Database Architecture and Chat Functionality in MimoApp

Kevin Estiven Lozano Duarte
20221020152
System Enginery Student
University Distrital Francisco Jose De Caldas

Introduction

MimoApp is an educational platform designed to enhance programming learning, featuring a real-time chat system. This chat facilitates interaction between students and tutors, promoting collaborative learning experiences. To ensure performance and scalability, an optimized database architecture has been implemented.

Objectives

- Analyze the database architecture of the chat in MimoApp.
- Explain the data flow and how real-time conversations are managed.
- Describe the security and scalability strategies implemented.

Importance of the Chat System

The real-time chat functionality enables:

- Seamless interaction among users, fostering collaborative learning.
- Effective communication between tutors and students for instant feedback.
- Motivation through real-time notifications and updates.

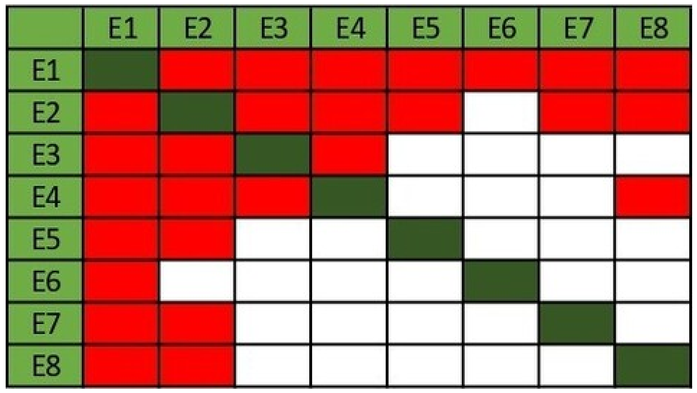

Figure: Database Relations and Architecture

Database Architecture

The database architecture for the chat in MimoApp is based on an SQL system optimized for message handling:

- **Relational Model:** Tables are normalized to avoid redundancies.
- **Indexes:** Composite indexes on time and conversation fields improve performance.
- **Table Partitioning:** Message tables are partitioned by date to manage large volumes of data.

Data Model

The database utilizes the following main tables:

- **Users:** Stores personal data, online status, and privacy settings.
- **Conversations:** Defines chat sessions and participants.
- **Messages:** Contains message content, read status, and timestamps.
- **Notifications:** Manages real-time alerts for new messages.

This design ensures referential integrity and facilitates scalability.

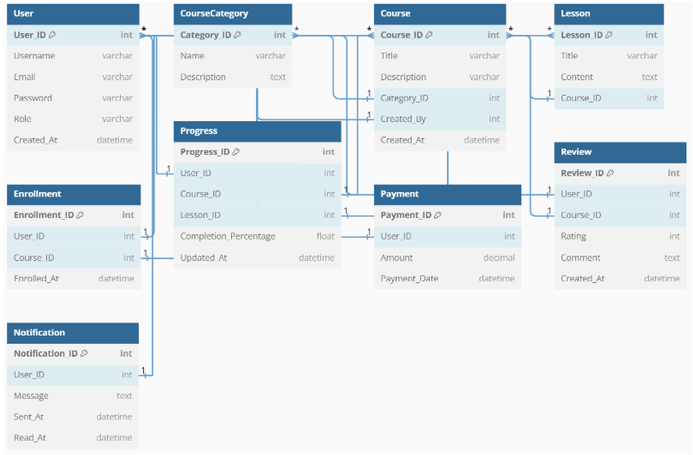

Figure: Chat Database Schema in MimoApp

Real-Time Chat Functionality

The chat architecture uses WebSockets to achieve real-time communication:

- **WebSockets:** Enable instant updates without page reloads.
- **State Synchronization:** Real-time synchronization ensures consistency of read/unread messages.
- **Connection Handling:** Redis is used as an intermediary to manage active connections.
- **Guaranteed Delivery:** Messages are stored in the database only after successful delivery.

Security and Scalability

To ensure a secure and scalable experience, the following strategies are implemented:

- **Authentication and Authorization:** JWT tokens are used to validate user identity.
- **Encryption:** All messages are securely transmitted using SSL.
- **Spam Prevention:** Content filters and message frequency limits are applied.
- **Scalability:** Load balancing and table partitioning are implemented to handle traffic growth.

Conclusions and Future Work

The chat database in MimoApp is optimized for performance and scalability, while the WebSocket architecture ensures an efficient real-time experience.

- The integration of ACID transactions ensures data consistency.
- Redis enhances efficiency in state synchronization.
- Future exploration includes using NoSQL databases for ephemeral conversations.
- Plans to incorporate data analytics for monitoring interactions and improving user experience.

References

- M. Stonebraker, "SQL Databases vs. NoSQL Databases," Communications of the ACM, 2018.
- M. Reinders, "WebSocket Essentials for Real-Time Applications," IEEE Internet Computing, 2020.
- Redis Labs, "Redis Documentation," `https://redis.io/documentation`.
- Stripe, "Online Payment Processing," `https://stripe.com`.
- FastAPI, "Fast Web Framework for Building APIs with Python," `https://fastapi.tiangolo.com`.