Hochschule Fulda
University of Applied Sciences

Distributed Applications (AI5109) Final Summary

Submitted to :
Prof Kritika Murugan

Submitted by :
Usama Tahir

Table of Contents

Introduction to Distributed Application Architectures 3

Architectures of Distributed Internet-based Applications 3

Middleware: Tasks, Goals, and Functions 4

Middleware Services and Architectural Concepts 4

Web Services, Microservices, and Service-Oriented Architecture (SOA) 5

Modern Programming Languages for Server-Side Development 6

Frameworks and Design Patterns for Distributed Applications 6

Application Servers 7

Topics Presented by Students
These were the following topics presented by the students in class 7

Final Project:
Project Overview 9

Features 10

- User Registration and Authentication: 10
- Real-Time Messaging 11
- Multimedia Sharing 11
- Message Deleting 12
- Message Editing 12
- Technologies Used 13
- System Architecture 13
- Setup and Configuration Firebase 13
- Implementation Details 14

Conclusion 14

References 14

Introduction to Distributed Application Architectures

In the course on Distributed Applications, we delved into the overarching concepts surrounding distributed application architectures, emphasizing their pivotal role in modern computing landscapes. We extensively explored the significance of distributed systems, recognizing them as fundamental pillars of contemporary computing infrastructures. Through in-depth discussions, we dissected the intricate mechanisms of scalability, fault tolerance, and performance considerations inherent to distributed systems. Understanding these crucial aspects is paramount for designing and implementing robust, resilient, and efficient distributed applications capable of meeting the demands of today's dynamic computing environments.

Architectures of Distributed Internet-based Applications

We thoroughly examined the diverse architectures employed in the development of Distributed Internet-based Applications. Our exploration began with the foundational Client-Server architecture, where we delved into its inherent client-server relationship, highlighting its role in facilitating efficient communication and data exchange. Additionally, we scrutinized the Peer-to-Peer architecture, dissecting its decentralized nature and its ability to foster direct communication between peers. Furthermore, we explored Hybrid Models, which amalgamate the strengths of both client-server and peer-to-peer architectures, offering versatility and scalability to suit varied application requirements. Through comprehensive case studies, we analyzed real-world applications deployed using different architectures, providing invaluable insights into their design considerations, performance characteristics, and practical implementations. These studies served as concrete examples illustrating the applicability and effectiveness of various distributed application architectures in addressing real-world challenges and scenarios
These were the following topics covered during this section of the course

- Client-server architecture.
- Peer-to-peer architecture.
- Hybrid models.

Middleware: Tasks, Goals, and Functions

In this course, we delved into the pivotal role of middleware within distributed systems, comprehensively exploring its significance in facilitating seamless communication and integration between disparate components. We dissected the overarching goals of middleware, emphasizing its paramount importance in interoperability, scalability, and reusability across distributed environments. Additionally, we scrutinized the fundamental functions of middleware, which revolve around abstracting complexities inherent in distributed systems, thereby shielding application developers from intricacies such as network protocols, data formats, and platform dependencies. Through practical examples we gained a holistic understanding of how middleware serves as a linchpin in orchestrating seamless interactions within distributed systems, ultimately facilitating the development and deployment of robust and efficient distributed applications.

These were the following topics covered during this section of the course

- Understanding the role of middleware in distributed systems.
- Tasks performed by middleware.
- Goals of middleware in facilitating communication and integration.
- Functions of middleware in abstracting complexities.

Middleware Services and Architectural Concepts

In our exploration of distributed applications, we delved into foundational components and paradigms essential for building robust and scalable distributed systems. We studied distributed object models, understanding how they facilitate seamless communication and interaction between distributed objects across a network. Messaging systems were examined as vital tools for asynchronous communication, with queuing mechanisms ensuring reliable message delivery and decoupling of components. Publish/subscribe paradigms provided flexible and scalable communication patterns, while peer-to-peer communication enabled direct interaction between distributed nodes without centralized servers. Through our comprehensive study of these paradigms, we gained insights into the diverse mechanisms underpinning distributed systems, laying the groundwork for designing and implementing efficient distributed applications.

These were the following topics covered during this section of the course

- Distributed object models.
- Messaging systems.
- Queuing mechanisms.
- Publish/subscribe paradigms.
- Peer-to-peer communication.

Web Services, Microservices, and Service-Oriented Architecture (SOA)

In this course we understand different architectural paradigms shaping modern computing landscapes. We began by immersing ourselves in the realm of web services, understanding their pivotal role in enabling interoperability and communication across diverse systems on the internet. Transitioning to microservices architecture, we delved into its principles emphasizing modularity, scalability, and resilience through the decomposition of applications into small, loosely coupled services. Simultaneously, we scrutinized Service-Oriented Architecture (SOA) and its significance in modern enterprise integration, elucidating its focus on reusable services to foster agility and interoperability. Through a comparative lens, we critically analyzed web services, microservices, and SOA, evaluating their respective strengths and weaknesses in different contexts, thereby equipping ourselves with invaluable insights for navigating the complexities of distributed system design in the future.

- Introduction to web services.
- Principles of microservices architecture.
- Understanding SOA and its relevance in modern enterprise integration.
- Comparative analysis of web services, microservices, and SOA.

Modern Programming Languages for Server-Side Development

In our exploration of programming languages commonly used in server-side development, with a focus on Java, Python, and Node.js our class predominantly utilized Java in lectures. Java emerged as a cornerstone language in our discussions, renowned for its versatility, robustness, and extensive ecosystem of libraries and frameworks tailored for distributed application development. Leveraging Java's platform independence and mature tooling, we delved into the intricacies of building scalable and reliable distributed systems. While we were granted the flexibility to explore alternative languages, Java's prominence in our class lectures underscored its relevance and applicability in real-world scenarios. Through hands-on exercises and practical demonstrations, we harnessed Java's strengths in distributed computing, while also acknowledging the diverse capabilities offered by other languages such as Python, Node.js, and Go. This comprehensive approach empowered us to make informed decisions regarding language selection based on project requirements and objectives, ensuring the successful implementation of distributed applications in diverse environments.

Frameworks and Design Patterns for Distributed Applications

In our exploration of server-side development for modern distributed applications, we delved into a myriad of programming languages tailored to meet the diverse needs of developers. We began by examining Java, a stalwart in server-side development, renowned for its robustness, scalability, and extensive ecosystem of libraries and frameworks. We then turned our attention to Python, celebrated for its versatility, succinct syntax, and prominence in web development and backend services. Additionally, we explored Node.js, leveraging its non-blocking I/O model and event-driven architecture for building highly responsive and scalable server-side applications. Lastly, we scrutinized Go, appreciated for its simplicity, concurrency support, and remarkable performance characteristics, making it an increasingly popular choice for server-side development in distributed systems. Through our comprehensive study of these modern programming languages, we gained insights into their unique strengths and applications, empowering us to make informed decisions in selecting the most suitable language for our server-side development endeavors.

Application Servers

In our exploration of frameworks and design patterns for distributed applications within the Java ecosystem, we delved into a rich array of tools and methodologies tailored to streamline development and enhance scalability. Our focus extended to frameworks such as Spring Boot, which offers comprehensive support for building microservices and distributed systems. Through practical demonstrations and hands-on exercises, we gained proficiency in leveraging Spring Boot's features such as dependency injection, aspect-oriented programming, and robust support for RESTful APIs. Additionally, we explored design patterns such as the Observer pattern, which facilitates loosely coupled communication between components in distributed architectures. By applying these patterns and frameworks in our projects, we honed our skills in architecting resilient and scalable distributed applications in Java, equipped with the knowledge to navigate the complexities of modern distributed computing environments effectively.

Topics Presented by Students

These were the following topics presented by the students in class

Web services: Standardized protocols for communication between different systems over the internet.
Transaction Processing: Managing transactions to ensure data integrity and consistency in distributed systems.
RPC (Remote Procedure Call): Mechanism for invoking procedures or functions on remote systems.Tasks, Goals, and Functions: Responsibilities and objectives of middleware in facilitating communication and integration.
Data Distribution: Distributing data across multiple nodes in a network.
Security: Ensuring confidentiality, integrity, and availability of data and resources in distributed systems.
Android Studio Setup and Installation: Installing and configuring Android Studio for Android application development.
Communication paradigms: Patterns and approaches for communication between distributed components.
Microservices: Architectural style where applications are composed of small, independently deployable services.

Information Systems: Systems for storing, managing, and retrieving information.
Container: Lightweight, portable environment for running applications and their dependencies.
Request/Reply: Communication pattern where a client sends a request and expects a response from a server.
SOA (Service-Oriented Architecture): Architectural style where services are organized and exposed for reuse.
Distributed Component: Modular software components distributed across a network.
Message-Based Interaction: Communication between distributed components using messages.
Messaging: Asynchronous communication between distributed systems via messages.
Distributed File Systems: File systems spanning multiple nodes in a network.
Orchestration: Coordination and management of distributed services to achieve a specific goal.
EAI (Enterprise Application Integration): Integrating disparate systems and applications within an enterprise.
Peer to peer: Decentralized communication model where nodes communicate directly with each other.
Application Server: Software framework for hosting and managing applications.
Multicast: Sending messages from one sender to multiple recipients simultaneously.
Layered: Architectural pattern where components are organized in layers.
Structured: Organized and well-defined approach to system design and development.
Fault Tolerance: System's ability to continue operating in the event of failures.
Queuing: Storing and managing requests in a queue to be processed sequentially.
Replication: Copying and distributing data across multiple nodes for redundancy and performance.
Serverless Computing: Cloud computing model where cloud providers manage infrastructure, allowing developers to focus on code.
Concurrency: Simultaneous execution of multiple tasks or processes.
Repository: Centralized storage for managing and versioning code, documents, or data.
Semi Centralized: Partially centralized system architecture where some components are centralized while others are distributed.
Distributed Mobile Computing: Computing tasks distributed across multiple mobile devices.
Parallelism: Simultaneous execution of multiple tasks to improve performance.
Client-Server: Architecture where clients request services from servers.
Object Based SOA: SOA approach where services are encapsulated as objects.
Load Balancing: Distributing incoming network traffic across multiple servers to optimize resource utilization.
Publish/Subscribe: Messaging pattern where publishers send messages to topics and subscribers receive messages based on their interests.

EJB (Enterprise JavaBeans): Component-based architecture for developing scalable, distributed applications.
Android Based Mobile Computing: Development of mobile applications using the Android platform.
Pipe and Filter: Design pattern where data flows through a series of processing components.
Resource Based: Accessing and manipulating resources via uniform resource identifiers (URIs).
Middleware: Software layer facilitating communication and interaction between distributed components.
Event Based: Triggering actions in response to events or messages.
MDB (Message-Driven Bean): EJB component that asynchronously processes messages.

Final Project

Project Overview

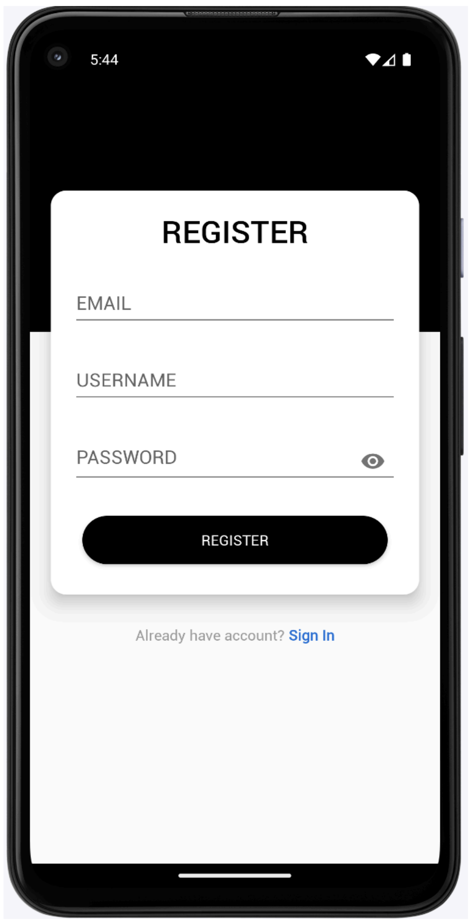
This Android Firebase Chat Application is a cutting-edge platform designed to facilitate seamless communication among users. Leveraging the robust capabilities of Android Studio and Firebase, we've created a feature-rich application that transcends traditional messaging paradigms. Here's a concise overview of its key features:

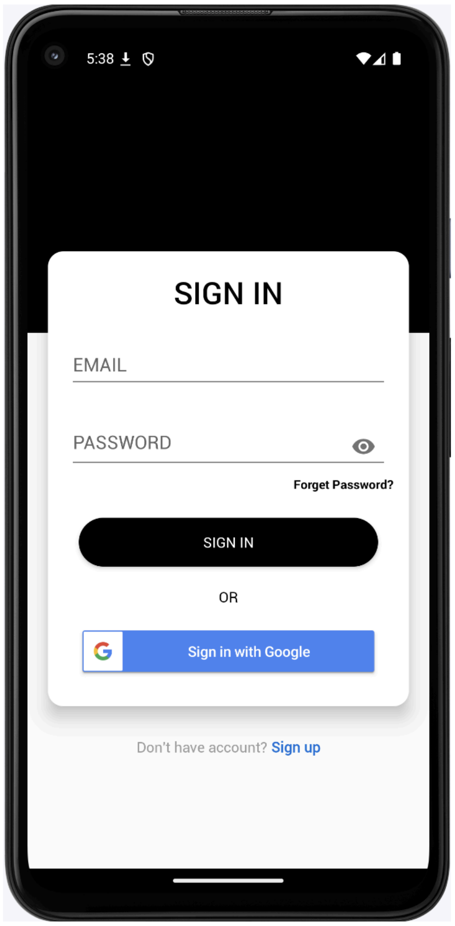
- **Real-Time Communication:** Users can engage in fluid conversations in real-time, enabling instantaneous interaction regardless of geographical locations.
- **Multimedia Sharing:** The application supports the sharing of images and videos, allowing users to express themselves vividly and enhance communication.
- **Authentication Services:** Robust authentication mechanisms provided by Firebase ensure secure user access, safeguarding sensitive information and maintaining user privacy.
- **User Profile Management:** Users have the flexibility to manage their profiles within the application, enabling personalization and customization of their experience.
- **Efficient Backend Services:** Firebase Realtime Database and Storage provide efficient data storage and retrieval, enabling smooth operation and scalability of the application.

Features

User Registration and Authentication:

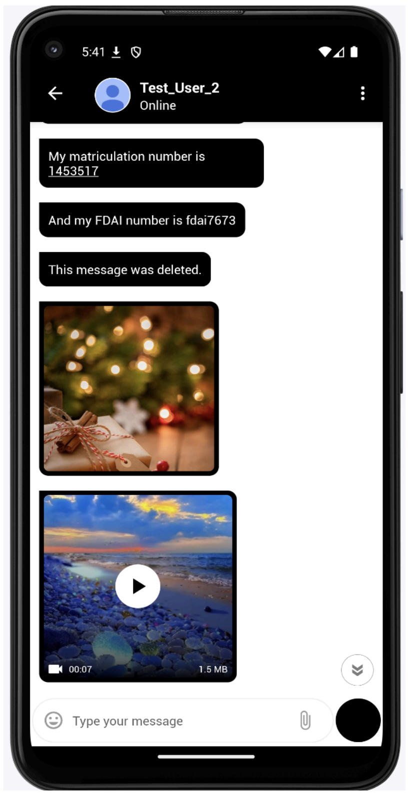
Our application offers a seamless user registration and authentication process, providing users with multiple options for creating and accessing their accounts. Users can register using their email and password, ensuring a straightforward and secure authentication method. Additionally, we integrate social media authentication options, such as Google and Facebook login, allowing users to conveniently access the application using their existing accounts. This versatility in authentication methods enhances user accessibility and convenience, catering to a wide range of preferences and user demographics.





Real-Time Messaging

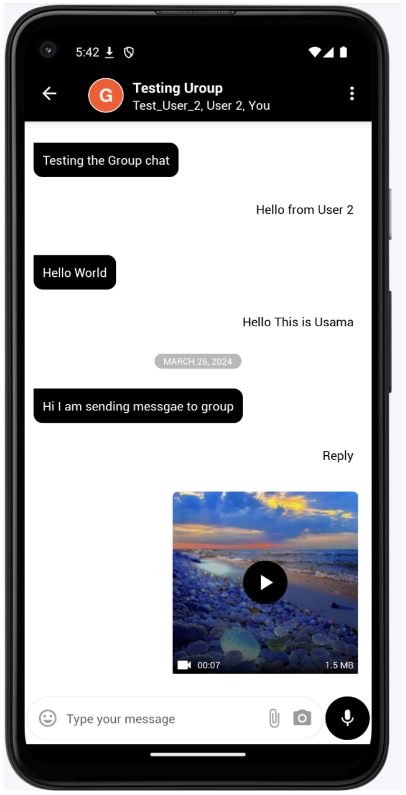
One of the core functionalities of our application is real-time messaging, enabling users to engage in fluid conversations with each other instantaneously. Leveraging Firebase Realtime Database, our messaging feature ensures that messages are delivered and displayed to users in real-time, eliminating delays and fostering seamless communication experiences. Whether users are exchanging text messages, sharing images, or sending videos, the real-time messaging feature ensures that conversations flow effortlessly, facilitating meaningful interactions and connections among users.



My matriculation number is 1453517

And my FDAI number is fdai7673

This message was deleted.



Testing the Group chat

Hello from User 2

Hello World

Hello This is Usama

Hi I am sending messgae to group

Reply

Multimedia Sharing

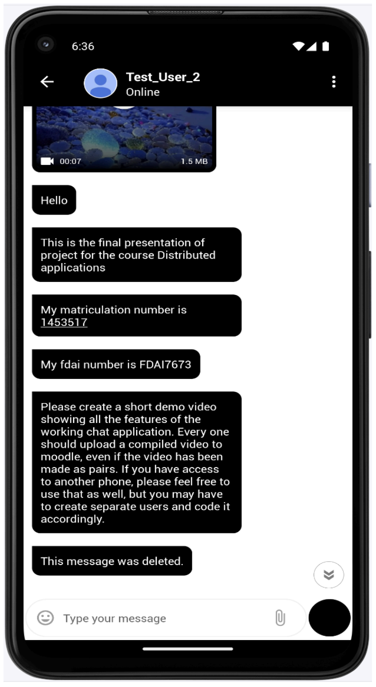
Our application empowers users to share multimedia content, including images and videos, with ease. With just a few taps, users can upload and share their favorite photos or videos with other users, enriching the communication experience and allowing for more expressive and engaging interactions. Leveraging Firebase Storage, multimedia files are securely stored and seamlessly retrieved, ensuring fast and reliable sharing experiences. Whether users are sharing memorable moments, creative expressions, or informative content, the multimedia sharing feature adds depth and vibrancy to their conversations, enhancing the overall user experience.

Message Deleting

Users can easily delete messages they have sent within a conversation thread. Whether it's correcting a mistake, retracting a message, or removing outdated information, the message deleting feature allows users to maintain the integrity of their conversations. Deleted messages are automatically removed from the conversation thread for all participants, ensuring consistency and clarity in communication. Leveraging Firebase Realtime Database, message deletions are synchronized in real-time across all devices, ensuring seamless and reliable message management experiences.

Message Editing

Our application also enables users to edit messages they have sent, providing an opportunity to refine their communication in real-time. Whether it's correcting typos, updating information, or adding additional context, the message editing feature allows users to fine-tune their messages without the need for deletion and re-typing. Edited messages are visually indicated within the conversation thread, providing transparency and clarity to all participants. Leveraging Firebase Realtime Database, message edits are seamlessly synchronized across all devices, ensuring consistency and coherence in communication experiences.



Hello

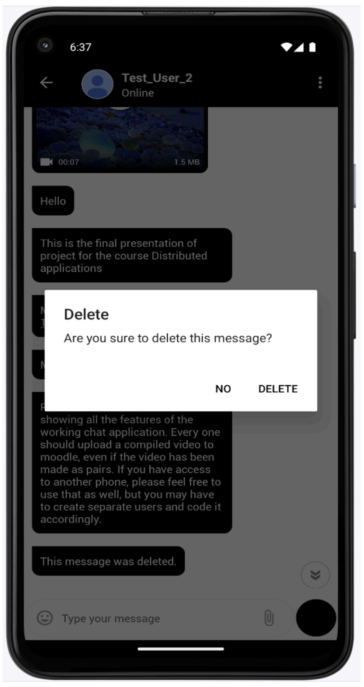
This is the final presentation of project for the course Distributed applications

My matriculation number is 1453517

My fdai number is FDAI7673

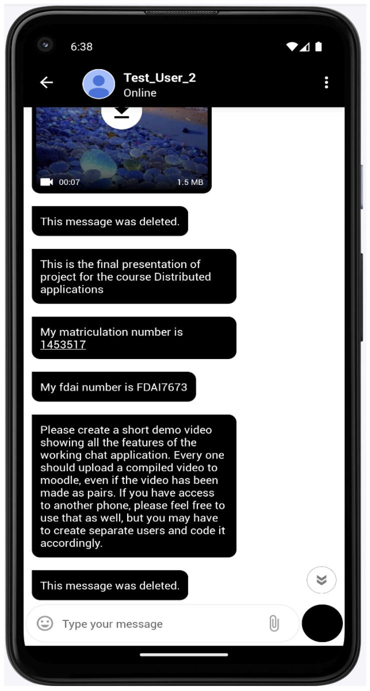
Please create a short demo video showing all the features of the working chat application. Every one should upload a compiled video to moodle, even if the video has been made as pairs. If you have access to another phone, please feel free to use that as well, but you may have to create separate users and code it accordingly.

This message was deleted.



Delete

Are you sure to delete this message?



This message was deleted.

This is the final presentation of project for the course Distributed applications

My matriculation number is 1453517

My fdai number is FDAI7673

Please create a short demo video showing all the features of the working chat application. Every one should upload a compiled video to moodle, even if the video has been made as pairs. If you have access to another phone, please feel free to use that as well, but you may have to create separate users and code it accordingly.

This message was deleted.

Technologies Used

- Android Studio: Integrated Development Environment (IDE) for Android app development.
- Firebase: Backend-as-a-Service (BaaS) platform provided by Google.
- Firebase Authentication: For user authentication.
- Firebase Realtime Database: Real-time NoSQL database for storing chat messages and other data.
- Firebase Storage: For storing images and videos.
- Kotlin: Programming language used for Android app development.

System Architecture

The system architecture comprises the Android client application and the Firebase backend services. The client application communicates with Firebase services for user authentication, real-time messaging, and multimedia storage. Firebase Realtime Database is used to store chat messages, while Firebase Storage is utilized for storing images and videos.

Setup and Configuration Firebase

To set up the project:

Step 1 : Install Android Studio and create a new project.

Step 2: Set up a Firebase project on the Firebase console

Step 3 : Add the Firebase configuration files to the Android project.

Step 4 : Configure Firebase Authentication, Realtime Database, and Storage services.

Step 5 : Implement the necessary authentication and database logic in the Android application.

Implementation Details

User Authentication:

Utilize Firebase Authentication SDK to implement user registration and login functionality.

Support various authentication methods such as email/password, Google Sign-In, and Facebook Login.

Real-Time Messaging:

Use Firebase Realtime Database to store and retrieve chat messages in real-time.

- Implement listeners to listen for changes in the database and update the UI accordingly.

Multimedia Sharing:

- Utilize Firebase Storage to store images and videos.
- Implement logic to upload and download multimedia files.

User Profile Management:

- Allow users to update their profile information using Firebase Authentication.
- Store additional user data in Firebase Realtime Database if necessary.

Conclusion

The real-time chat application developed using Android Studio and Firebase provides a seamless communication platform for users. It incorporates essential features such as user authentication, real-time messaging, multimedia sharing, and offline support, making it a versatile solution for modern communication needs.

References

You can find the source code of the firebase chat application on the following link.

https://github.com/krimuru9336/AI5109-Distributed-Applications/tree/fdai7673