# Proposal for GSoC 2025

RocketMQ-refactor dashboard

## About Me


**BASIC**

**English Name: Crazylychee**
**Chinese Name:** 许億驰(Xu Yichi)

**INFORMATION**

**Email:** xu2757197595@outlook.com
**Phone**: +86 15118878106
**Github:** **Crazylychee (Xu Yichi)**


**EDUCATION**

**Guangdong University of Technology, Guangzhou, China**
**Computer Science and Technology, expected graduation June 2026**

**THE PR**

pref: optimize the response speed of the query api by Crazylychee · Pull Request #273 · apache/rocketmq-dashboard

[ISSUE #274] fix:Failed to find messages older than 3 days using message ID by Crazylychee · Pull Request #275 · apache/rocketmq-dashboard

[issues #279] fix: align top navigation bar styles by Crazylychee · Pull Request #280 · apache/rocketmq-dashboard

**RELATED EXPERIENCE**

- Proficient in Java, familiar with Spring, fluently use Maven
- Familiar with Linux and Docker
- Proficient in using Vue and React to build front-end applications
- Crazylychee Here is my resume.

## Project

### Project Name

Refactoring the RocketMQ Dashboard UI and Enhancing Usability

### Project Description

Refactoring the RocketMQ Dashboard UI and Enhancing Usability

## Background

### Current Status & Existing Implementation

1. The current system is implemented using Bootstrap + AngularJS, but suffers from issues such as misal igned text elements, outdated chart styling, etc.
2. Date picker interactions require manual page refreshes, indicating incomplete AngularJS data binding functionality.
3. Certain API endpoints repeatedly recreate time-consuming resources, resulting in response times excee ding 3 seconds in Windows environment.
4. Security vulnerabilities exist: passwords are stored in plain text, no session management, and coarse-gr ained permission controls.
5. Documentation requires significant improvements in completeness and clarity.
6. The official Docker image is outdated and no longer maintained.

### Unmet Requirements

1. Modern visual design language, unified interaction patterns, and dynamic chart visualization capabiliti es.
2. Real-time interaction feedback (e.g., automatic data updates) and More efficient workflow.
3. Fast Dashboard API responses with efficient utilization of time-consuming resources.
4. Multi-role permission management (RDBC), audit logging for sensitive operations.
5. Reduced learning curve through improved discoverability of key features.
6. Stable, production-ready, and lightweight containerized deployment experience.

### Reusable Components

1. AngularJS business logic, page layout foundations, and backend data interfaces.
2. Date picker components and backend query APIs.
3. Existing resource creation workflows.
4. User authentication module and basic permission validation logic.
5. Current documentation drafts and troubleshooting blog content.
6. docker-compose configurations and build pipeline scripts.

### Areas Requiring Refactoring

1. Rewrite event listeners (e.g., $watch), integrate loading animations (Spinner).

2. Overhaul CSS styling, migrate to modern chart libraries (e.g., ECharts), adopt modular CSS preproces sing (Sass/Less).
3. Optimize API logic to maximize reuse of time-consuming resources.
4. Implement Spring Security + BCrypt encryption, add session timeout controls, phased RBAC adoption . Introduce audit logging.
5. Expand documentation with detailed guidance and examples.
6. Update deprecated Docker images and establish regular maintenance cycles.

**Design / description of work** and **Schedule**d

Phase 1: Frontend Architecture Modernization (Weeks 1-4)

| Task | Priority | Innovation Points | Integration Approach |
|---|---|---|---|
| **Optimize UI Styling** | Mandatory | Modernized visual presentation | Gradual style replacement while preserving existing HTML structure. Rewrite partial CSS for modern design implementation. |
| **Replace Static Chart Library with ECharts** | Optional | Interactive dynamic charts (e.g., real-time monitoring curves), responsive rendering | Encapsulate ECharts components, replace original AngularJS chart directives while maintaining consistent data interfaces. |
| **Refactor Event Listeners (e.g., $watch) + Enhance Workflow Refactoring** | Mandatory | Optimized data binding efficiency through $watch, reduced unnecessary re-rendering | Refactor listener logic in AngularJS controllers. Optimize the workflow, such as creating the subject with the namespace. |

Phase 2: API Performance Optimization (Weeks 4-7)

| Task | Priority | Innovation Points | Integration Approach |
|---|---|---|---|
| **Optimize Resource-Intensive Operations** | Mandatory | Async + caching mechanisms Performance bottleneck identification | Implement asynchronous processing with caching strategies. Audit codebase to eliminate redundant resource loading through pooling/reuse patterns. |
| **Phased RBAC Implementation** | Mandatory | File-based RBAC configuration Granular method-path controls | Enhance YAML configuration schema (role-permission.yml) for multi-role definitions. Integrate dynamic policy resolution in Spring Security context. |

Phase 3: Infrastructure & Security Hardening (Weeks 7-10)

| Task | Priority | Innovation Points | Integration Approach |
|---|---|---|---|
| **Spring Security + BCrypt Implementation** | Mandatory | Password encryption without DB dependency | Enhance user-roles.yml with BCrypt hashing. Configure Spring Security for in-memory authentication and session timeout control. |

| | | | |
|---|---|---|---|
| | | In-memory session storage | |
| **migrate to Java 17 and Spring Boot 3** | Optional | Java 17 and Spring Boot 3 | improve unit/integration test coverage and do Incremental migration |
| **Audit Log Implementation** | Mandatory | File-based operation tracking Zero external storage dependency | Implement AOP interceptors for sensitive endpoints. Align log format with right patterns (timestamp/user/action). |

Phase 4: User Experience Refinement (Weeks 11-13)

| Task | Priority | Innovation Points | Integration Approach |
|---|---|---|---|
| **Intro.js Onboarding Integration** | Optional | Context-aware feature guidance Multi-language support | Trigger tour logic via AngularJS router transitions. Implement dynamic content matching based on current page functionality. |
| **Documentation & Image Maintenance** | Mandatory | "5-Minute Quickstart" tutorial Troubleshooting section | Sync documentation repo with codebase via Git hooks. Implement automated image tag versioning aligned with Git releases. |

**Some implementation details in my solution as examples:**

During my analysis of the dashboard's RocketMQ5 support, I identified significant fragility in the implementat -ion. In a stress test with 900 topics and 2300 consumers, I observed interface timeouts and failed queries. Usin -g Arthas commands, I pinpointed the following root causes for slow topic queries:

1. Each topic triggers an examineTopicConfig call (~2s) without exception handling
2. Redundant examineBrokerClusterInfo executions (115ms each) retrieving duplicate data
3. No caching mechanism for repeated queries
4. Misclassification bug: DLQ topics being incorrectly identified as delay topics using %D pattern

My Optimization Implementation:

Topic Query Improvements:

1. Cached examineBrokerClusterInfo results by:
   - Extracting it into a service class
   - Setting 5-minute cache expiration
2. Implemented two-level caching for examineTopicConfig:
   - Initial call populates cache
   - Frontend refresh triggers cache invalidation
3. Fixed topic type identification logic
4. Use the getAllTopicConfig method in the MqAdminExt interface to query topic types in batches.

Consumer Query Optimization:

1. Cached consumer group configurations with:
    - Cache warmup on first query
    - 24-hour TTL with LRU eviction
2. Implemented dual refresh strategy:
    - Scheduled cache rebuild at 2:00 AM daily
    - Exposed per-consumer refresh API
3. Collaborated on RocketMQ batch query API development
4. Make full use of the consumer type returned by getAllSubscriptionGroup in the MqAdminExt interface to simplify the query of consumer type.

Current Status:

I have implemented these solutions in MyDetailCodeAboutThat and observed:

- Topic query latency reduced from 30s to <1s
- Consumer group query time decreased from 6 min to 800ms
- Fixed DLQ/delay topic misclassification

I look forward to contributing these optimizations to the community.

## Results for the Apache community

**For Users:**

1. Smoother and more intuitive user experience
2. Enhanced security and compliance
3. Stable deployment and maintenance experience
4. Reduced operational costs

**For Developers:**

1. Code architecture that is easier to maintain and extend
2. Standardized development processes and documentation systems
3. Strengthened community ecosystem

**For the Community:**

1. A positive onboarding experience can attract more users to adopt RocketMQ and encourage greater participation in its maintenance, increasing community vibrancy.

## Deliverables

- Optimize UI Styling
- Refactor Event Listeners (e.g., $watch) + Enhance Workflow Refactoring
- Optimize Resource-Intensive Operations
- Implement Spring Security + BCrypt Authentication
- Audit Log Implementation
- Documentation & Image Maintenance

- Replace Static Chart Library with ECharts  (optional)
- Phased RBAC Implementation
- Migrate to Java 17 and Spring Boot 3  (optional)
- Intro.js Onboarding Integration  (optional)

## Other Time Commitments

During the summer break, I will dedicate approximately 4 hours daily, totaling around 30-40 hours per w eek. Any additional free time will also be allocated to these efforts.

## Community engagement

- Engage the community via issues;
- Discuss with mentors via WeChat;
- Submit deliverables through PRs.