**📝 Project Summary – Real-Time Chat Application (Spring Boot + JavaFX)**

**🎯 Objective**

The goal of this project was to build a **full-stack real-time chat application** where users can:

* Register and log in,
* Send and receive text messages instantly,
* Share media (images, videos, documents),
* Store conversations in a database for history retrieval,
* Access and display files directly in the chat interface.

**🏗️ Architecture Overview**

The application follows a **layered architecture** with clear separation of concerns:

* **Frontend:** JavaFX desktop client (UI for chat, login, file sharing).
* **Backend:** Spring Boot server handling REST APIs, WebSocket messaging, file upload, and persistence.
* **Database:** MySQL for storing users and message metadata.
* **File Storage:** Local uploads/ folder for storing media files.

**⚙️ Backend Components**

**1. Controllers**

* **AuthController** – handles user registration & login.
* **UserController** – manages user operations.
* **ChatController** – manages real-time chat:
  + @MessageMapping("/sendMessage") for WebSocket text/media messages.
  + REST API /messages/sendFile to upload + broadcast file messages.
* **MessageController** – fetches chat history (/messages/chat).
* **FileUploadController** – handles raw file uploads to uploads/.

**2. Service Layer**

* **UserService / Impl** – handles user logic (registration, lookup).
* **MessageService / Impl** – processes messages:
  + Text → stored in DB as content.
  + Media → stores fileUrl + fileType (points to file in /uploads/).

**3. Repositories**

* **UserRepository** – manages User entity persistence.
* **MessageRepository** – manages Message entity persistence.

**4. Entities**

* **User** – contains username, password, and other details.
* **Message** – contains sender, receiver, timestamp, and either:
  + content (for text), or
  + fileUrl + fileType (for media).

**🗄️ Data Storage**

**📌 Database (MySQL)**

* Stores only **metadata**:
  + Text content
  + File URL (path to uploaded file)
  + File type (e.g., image/png, video/mp4)
  + Sender, Receiver, Timestamp

**📌 File System**

* All uploaded files stored in uploads/ directory.
* Files served via Spring’s static resource mapping (/files/\*\* → uploads/).
* Example:
  + Disk: uploads/uuid\_photo.png
  + Accessible: http://localhost:8080/files/uuid\_photo.png

**📡 Real-Time Communication (WebSocket)**

* Clients subscribe to /topic/messages.
* When a message is sent:
  + Controller passes it to MessageService.
  + Message is persisted in DB.
  + DTO is broadcasted via WebSocket → instant update for all clients.

**🎨 Frontend (JavaFX Client)**

* Login/Registration UI → connects with Auth APIs.
* Chat Window:
  + Sends text & media (through WebSocket or REST).
  + Subscribes to /topic/messages for real-time updates.
  + Renders:
    - **Text messages** in chat bubbles.
    - **Images** in ImageView.
    - **Videos** via MediaPlayer.
    - **Other files** as clickable download links.
* Chat history retrieved on load (/messages/chat).

**🧪 Testing & Debugging (Highlights)**

* **Day 29:** Fixed a bug where media messages were ignored:
  + ChatController.sendMessage(...) was only passing content, dropping file info.
  + Updated MessageServiceImpl to handle both text + media.
  + Result: media now appears in chat and is saved in DB.
* **Day 30:** Final testing:
  + Verified authentication, text, and media flows.
  + Ensured WebSocket broadcasts are instant.
  + Confirmed DB rows correctly record metadata.
  + JavaFX correctly renders all media types.

**📊 Flow of Operations**

1. **User registers/logs in** (AuthController + UserRepository).
2. **User sends a message**:
   * If text → passed via WebSocket → persisted in DB → broadcast to others.
   * If media → file uploaded → fileUrl/fileType stored in DB → broadcast to others.
3. **Other clients receive the message** via WebSocket broker.
4. **Clients render** either text, media preview, or a file link.
5. **Chat history** available via REST endpoint for later retrieval.

**✨ Final Status**

* **All features complete**: login, chat, media sharing, persistence, real-time updates.
* **Stable architecture**: Spring Boot backend, JavaFX frontend, MySQL + file system storage.
* **Scalable design**: easy to move uploads to cloud storage (e.g., S3) later.