



Hi, I'm DeepSeek. How can I help you today?



Design a spring boot application for chatbot chat messaging app the exact requirement is given as You are tasked with building a backend microservice to store chat histories generated by a RAG (Retrieval-Augmented Generation) based chatbot system.

Each conversation between a user and an AI assistant should be securely saved along with any relevant retrieved context.

Objective

Build a production-ready backend microservice that:

- Stores and manages chat sessions and chat messages.
- Supports session management features like rename, mark as favorite, and delete.
- Provides APIs to create sessions, add messages, retrieve messages, and manage sessions.
- Follows best practices for scalability, security, error handling, and configuration management.

Requirements

Core Functionalities

- Start and maintain chat sessions for users.
- Save messages within a session, recording the sender, content, and optional context.
- Enable renaming of chat sessions.
- Allow sessions to be marked or unmarked as favorite.
- Allow deletion of a chat session and its associated messages.
- Provide a way to retrieve the message history of a chat session.

Technical Expectations

- You are free to choose any backend language, framework, and database.
 - Environment-specific configurations should be properly managed (e.g., through .env files).
 - All APIs should be protected using API key authentication (read from environment variables).
 - Implement rate limiting to prevent abuse of the APIs.
 - Add centralized logging and global error handling.
 - The project must be Dockerized for local setup.
-
- Provide a README with setup instructions.

Deliverables

- Source code in a GitHub repository.
- .env.example file documenting all environment variables used.
- README.md including:
 - Setup and running instructions.

- o A description of the available APIs.
- o Optional: Include API documentation (e.g., Swagger) if implemented.

Bonus

- Add health check endpoints.
- Provide Swagger/OpenAPI documentation.
- Dockerize a database management tool (e.g., pgAdmin, Adminer) for easier browsing.
- Write basic unit tests for your services or business logic.
- Implement CORS configuration for security.
- Add pagination support when retrieving chat messages.



Here's a comprehensive guideline on how to create a Spring Boot application for a chatbot messaging app according to the requirements you've provided. The application will be designed as a backend microservice to store chat histories generated by a chatbot system.

Project Setup

1. Spring Boot Initialization:

- You can generate a Spring Boot application using [Spring Initializr](#) with the following dependencies:
 - Spring Web
 - Spring Data JPA
 - Spring Security
 - H2 Database (or any preferred database like PostgreSQL)
 - Spring Boot DevTools (optional, for development)
 - Lombok (for reducing boilerplate code)
 - Spring Boot Actuator (for health checks)

2. Directory Structure:

- Organize your application into suitable packages.

lua

Copy code

```
com.chatbot
    └── controller
        ├── ChatSessionController.java
        └── ChatMessageController.java
    └── service
        ├── ChatSessionService.java
        └── ChatMessageService.java
    └── model
        ├── ChatSession.java
        └── ChatMessage.java
    └── repository
        ├── ChatSessionRepository.java
        └── ChatMessageRepository.java
    └── security
        └── ApiKeyFilter.java
    └── exception
        └── GlobalExceptionHandler.java
    └── config
        ├── SwaggerConfig.java
        └── CORSConfig.java
    └── ChatbotApplication.java
```

3. Entity Classes:

- ChatSession:

kotlin

 Copy code

```
@Entity
public class ChatSession {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String userId;
    private String name;
    private boolean isFavorite;

    @OneToMany(mappedBy = "chatSession", cascade = CascadeType.ALL)
    private List<ChatMessage> messages;
}
```

- ChatMessage:

kotlin

 Copy code

```
@Entity
public class ChatMessage {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String sender;
    private String content;
    private String context;

    @ManyToOne
    @JoinColumn(name = "session_id")
    private ChatSession chatSession;
}
```

4. Repositories:

- Create JPA repositories for both entities.

java

 Copy code

```
public interface ChatSessionRepository extends JpaRepository<ChatSession, Long> {
public interface ChatMessageRepository extends JpaRepository<ChatMessage, Long> {
```

5. Service Layer: Implement methods for session and message management.

less

 Copy code

```
@Service
public class ChatSessionService {
    // Implement methods to create, delete, rename, mark as favorite
}

@Service
public class ChatMessageService {
    // Implement methods to save messages and retrieve chat history.
}
```

6. Controllers: Expose REST API endpoints to manage chat sessions and messages.

less

 Copy code

```
@RestController
@RequestMapping("/api/chat")
public class ChatSessionController {
    // @Autowired services and implement endpoints for CRUD operations
}
```

```
@RestController
@RequestMapping("/api/message")
public class ChatMessageController {
    // @Autowired services and implement endpoints to manage message
}
```

7. Security Configuration: Implement API key authentication.

java

 Copy code

```
@Component
public class ApiKeyFilter extends OncePerRequestFilter {
    // Logic for API key authentication
}
```

8. Global Exception Handling: Handle exceptions globally.

kotlin

 Copy code

```
@ControllerAdvice
public class GlobalExceptionHandler {
    // Handle exceptions and return proper response codes/messages
}
```

9. Logging: Use `Logger` from SLF4J for centralized logging.

10. Documentation with Swagger:

- Create Swagger configuration to auto-generate API docs.

less

 Copy code

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    // Your Swagger configuration
}
```

11. CORS Configuration:

typescript

 Copy code

```
@Configuration
public class CORSConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").allowedOrigins("*");
    }
}
```

12. Health Check Endpoint: Enable the Actuator and create a custom health

endpoint if necessary.

13. Rate Limiting: Implement a simple rate limiter using a



Provide a prompt ...

Send

Save

Download Chat DeepSeek



Trade - You
Copy Trades
Best Trader
-the-best.com
ite

 DOWNLOAD

AD • 16+

Download Free
Pixelsee. Media
Player For PC
 promo.pixelsee.app

[Download](#)



AD • 16+

Girl washes herself
3D MODEL for CNC
900 ₽
 oaok.ru

[Learn more](#)



AD • 16+

Need speed
proxies? Click.
 proxyelite.biz

[Learn more](#)



[Download Chat](#)