# Case Study

## SPARKCHAT

IGNITE YOUR CONVERSATIONS. FAST, SIMPLE, AND SEAMLESS MESSAGING THAT CONNETS YOU INSTANTY!

ALPHAZ

Developed By Code Alphaz

# Team Code Alphaz

**Details of Team Members:**

| Name | Email | Contact Number |
|---|---|---|
| Nivindu Lakshitha | nivindulakshitha@gmail.com | 076 612 2495 |
| Nadun Daluwatta | nadundaluwatta26@gmail.com | 076 221 3088 |
| Dulara Abhiranda | abhiranda21@gmail.com | 076 022 3115 |
| Dulaj Hansana | dulajhansana1973@gmail.com | 077 893 7472 |

SparkChat

Ignite your conversations. Fast, simple, and seamless messaging that connects you instantly!

# Case Study: **Building a Scalable Real-Time Messaging Platform with Spark Chat Using React and Ballerina**

## 1. Introduction

Spark Chat is a real-time messaging platform designed to connect users seamlessly across devices. The goal of the project was to upgrade Spark Chat's infrastructure to support a rapidly growing user base, enhance real-time communication, and improve cross-device synchronization. To achieve this, we developed a modern front-end with **React** and used **Ballerina** for the back end and server-side architecture. Ballerina, a relatively new language, was chosen for its microservices capabilities and its powerful handling of API integrations.

### The Problem

With Spark Chat's expanding user base, the system faced several challenges:

- Scalability Issues: The platform's previous architecture could not handle large-scale user activity without message delays.
- Outdated Technology Stack: The backend struggled with real-time communication, and the lack of cross-device message synchronization led to user frustration.
- Limited Server Efficiency: The existing system was not optimized for handling API integrations and distributed systems efficiently.

### The Solution

We implemented a new architecture combining **React** for a robust, responsive front end and **Ballerina** for a highly scalable and efficient back end. Ballerina's integration-focused language capabilities made it an ideal choice for building a modern microservice architecture capable of handling the real-time requirements of Spark Chat.

## Technologies Used:

- **Frontend**: React with Next.js for building a fast, dynamic chat interface with real-time updates.
- **Backend**: Ballerina for building microservices that handle message delivery, user management, and API integrations.
- **Database**: MongoDB for storing chat histories and user data.
- **Real-time Communication**: WebSockets implemented in Ballerina for instant, scalable message delivery across devices.
- **Deployment**: Docker for containerization and deployment of the microservices, ensuring the system could scale horizontally.

## 2. Execution and Process

### A. **Frontend Development (React)**:

- We developed the front-end using React with Next.js to enable fast server-side rendering and client-side updates. The component-based structure allowed us to build a highly modular and scalable chat interface.

- Custom UI components for the chat list, message input, and chat window were designed to improve user engagement and responsiveness.

### B. **Backend Development with Ballerina**:

- **Ballerina** was chosen for its built-in support for microservices, API integration, and cloud-native development. We divided the backend into several microservices:
    - Chat Service: Handles real-time messaging using WebSockets for communication between the client and server.

    - User Service: Manages user authentication, profiles, and cross-device synchronization.

    - Notification Service: Sends real-time notifications and handles message delivery status.

- **API Gateway**: Ballerina's ability to handle APIs made it easy to integrate with external services such as push notifications and third-party analytics tools.

C. **Real-Time Messaging (WebSockets with Ballerina)**:

- We implemented WebSocket-based real-time messaging to allow instant delivery of messages and synchronization across devices. Ballerina's native support for WebSockets made it easy to build the real-time communication layer while ensuring efficient resource usage.

D. **Microservices Architecture**:

- Using Ballerina's strong support for microservices, we built an architecture where different services (chat, users, notifications) could scale independently. This ensured that as the user base grew, we could scale specific services as needed without overloading the system.

E. **Testing and Iteration**:

- Continuous integration and delivery (CI/CD) pipelines were set up using GitHub Actions and Docker to automate testing, building, and deployment.

## 3. Challenges Faced

- Learning Curve with Ballerina:

  As Ballerina is a relatively new language, our team initially faced a learning curve. However, Ballerina's clean syntax and extensive documentation helped us quickly adopt it for building efficient, scalable microservices.

- Message Synchronization Across Devices:

  Ensuring that messages were synchronized across devices in real time was a complex challenge. We overcame this by leveraging Ballerina's WebSocket support and its concurrency model, which allowed us to handle multiple client connections simultaneously with minimal latency.

- Performance Optimization:

  During testing, we encountered performance issues with database queries under high load. By implementing caching mechanisms and optimizing our MongoDB queries, we significantly reduced the query response times.

## 4. Results

The new Spark Chat system achieved the following key outcomes:

- Real-Time Synchronization: Messages are delivered instantly and synchronized across all connected devices, improving user satisfaction.

- User Engagement: User engagement increased by 30% post-launch, with users spending more time on the platform due to the improved messaging experience.

- Backend Efficiency: Ballerina's microservices architecture and API-handling capabilities significantly improved the performance of the backend, reducing server load and improving response times for API requests by 25%.

## 5. Lessons Learned

- Adopting Ballerina for Cloud-Native Development:

  Using Ballerina allowed us to efficiently manage microservices and APIs, making it an excellent choice for cloud-native applications. Its strong support for real-time communication and API integration made our backend much more flexible and scalable.

- Cross-Team Collaboration:

  The success of the project heavily relied on collaboration between the frontend (React) and backend (Ballerina) teams, ensuring that the two systems worked seamlessly together.

- Early Testing for Scalability:

  Early testing with load simulations helped us identify potential bottlenecks, allowing us to optimize the system for high concurrency from the start

## 6. Conclusion

By using React for the front end and Ballerina for the back end, we built a highly scalable, efficient real-time messaging platform for Spark Chat. The platform now offers fast, reliable communication with cross-device synchronization, supporting the growing user base. In the future, we plan to expand Spark Chat’s features to include AI-based chat moderation, message analytics, and deeper integration with third-party services.