Creating a **developer- and post-release-friendly document** for your Router Control project with user authentication and service interaction requires a focus on clarity, structure, and self-sufficiency. Here's how to structure and draft such a document:

**1. Title and Purpose**

Start with a clear title and an introductory purpose.

**Example:**

**Router Control Project: User Authentication and Service Interaction**  
This document explains the architecture, configuration, and steps to modify or extend the Router Control project, focusing on user authentication and interactions with external services. It aims to ensure developers and stakeholders can work independently without requiring direct developer intervention.

**2. Table of Contents**

Include a table of contents for easy navigation:

* Project Overview
* Architecture Diagram
* Key Components and Responsibilities
* Authentication Workflow
* Service Interaction Details
* Configuration Guide
* Deployment Process
* Common Use Cases
* Maintenance and Troubleshooting
* Glossary and References

**3. Project Overview**

Provide a high-level overview of the project.

* **Objective**: Explain what the project does.
* **Core Features**:
  + User authentication and authorization.
  + Interaction with external services to fetch details.
  + Role-based access control.
* **Tech Stack**: List technologies, frameworks, and libraries used (e.g., Spring Boot, MongoDB, JWT, etc.).

**4. Architecture Diagram**

Include a visual representation of the architecture, showing:

* API Gateway
* Authentication Service
* Router Control Service
* Interactions with other microservices
* Data Flow

Tools like Lucidchart, Draw.io, or PlantUML can help create clear diagrams.

**5. Key Components and Responsibilities**

Explain each module/component with their roles.

**Example:**

1. **API Gateway**:
   * Routes incoming requests.
   * Validates JWT tokens.
2. **Authentication Service**:
   * Generates and validates tokens.
   * Manages user roles and permissions.
3. **Router Control Service**:
   * Handles routing logic.
   * Interacts with external services for data retrieval.

**6. Authentication Workflow**

Describe the user authentication process step-by-step:

* **Login**: How credentials are validated.
* **Token Generation**: JWT structure, claims (e.g., roles, username).
* **Token Validation**: How and where tokens are validated.
* **Role-based Authorization**: Explain the role hierarchy and permissions.

Include diagrams or sequence flows for clarity.

**7. Service Interaction Details**

Detail how the Router Control service interacts with external services:

* **API Endpoints**: List endpoints and their purposes.
* **Authentication Requirements**: If tokens or credentials are needed for service-to-service calls.
* **Response Formats**: Provide example payloads (JSON or XML).
* **Error Handling**: Define how to handle common errors (e.g., 404, 500).

**8. Configuration Guide**

Provide a detailed guide to configure the project without editing the code:

* **External Configuration**: Use application.yml or application.properties for environment-specific settings.
* **Authentication Config**:
* auth:
* jwt-secret: <your-secret>
* token-expiry: 3600
* **Service URLs**:
* services:
* external-service1: https://service1.example.com
* external-service2: https://service2.example.com

Explain how to override these configurations for different environments (e.g., dev, staging, production).

**9. Deployment Process**

Describe the deployment steps in detail:

* Pre-requisites (e.g., Docker, Kubernetes).
* Step-by-step deployment guide.
* Verification checklist post-deployment.

**10. Common Use Cases**

Provide examples of frequent tasks:

* Adding a new role or permission.
* Changing token expiration time.
* Modifying an API route.
* Adding a new external service interaction.

**11. Maintenance and Troubleshooting**

Add self-service troubleshooting steps:

* **Logs**: Where to find logs for debugging.
* **Common Issues**:
  + "Invalid Token" errors.
  + Service unreachable.
* **Solutions**: How to fix or escalate the issue.

**12. Glossary and References**

Define technical terms and acronyms (e.g., JWT, API Gateway).  
Provide links to code repositories, API documentation, and related resources.

**13. Best Practices**

Share guidelines for extending the project:

* Follow the existing coding patterns.
* Use externalized configuration for environment-specific changes.
* Test thoroughly with mocks for service interactions.

**14. Additional Tools**

Include helpful tools and commands:

* Common curl commands to test APIs.
* Example scripts for data seeding or migrations.

**Documentation Tools**

To make it easy to navigate and maintain:

1. Use tools like **Markdown** (for GitHub-hosted docs), **Confluence**, or **Notion**.
2. Include code snippets, diagrams, and examples.
3. Make the document searchable with clear headings.

Would you like help drafting a specific section of this document?