📘 Mini Chat Application – Full Development Documentation

1. Project Overview

📌 Project Title

Mini Secure Chat Application

📌 Project Description

A full-stack chat application that allows authenticated users to send and receive messages securely using JWT authentication.

2. Objectives

- Implement secure user authentication
- Build message communication system
- Learn full-stack development
- Understand JWT security architecture
- Implement REST-based chat system

3. Technology Stack

Backend

- Spring Boot
- Spring Security
- JWT Authentication
- Spring Data JPA
- MySQL Database

Frontend

- HTML
- CSS
- JavaScript (Vanilla JS)

Tools

- Postman
- VS Code / IntelliJ
- MySQL Workbench

4. System Architecture

```
Frontend → REST API → Spring Boot → Database
```

Authentication Flow:

```
Login → JWT Token → Secure API Calls
```

5. Database Design

User Table

Field	Type
id	Long
username	String
email	String
password	String

Message Table

Field	Type
id	Long
content	String
sender	User
timestamp	DateTime

6. Backend Development Steps

Step 1 – Create Spring Boot Project

Add Dependencies:

- Spring Web
- Spring Security
- Spring Data JPA
- MySQL Driver
- Lombok
- Validation

Step 2 – Configure Database

application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/chat_app
spring.datasource.username=root
spring.datasource.password=6208gct

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Step 3 – Create Entity Classes

User Entity

```
@Entity
public class User
```

Contains:

- username
- email
- password

Message Entity

```
@Entity
public class Message
```

Contains:

- content
- sender
- timestamp

Step 4 – Create Repository Layer

UserRepository:

```
Optional<User> findByEmail(String email);
```

MessageRepository:

```
JpaRepository<Message, Long>
```

Step 5 – Password Encryption

Create PasswordConfig:

```
BCryptPasswordEncoder
```

Purpose:
Secure password storage.

Step 6 – Authentication Service

Functions:

- Signup user
- Login user
- Encrypt password
- Generate JWT token

7. JWT Authentication Implementation

JwtUtil Class Responsibilities

generateToken()

Creates JWT token during login.

Stores:

- user email
- issue time
- expiry time

extractEmail()

Reads email from token during request validation.

8. Security Configuration

SecurityConfig Class

Responsibilities:

- Disable CSRF
- Allow public APIs (/auth)
- Protect other APIs
- Register JWT filter
- Configure CORS

9. JWT Filter Implementation

Purpose

Intercepts every request and validates JWT token.

Filter Flow

```
Receive Request
    ↓
Check Public Endpoint
    ↓
Extract Token
    ↓
Validate Token
    ↓
Load User Details
    ↓
Set Authentication Context
    ↓
Allow Controller Execution
```

10. API Development

Authentication APIs

Signup API

```
POST /auth/signup
```

Registers new user.

Login API

```
POST /auth/login
```

Returns JWT token.

Message APIs

Send Message

```
POST /messages
```

Stores message in database.

Get Messages

```
GET /messages
```

Returns message history.

11. Frontend Development Steps

Step 1 – Create Project Structure

```
chat-app/
 ├ signup.html
 ├ login.html
 ├ chat.html
 ├ css/
 └ js/
```

Step 2 – Signup Page

Collect:

- username
- email
- password

Calls:

```
POST /auth/signup
```

Step 3 – Login Page

Collect:

- email
- password

Stores:

```
JWT token in localStorage
```

Step 4 – Chat Page

Features:

- Display messages
- Send messages
- Logout option

12. Frontend API Integration

Send Token In Request

```
Authorization: Bearer TOKEN
```

Load Messages

Fetch messages every few seconds to simulate real-time chat.

13. UI Enhancements

- Message bubble design
- Sidebar user list
- Auto scroll
- Responsive layout

14. Testing Strategy

Backend Testing

Use Postman to verify:

- Signup
- Login
- Message sending
- Unauthorized access handling

Frontend Testing

Verify:

- Token storage

- UI updates
- Message refresh

15. Security Considerations

- Password hashing
- Token expiry
- Unauthorized access blocking
- CORS protection

16. Future Enhancements

- Private messaging
- WebSocket real-time chat
- File sharing
- Online user tracking
- Role-based access
- Refresh token support

17. Deployment Suggestions

- Backend → Docker + Cloud hosting
- Database → Cloud MySQL
- Frontend → Netlify / Vercel

18. Learning Outcomes

Students will learn:

- REST API development
- Authentication systems
- Spring Security architecture
- Database integration
- Full-stack communication
- UI/UX implementation

⭐ Conclusion

This project demonstrates complete full-stack secure chat application development and introduces real-world authentication architecture used in modern applications.