Math Microservice Project

Alexandra-Elena Pitaru
Junior Data Engineer, Endava Cluj-Napoca

Mariana-Ionela Muntian
Junior Data Engineer, Endava Cluj-Napoca

July 22, 2025

Contents

Abstract

This document outlines the Math Microservice: a web service built with FastAPI that offers three operations—power, Fibonacci, and factorial—secured via JWT, monitored with Prometheus, and containerized using Docker. It includes an HTML frontend for basic interaction and automatic Swagger docs.

1 Overview

The microservice provides three mathematical endpoints accessible via REST API and protected by JSON Web Tokens (JWT). It logs operations to a local SQLite database and includes Prometheus integration to expose service metrics. A simple frontend allows users to log in and access operations visually.

2 Architecture

The service follows an MVC-like architecture:

- **Models**: SQLAlchemy models to log requests.
- **Controllers/Services**: Business logic for calculations.
- **Views**: FastAPI routes.

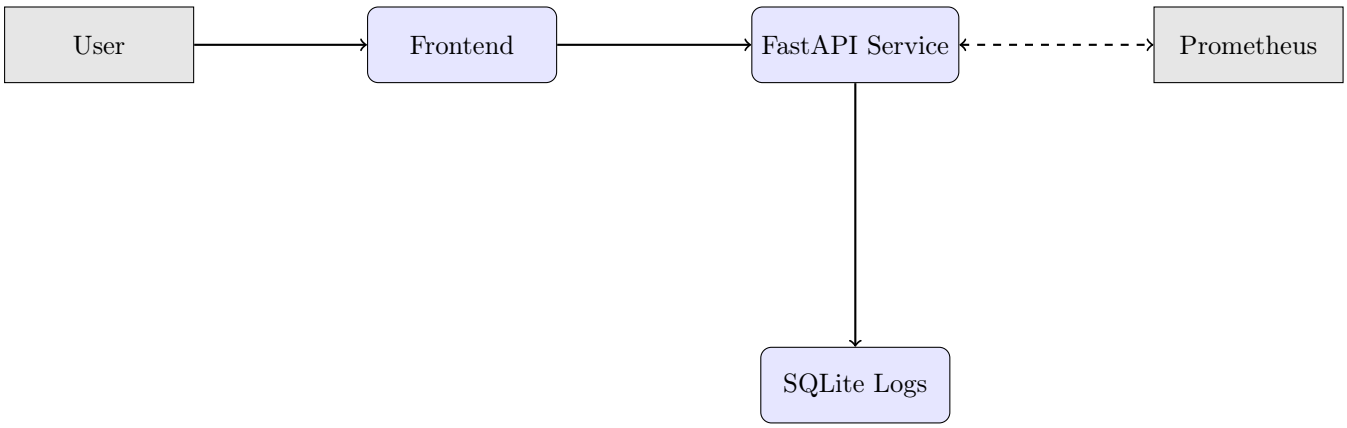

Figure 1: System Architecture

3 Operations and Code

3.1 Power

POST endpoint: `/api/v1/pow?x=2&y=3` returns 8.

3.2 Fibonacci

GET endpoint: `/api/v1/fibonacci/6` returns 8.

3.3 Factorial

GET endpoint: `/api/v1/factorial?n=5` returns 120.

```
from functools import lru_cache

def compute_pow(x: float, y: float) -> float:
    return x ** y

@lru_cache(maxsize=128)
def compute_fib(n: int) -> int:
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

@lru_cache(maxsize=128)
def compute_factorial(n: int) -> int:
    if n < 0:
        raise ValueError("n must be >= 0")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

Listing 1: Mathematical Service Functions

4 Authentication

The authentication mechanism relies on JSON Web Tokens (JWT). Users send a POST request to the `/login` endpoint with their credentials. If the credentials are valid, the server responds with a JWT.

This token includes user identification and an expiration time. It is signed using a secret key to ensure authenticity. Once received, the client must attach this token to the Authorization header in all subsequent API calls:

```
Authorization: Bearer <your_token_here>
```

The FastAPI backend validates this token for each protected endpoint. If the token is missing, invalid, or expired, the API responds with a 401 Unauthorized error.

Internally, authentication is implemented using FastAPI's OAuth2PasswordBearer, together with PyJWT for token encoding and decoding. The token validation is applied using FastAPI dependencies on secure routes, ensuring that only authenticated users can access them.

This system ensures stateless authentication, no server-side session storage is required, and the user's identity can be reliably used across the app.

5 Frontend

The frontend consists of two HTML pages: `login.html` and `math.html`, both served as static files through FastAPI's `StaticFiles` mechanism.

- **login.html**: Provides a basic login form where the user submits their username and password. Upon successful authentication, a JWT token is received and stored locally (e.g., in localStorage).

- **math.html**: After logging in, this page allows users to perform operations such as power, factorial, or Fibonacci by submitting values through a form. The JWT token is attached to each request header.

The frontend is lightweight and does not require JavaScript frameworks. Styling is handled via embedded CSS for simplicity. This makes the service easily testable by non-technical users and suitable for demonstration purposes.

6 Monitoring

The microservice exposes key runtime metrics on the `/metrics` endpoint using the Prometheus FastAPI Instrumentator library. These metrics include total HTTP request counts, response durations, and status code breakdowns.

Prometheus is configured to periodically scrape this endpoint. Once scraped, the data can be visualized using tools like Grafana, allowing developers and stakeholders to monitor the service in real time.

This setup enables basic observability, helping identify performance bottlenecks or failure patterns without significant performance overhead or complexity.

7 Dockerization

The app is fully containerized. Use:

```
docker build -t math-microservice .
docker run -p 8000:8000 math-microservice
```

Or with Compose:

```
docker-compose up --build
```

8 Conclusion

This microservice combines security (JWT), caching, logging, monitoring, and frontend delivery into a lightweight FastAPI-based solution. It is easy to deploy, observe, and use.