CASE STUDY

Reengineering Legacy Services For A Leading Marketing Solutions Provider

thenaworks

Index

Client Background

The client is a global leader in digital solutions for destination marketing. With a mission to revolutionize the way travel brands engage with their target audiences, operating one of the world's largest travel ad networks, giving access to over 100 million travel shoppers monthly. Their direct publisher integrations circumvent ad exchanges and ensures destination-specific targeting and maximizes engagement rates across large-format ad units.

Facilitating over $1 billion in annual bookings for advertisers, the client leverages its expertise and market insights to introduce innovative solutions to destination marketers. This case study highlights one of the several projects within our ongoing partnership, showcasing how our collaborative efforts drive value and success.

Project Overview

Team Makeup

1 DevOps

1 Backend

2 Frontend

1 Data Engineer

Project Duration

10 month

Type of Service

Managed Solution

Tech Stack

The resulting Spring Boot template stack for our backend can be described as:

- Kotlin, Spring Boot, and Gradle
- Spring WebFlux for both HTTP server and client
- OpenApi for API contract (primarily serving React app)
- Spring Data R2DBC for Postgres integration
- Flyway for database migrations
- Kotest for both unit and integration tests
- WireMock for mocking/stubbing the REST services we had to consume
- Micrometer for observability
- Prometheus as a sink for observability metrics
- Grafana as metrics dashboard
- Docker as the container for our applications/service
- Kubernetes as our container orchestration runtime

Streamlining Marketing Campaign Management

With a vision to streamline their marketing campaign management process, our client sought to develop a new backend service, allowing customers to efficiently input campaign details such as budgets, duration, and target locations.

The client aimed to modernize their infrastructure by rewriting three legacy microservices using Vert.x, Kotlin, and Spring Boot technologies. Initially, our team was made up of two backend developers and three frontend developers for the service, with the frontend built using Typescript and React. To bolster our efforts, the client's platform team provided crucial support with AWS, Kubernetes, CI/CD, and security aspects throughout the project's duration.

Challenges

- **Adapting to Pre-Established Designs:** Our engagement commenced after the solution design phase had concluded, leaving us with the task of understanding pre-established designs and business domains to execute implementation effectively.

- **Absence of Internal Spring Boot Template:** The absence of an internal Spring Boot template necessitated the development of one concurrently with project delivery, adding complexity to our workflow.

- **Balancing Performance and Simplicity:** Lack of precise nonfunctional requirements, such as concurrent user support and latency thresholds, led to potential over-engineering. For instance, while utilizing Spring Reactor APIs for HTTP and Postgres integration enhanced performance, it introduced tradeoffs in code simplicity that were not initially anticipated.

Solution

The project spanned four months for the development of the service and six months for rewriting the three legacy Vert.x services. Adopting a lightweight Scrum approach, we conducted two or three daily meetings per week, held retrospective meetings post-sprint, and engaged in refining sessions prior to each sprint. Furthermore, our workflow included rigorous code reviews, ensuring the quality and coherence of our deliverables.

Since we already had a design to follow, we implemented it using a modern stack. We had to develop an internal Spring Boot template leveraging the libraries and tools we needed for the final solution.

Implementation

Despite leveraging an OpenAPI specification, our implementation journey often required iterative refactoring to enhance and address issues within our API. As collaboration between the front and backend teams progressed, certain features necessitated rewrites to meet evolving requirements and optimize performance.

While we possessed the capability to suggest alternative technology stacks, the predetermined stack by the client remained unchanged throughout the project's duration. However, for solutions characterized by:

- Minimal emphasis on brand or visual identity
- Primarily focused on data entry forms rather than web content
- Limited user base, typically ranging from a dozen to hundreds of users

Approach

We proposed an approach enabling seamless collaboration between frontend and backend developers within a unified codebase. This approach, exemplified by Vaadin Flow (OSS), offers several advantages based on our experience:

1. **Unified Development Environment:**
   - Developers can work on end-to-end feature implementation seamlessly, eliminating impedance mismatch and API specification friction.
   - Reduced rework, resulting in faster time to delivery and enhanced code maintainability.

2. **Enhanced User Experience:**
   - Vaadin Flow facilitates the creation of polished and professional user interfaces with customizable styling using plain CSS.
   - Support for Figma integration enables UX designers to craft professional-looking mockups, further enhancing UI aesthetics and usability.

3. **Operational Efficiency:**
   - Consolidated frontend and backend components within a single application reduce operational overhead, minimizing resource utilization and deployment complexity.

Vaadin Flow's seamless integration with Java or Kotlin, coupled with its compatibility with Spring Boot, ensures a smooth development experience. Additionally, Vaadin Hilla offers an alternative approach for frontend development using React, with simplified backend integration.

While Vaadin Flow was not explicitly proposed for the client, it has previously demonstrated its efficacy in critical applications, such as an electronic invoice product for Thomson Reuters Brazil over a decade ago. This proven track record underscores its suitability for projects with similar characteristics, emphasizing efficiency, usability, and scalability.

“In the face of pre-defined architectures and client expectations, our team at Athenaworks leveraged their collective expertise to meet the project goals, exceeding our client's requirements. This project wasn't just about meeting deadlines; it was a demonstration of how flexible and responsive software development could turn perceived limitations into launchpads for innovation.”

Results

Through diligent implementation and meticulous testing, we successfully delivered the service, characterized by the following key attributes:

Comprehensive Test Coverage: Utilizing high-level integration tests in a Behavior-Driven Development (BDD) style, we ensured that every conceivable use case scenario, from the simplest to the most complex, was thoroughly tested. This approach facilitated rapid onboarding for new maintainers, providing clear insights into the service's functionality even for non-technical stakeholders.

Exceeding Test Coverage Targets: Addressing the client's requirement for a minimum of 90% test coverage, our commitment to quality assurance resulted in consistently achieving test coverage levels approaching 100%, ensuring robustness and reliability of the service.

Actionable Insights through Monitoring: Operational and critical business metrics were seamlessly integrated with Prometheus and visualized through Grafana dashboards. This provided stakeholders with real-time insights into service performance, enabling informed decision-making and proactive issue resolution.

Stability and Resilience: Implementation of circuit breakers when interfacing with external services ensured the stability and resilience of our microservices architecture, safeguarding against service disruptions and enhancing overall reliability.

Following the successful development of the service, we leveraged our learnings to drive further impact:

Development of a Spring Boot Template Repository: Drawing from our experiences in developing the service, we established a reusable Spring Boot template repository. This repository served as a foundational framework, encapsulating best practices and facilitating streamlined development processes for subsequent projects.

Legacy Microservices Modernization: Utilizing our template repository, we embarked on the rewrite of three legacy Vert.x microservices into Spring Boot. Each rewrite was met with success, with notable improvements in latency and throughput. In certain instances, optimizations were so significant that operational costs were dramatically reduced, exemplified by the reduction from 8 Pods to only 2 in production environments.

Contact Us

athenaworks.com

athenaworks