Project Synopsis for UIU CSE FEST Project Showcase 2025

Team District12

December 28, 2024

Optimizing Request Servicing for Limited Resource Systems

Introduction

We developed an optimized full-stack application for simultaneous high-traffic bookings, addressing potential database inconsistencies and reducing service time.

Scope

FastPass is designed for platforms where users concurrently compete for limited resources, such as tickets, or flash sales.

This solution ensures minimal database interaction, so -

- **users** experience minimal service delays,

- while **businesses** benefit from cost-effective, efficient, and scalable infrastructure for real-world heavy-loads.

Objectives and Motivation

Objectives: -

- Ensuring database consistency during high-demand booking attempts
- More scalable infrastructure
- Enhance performance using cache-level locking
- Faster, reliable server responses improving user experience

Motivation: -

Addressing the challenges of high-traffic platforms, such as-

- Database inconsistencies
- Delayed response times
- System bottlenecks
- Unsatisfactory user experiences

Project Features

- **Cache-Level Locking for High-Traffic Readiness:**

  - Locking mechanism at cache level protects database.
  - Reduces unnecessary database interactions.

- **Faster and Reliable Booking:**

    – Minimal service delays, regardless of successful or unsuccessful booking.

- **Optimized Resource Management:**
  - Caching for faster ticket availability updates.
- **Microservice Architecture:**
  - Independent microservices for scalability, leveraging diverse frameworks.

Methodology

Idea: We implemented booking services using cache instead of database operations, and promoted the locking mechanism for ensuring database consistency from database, to service level.

We used Redis to temporarily cache booked tickets. A ticket can only be booked if not found in cache. Canceled bookings are removed from cache, while confirmed purchases are recorded in database.

Responses for all bookings, cancellations, or purchases are served from cache level, avoiding direct database interaction.

These are processed within a lock ensuring conflict-free updates of booking information, with minimal logic inside the lock to reduce wait times.

The caching and locking logic are encapsulated in [an independent server](#).

For demonstration, we developed a full-stack train ticketing system with proper microservice and DevOps practices. Find a complete, documented implementation [here](#).

The auxiliary services manage [authentication](#), [business details](#), [ticket management](#) and [pricing](#), complementing the core program.

Technology Stack

Backend: NodeJS, ExpressJS, Django

Frontend: ReactJS

Database: PostgreSQL, Redis (caching and locking)

Containerization: Docker

Alongside Kubernetes, GitHub Actions, K6, Supertest, Prometheus and Grafana for DevOps.

Implementation

Phases: -

- Designed, created and populated database
- Developed auxiliary microservices
- Implemented booking and cancellation operations through ticket caching
- Applied lock at cache level
- Implemented containerization, deployment, automation, testing, monitoring

Limitations

The solution relies on Redis for caching. If Redis is temporarily unavailable, the system must fall back to slower atomic database interactions.

Conclusion

FastPass transforms high-traffic booking systems, with atomic and efficient server level transactions, improving user satisfaction.