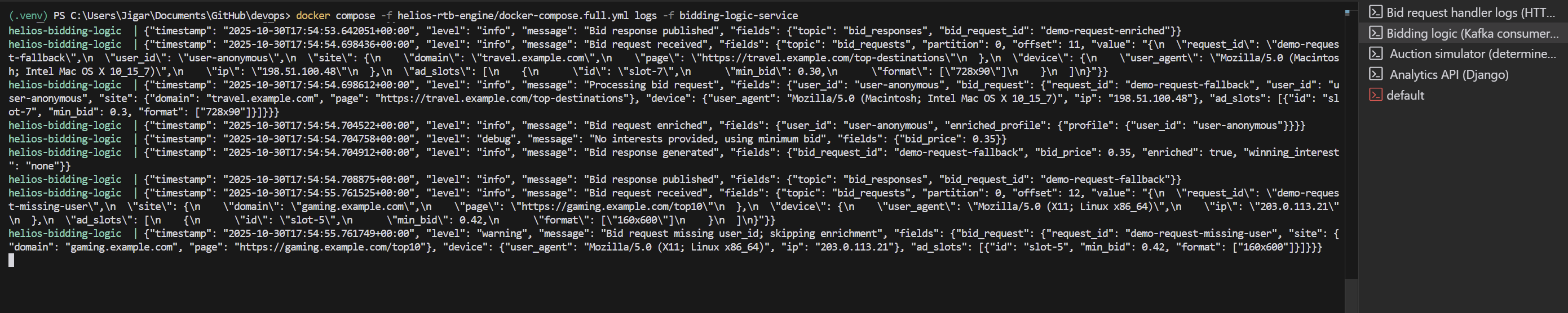
13 " auction\_timestamp ": "2025 -10 -30 T12 :00:02 Z"

14 }

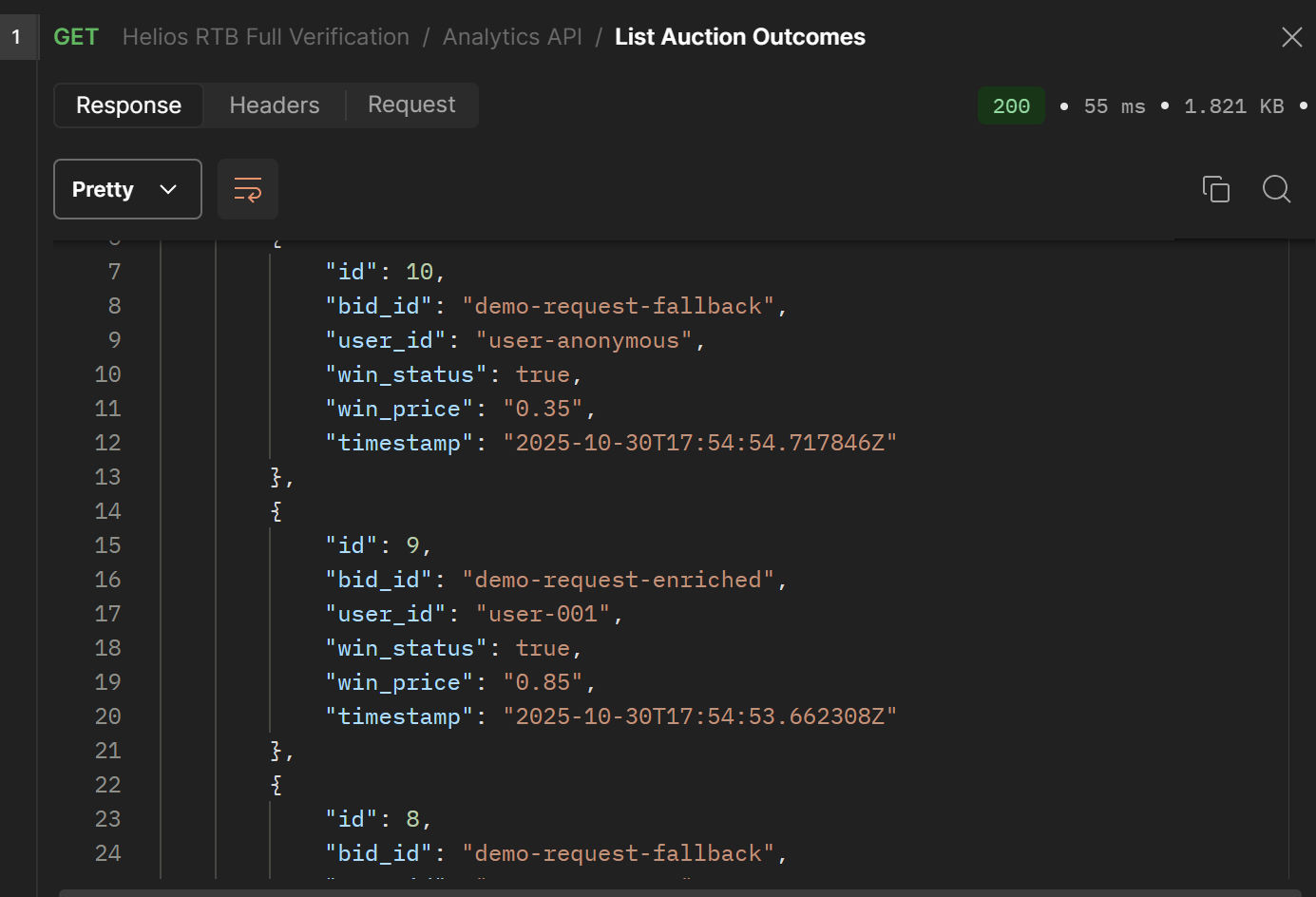
Listing 2.8: Auction Outcome Schema

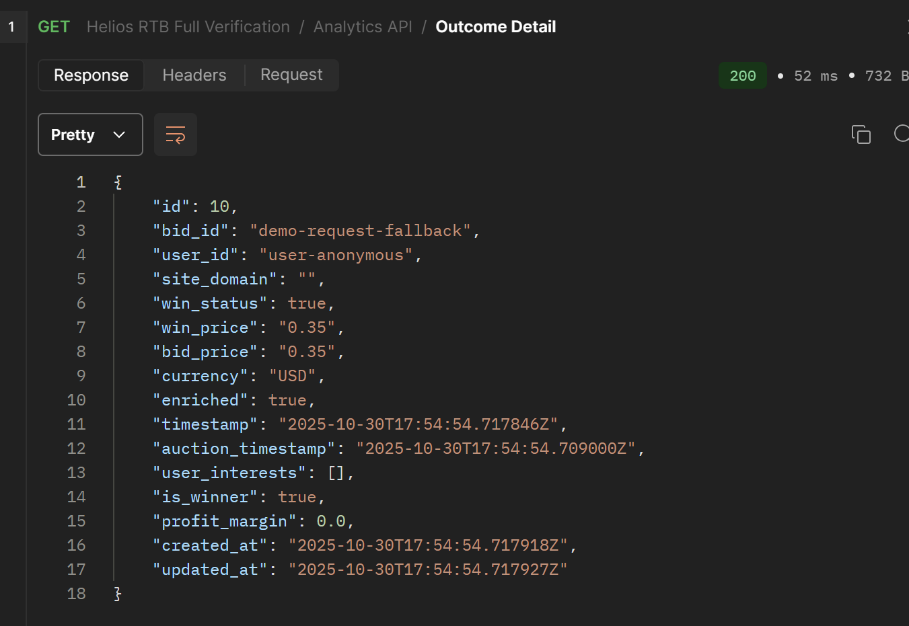
ScreenShots of some Results:

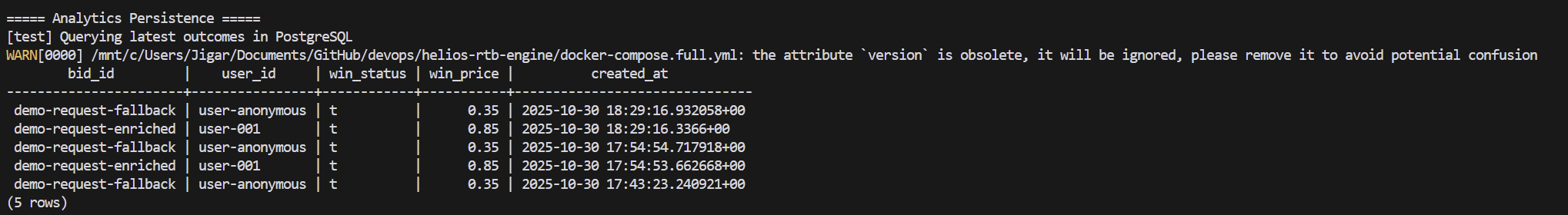
Bidding Logic Servvice in ACTION.

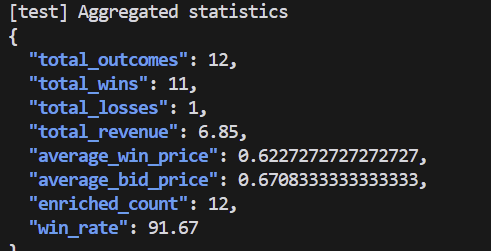


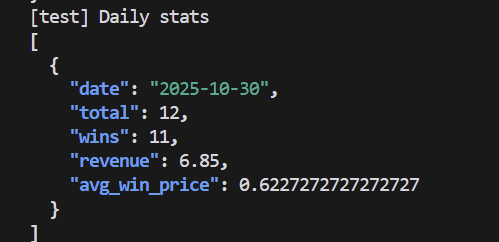
Analytics API:

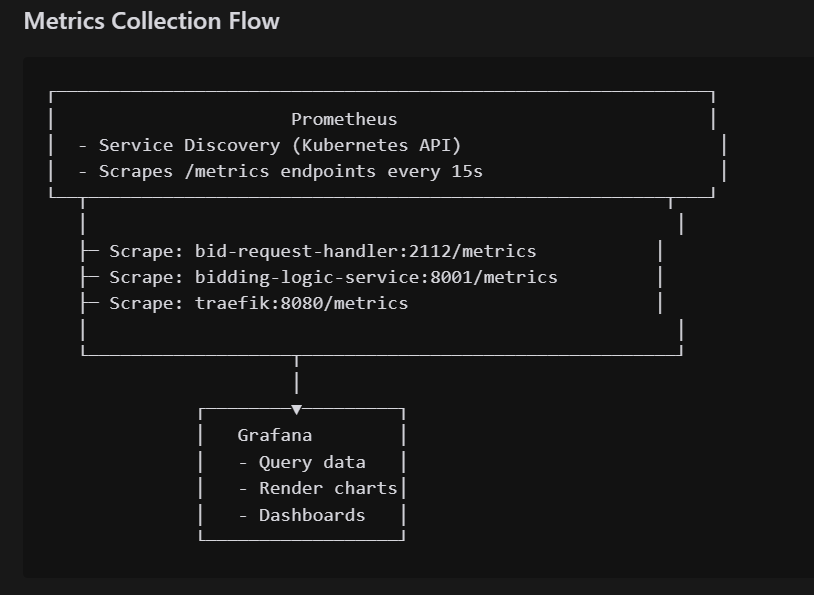
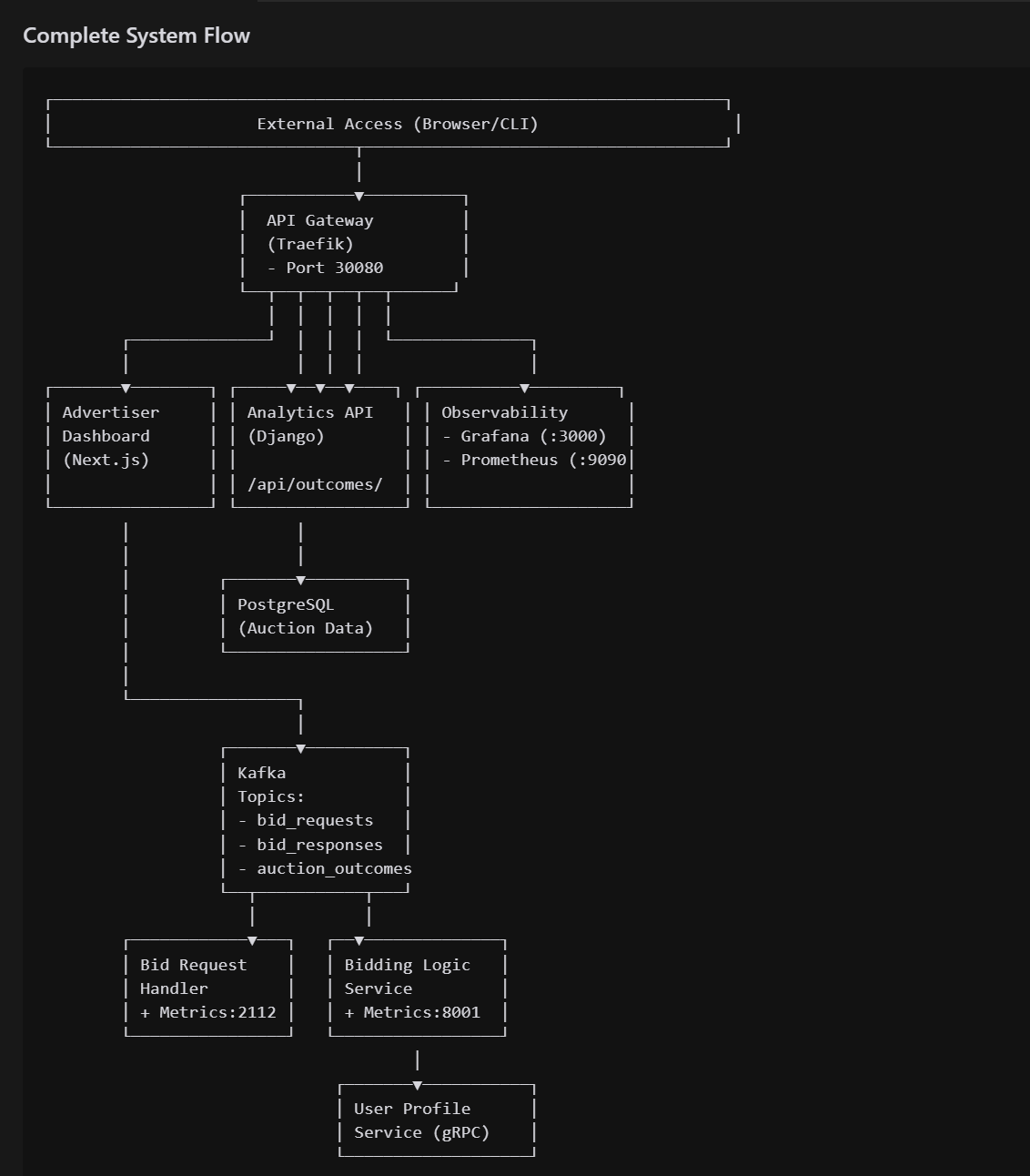


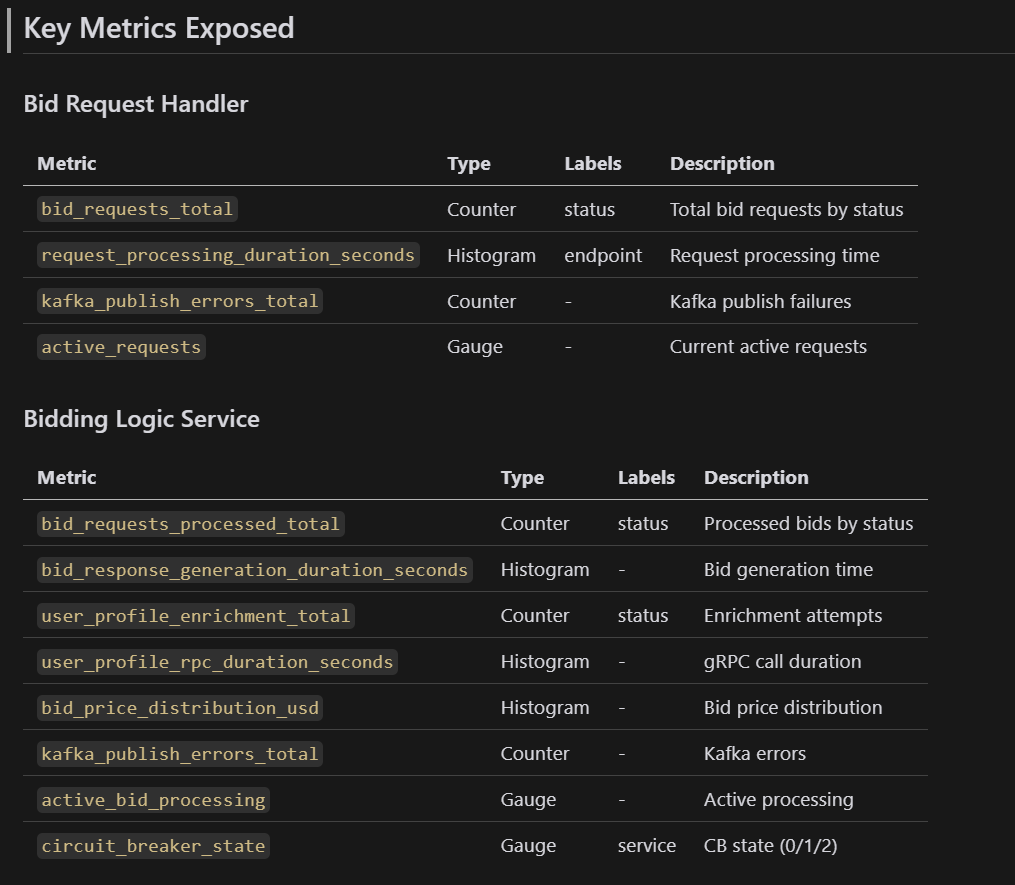
If Winner:











Advertisement Dashboard:

