Cloud Communication Software Architecture for Educational Institutions

Petar Stamenić[1], dr. Nemanja Radosavljević[2]

Abstract **-** This paper presents the design and implementation of a cloud-based communication software architecture, tailored for educational institutions. The system, RafBook-Backend, provides a unified platform for messaging, voice communication, user administration, and media sharing. Implemented in Java 17 using the Spring Boot framework, the solution integrates real-time communication technologies such as WebSocket, Firebase Cloud Messaging, and WebRTC for peer-to-peer audio channels. Data persistence is ensured with PostgreSQL, while Redis supports caching for performance optimization. The application is containerized with Docker, supported by GitHub Actions for CI/CD automation. System monitoring and logging rely on Prometheus, Grafana, and OpenSearch. The goal of this work is to provide a robust, scalable, and open-source communication tool that educational institutions can locally host and easily adapt to their specific needs.

Keywords - Caching, CI/CD, Cloud communication, Docker, Educational institutions, Grafana, Java, Open-source software, Prometheus, Redis, Spring Boot, WebRTC, WebSocket

I. INTRODUCTION

The purpose of this paper is to present the design and implementation of a communication platform tailored for educational institutions. Existing solutions, such as Discord, Slack, or combinations of Zoom and e-mail, have shown significant limitations when applied in an academic context. These limitations include subscription costs, restrictions on file sharing, message length, sound and video quality, as well as fragmented communication

[1]Petar Stamenić , Računarski fakultet, Belgrade, Serbia, pstamenic@raf.rs

[2]Nemanja Radosavljević, Računarski fakultet, Belgrade, Serbia, nradosavljevic@raf.rs

workflows.

The proposed system, RafBook-Backend, addresses these issues by offering an open-source, locally hosted platform that integrates real-time messaging, voice communication, user and role management, as well as notification mechanisms. The system is built with Java 17 and the Spring Boot framework, organized into functional services (user, messaging, orchestration, voice, file storage). PostgreSQL is used for database management, while Redis supports caching.

The goal of this paper is not only to describe the technical architecture and implementation details but also to highlight the significance of open-source communication systems for academic environments. By providing a scalable, adaptable, and cost-effective platform, this work contributes to the development of digital infrastructure in education.

II. RELATED WORK

Several communication platforms are widely used in academic and professional contexts, most notably Slack, Microsoft Teams, and Discord. While each of these platforms provides valuable functionality, they also introduce significant limitations when adapted for educational use.

A. Slack – team collaboration platform

Slack is primarily designed as a business-oriented collaboration tool. It integrates messaging, file sharing, and workflow automation. However, in its free version, Slack restricts message history and the number of integrations, which makes it unsuitable for long-term academic collaboration where persistent access to messages and resources is crucial.

B. Microsoft Teams – enterprise communication and collaboration platform

Microsoft Teams provides video conferencing, chat, and integration with Microsoft 365 services. It is commonly used in universities for online classes and meetings. Nevertheless, Teams often requires institutional licenses, which can be costly, and its complexity makes it less flexible for customization compared to open-source alternatives.

C. Discord – community communication platform

Discord is widely adopted among students due to its ease of use and community features. It supports text, voice, and video communication. However, in the free version, file sharing is limited to 5 MB, messages are capped at 2,000 characters, and screen sharing is restricted to 720p at 30 fps. These limitations significantly reduce its usability in academic settings where larger files, code snippets, and higher-quality streaming are often required.

D. Why RafBook-Backend?

In contrast to the above-mentioned tools, RafBook-Backend provides an open-source, locally hosted platform under the MIT license. This approach eliminates subscription costs, removes functional restrictions, and allows educational institutions to fully control their communication infrastructure. The system can be customized for specific needs, ensuring scalability and sustainability without reliance on third-party commercial services.

III. SYSTEM ARCHITECTURE

he RafBook-Backend system was designed as a monolithic Spring Boot application organized within a single Maven module, but logically divided into functional services. These include messaging, orchestration, user management, voice communication, and file storage. Each service contains dedicated layers for configuration, controllers, data transfer objects (DTOs), exceptions, mappers, models, repositories, and service classes. PostgreSQL is used as the main relational database, while Redis[8][9] supports caching and session management.

A. Choice of programming language – Java 17

Java[1] was chosen as the primary development language due to its maturity, portability, and strong ecosystem. The adoption of Java 17, a long-term support (LTS) release, provides access to modern language features such as lambda expressions, records, and sealed classes. These improvements simplify code readability and maintainability, while LTS support ensures stability for production environments.

B. Build automation – Maven

Maven was selected as the build automation tool. It provides a standardized project structure, dependency management, and integration with testing and

reporting frameworks. In comparison, Gradle offers more flexibility and performance in some cases, but Maven's convention-over-configuration model and extensive adoption in the Java ecosystem make it a reliable choice for educational and collaborative projects.

C. Application framework – Spring Boot

Spring Boot was chosen as the application framework because it simplifies the configuration of enterprise-grade Java applications. Unlike Java EE, which requires extensive boilerplate and external configuration, Spring Boot provides auto-configuration, embedded servers, and integration with testing frameworks. This accelerates development and reduces complexity, enabling faster prototyping and deployment.

IV. INFRASTRUCTURE

The system is containerized using Docker, which ensures reproducibility, portability, and isolation of services. A separate file service is deployed as an independent container, accessible regardless of the application state. This modular deployment facilitates local testing and production readiness.

A. Docker and containerization

Docker enables the packaging of the RafBook-Backend application into lightweight containers that can be run consistently across different environments. By using Docker Compose, infrastructure services such as PostgreSQL, Prometheus, Grafana, OpenSearch, and Filebeat can be orchestrated locally, forming a reproducible testing and development setup.

B. Kubernetes and Helm (planned)

Although not yet implemented, Kubernetes has been identified as the next step for scaling and managing the system. Kubernetes would allow automated deployment, service discovery, and load balancing, while Helm charts would simplify the configuration of complex deployments. This expansion would enable the system to serve larger institutions with higher availability requirements.

C. Monitoring and logging (Prometheus, Grafana, OpenSearch)[16][17][21]

Prometheus collects metrics from the backend via the Spring Boot Actuator endpoint, including request latency, error rates, and system load. Grafana visualizes these metrics through dashboards tailored to developers and administrators. OpenSearch, combined with Filebeat, provides centralized log management and enables querying for debugging and auditing. Together, these tools provide observability and proactive system maintenance.

D. Security aspects (JWT, RBAC, SSL/TLS)

System security is ensured through multiple layers. Authentication relies on JSON Web Tokens (JWT), which provide stateless, scalable session handling. Role-Based Access Control (RBAC) defines fine-grained permissions for students, professors, and administrators. All communication between clients and the server is secured via SSL/TLS encryption, while user credentials are hashed using the BCrypt algorithm.

V. CI/CD PROCESS AND CONTINUOUS DELIVERY

To accelerate development, testing, and deployment, the RafBook-Backend project implements a complete CI/CD[12] pipeline using GitHub Actions. The pipeline ensures automated builds, test execution, Docker image creation, containerized execution, and monitoring integration.

A. Workflow A: Integration, build, and integration tests

This workflow is triggered by a push or pull request to the main branch. It ensures that every change introduced to the repository is tested and verified before being merged.

Steps:

1. **Checkout repository** – fetches the source code.
2. **Set up JDK 17** – prepares the Java runtime environment using Temurin distribution.
3. **Cache Maven packages** – speeds up builds by reusing downloaded dependencies.
4. **Build with Maven** – executes mvn clean verify, running all unit tests.

5. **Generate JaCoCo report** – creates a code coverage report.
6. **Upload JaCoCo report** – stores coverage results as an artifact.
7. **Build Docker image** – packages the application into a Docker container.
8. **Run backend container** – starts the application on port 8080.
9. **Execute Newman tests** – runs Postman collections against the running backend, validating integration.
10. **Clean up containers** – stops and removes Docker containers after test execution.

This workflow guarantees that the application is continuously tested at both the unit and integration levels before deployment.

B. Workflow B: Manual deploy to remote server

This workflow is manually triggered to build and deploy a new Docker image to the remote production server. It accepts as input the list of services to restart, such as app, prometheus, grafana, or opensearch.

Steps:

1. **Checkout repository** – ensures the correct branch is used.
2. **Authenticate Docker Hub** – logs in using GitHub secrets.
3. **Generate image tag** – creates a unique tag based on timestamp.
4. **Build and push image** – builds and uploads the Docker image.
5. **Install sshpass** – prepares for secure remote access.
6. **Update docker-compose file** – replaces the application image tag.
7. **Upload to server** – copies the updated configuration.
8. **Deploy via SSH** – pulls the new image and restarts selected services.

This process ensures controlled, reproducible deployments with rollback possibilities through tagged Docker images.

C. Supporting tools

Maven is used for building and testing, while Docker and Docker Compose provide reproducible environments for application and infrastructure services. Filebeat is configured to forward logs to OpenSearch, and Prometheus scrapes metrics for Grafana visualization.

D. Security and rollback strategies

Secrets such as database credentials and Docker Hub tokens are managed securely through GitHub Secrets. Rollback is supported by tagging each Docker image with commit SHA or timestamp, allowing restoration to a previous stable version in case of failure.

VI. SYSTEM FUNCTIONALITIES

The RafBook-Backend platform implements a wide range of functionalities that combine user management, messaging, file sharing, and voice communication. These modules are organized logically into services to ensure clarity, maintainability, and scalability.

A. User management and roles

The system allows creation, update, and deletion of user accounts with role-based access control (RBAC). Roles include students, professors, and administrators, each with specific permissions. Authentication is performed through JSON Web Tokens (JWT), while user passwords are securely hashed with BCrypt.

Each student may belong to multiple study programs, receiving role tags that automatically grant access to relevant communication channels. Professors and administrators have extended permissions such as creating channels, managing roles, and sending announcements.

B. Communication channels and categories

Channels are structured hierarchically into categories that represent study programs and courses. For example, a course can include a *general* channel for discussion, an *archive* for documents, and specialized channels for assignments.

The system supports:

1. **CategoryService** – management of course categories.
2. **TextChannelService** – creation and moderation of text channels.
3. **MessageService** – sending, editing, and deleting messages.
4. **ReactionService** – reactions to messages for fast feedback.

5. **FileService** – file upload and sharing, supported by a standalone file service accessible independently of the backend.[22]

C. Voice communication and voice channels

Voice channels enable real-time audio communication between users. The service supports:

- Creation and management of dedicated voice channels.
- User caching with VoiceChannelCache for tracking participants in real time.
- WebSocket-based signaling for establishing sessions.
- WebRTC[3][4][6][15][20] peer-to-peer audio streaming, ensuring low latency and high-quality communication.

This architecture allows professors to host lectures or discussions in which students can request permission to speak, ensuring structured communication.

VII. Testing and quality assurance

Testing plays a crucial role in ensuring the reliability and stability of the RafBook-Backend platform. A multi-level testing strategy was adopted, covering unit, code coverage, and integration tests within the CI/CD pipeline.

A. Unit testing (JUnit 5)

JUnit 5 was used to implement unit tests for individual components and services. The framework provides annotations such as @Test, @BeforeEach, and @AfterEach, making it easier to structure tests and isolate functionalities. Unit testing guarantees that core business logic works correctly before integration with other services.

B. Code coverage (JaCoCo)

JaCoCo was integrated to measure how much of the codebase is exercised by the test suite. The tool produces detailed reports that highlight which classes, methods, and lines of code are covered by tests, and which remain untested. This helps identify weak points in the test strategy and ensures a higher degree of system reliability.

C. Integration testing (Newman/Postman)

Integration tests were conducted using Newman, the command-line runner for Postman collections. Since most developers are already familiar with Postman, the learning curve is minimal. Newman executes predefined API test scenarios against the running backend, validating interactions between services such as user authentication, channel management, and messaging.

By combining unit tests, coverage analysis, and integration tests, the project achieves a robust testing process that ensures continuous quality within the CI/CD workflows.

VIII. Conclusion and future work

This paper presented the design and implementation of RafBook-Backend, a cloud communication platform tailored to the needs of educational institutions. The system integrates user and role management, messaging, file sharing, and real-time voice communication into a single, self-hosted solution. Its architecture, built with Java 17 and Spring Boot 3, emphasizes modularity, scalability, and security.

The implemented CI/CD pipeline ensures continuous integration, automated testing, and deployment through GitHub Actions and Docker, with observability provided by Prometheus, Grafana, and OpenSearch. Security is guaranteed through JWT authentication, RBAC authorization, and encrypted communication channels.

The original contribution of this work lies in offering an open-source, institution-controlled alternative to commercial communication platforms. The system can be easily adapted by each institution to its specific requirements without subscription costs or functional limitations.

Future work will focus on extending the platform with Kubernetes orchestration, Redis caching for performance optimization, and advanced audio/video features such as screen sharing. Additional enhancements include dynamic speaking permissions in voice channels, flexible cross-course access management, and an application/bot ecosystem to further support collaboration.

LITERATURA

[1] https://www.ebsco.com/research-starters/computer-science/sun-microsystems-introduces-java

[2] Merelo, J.J., Castillo, P.A., Mora, A.M. et al. Chatbots and messaging platforms in the classroom: An analysis from the teacher's perspective. Educ Inf Technol 29, 1903–1938 (2024). https://doi.org/10.1007/s10639-023-11703-x

[3] Mahmoud, H., Abozariba, R. A systematic review on WebRTC for potential applications and challenges beyond audio video streaming. Multimed Tools Appl 84, 2909–2946 (2025). https://doi.org/10.1007/s11042-024-20448-9

[4] H. Alvestrand, "Overview: Real Time Protocols for Browser-based Applications -- draft-ietf-rtcweb-overview-13," IETF Network Working Group, Nov. 2014.

[5] H. Silveria Sonego et al., "Use of Moodle as a tool for collaborative learning: a study focused on wiki," IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, vol. 9, no. 1, pp. 17--21, Jan. 2014.

[6] Feher, B., Sidi, L., Shabtai, A., & Puzis, R. (2016): The Security of WebRTC.

[7] Professional communication tools in Higher Education: a case study in implementing slack in the curriculum. - MENZIES, R. and ZARB, M.

[8] Privalov, M. V., & Stupina, M. V. (2024): Improving web-oriented systems using Redis caching.

[9] Redis vs. Memcached in Microservices Architectures: Caching Strategies Surbhi Kanthed

[10] Hwang K, Dongarra J, Fox G. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Morgan Kaufmann; 2013.

[11] Cervantes H, Kazman R. Designing Software Architectures: A Practical Approach. Addison-Wesley; 2016.

[12] Razvoj softvera orijentisanog na procese (DevOps mikroservisi kontejneri) - Милош Раденковић

[13] Baze podataka - Снежана Поповић

[14] https://docs.spring.io/spring-framework/reference/index.html

[15] https://webrtc.org/getting-started/overview

[16] https://grafana.com/docs/

[17] https://opensearch.org/docs/latest/

[18] Design, Implementation, and Evaluation of a Web-Based System for Alumni Data Collection - Danijel Mijic

[19] The Design and Implementation of the University Alumni Management System - Chong Yuan (Udutech Inc., Shijiazhuang, China), Xi Zhao (Udutech Inc., Shijiazhuang, China), and Ye Liu (Udutech Inc., Shijiazhuang, China)

[20] WebRTC technology overview and signaling solution design and implementation - Branislav Sredojev; Dragan Samardzija; Dragan Posarac

[21] Comprehensive Monitoring and Logging in Kubernetes: Best Practices and Tools - ANIRUDH MUSTYALA

[22] Secure Cloud Storage and File Sharing - IEEE