Airtribe

Backend Engineering Launchpad - Case Study

Chronos : Job Scheduler System

Author: Airtribe

Chronos

Background & Objective

In modern computing environments, being able to efficiently schedule, manage, and monitor tasks, also known as jobs, is crucial. These could range from simple tasks like sending a weekly email to more complex operations like processing data or maintaining databases. This project will have you design and implement a robust and scalable backend for a job scheduling system.

Develop a reliable and scalable distributed job scheduling system that can execute, manage, and monitor a variety of tasks. The system should support one-time jobs as well as recurring jobs, providing comprehensive job management functionality. While the system will primarily handle backend functionalities, there should be provision for interaction with a potential frontend through well-defined APIs.

Key Features

1. **Job Submission:** Implement a mechanism for users to submit jobs that can be executed either immediately or at a specific future time. The submitted jobs can be of various types and complexities.

2. **Recurring Jobs:** The system should be capable of handling jobs that recur at specified intervals. This could include hourly, daily, weekly, or monthly tasks.

3. **Job Management:** Create APIs for users to manage their jobs. This should allow them to view the status of their jobs, cancel jobs, and reschedule jobs.

4. **Failure Handling:** Design a mechanism to handle job failures, with a system for automatic retries. If a job consistently fails, the system should notify the user.

5. **Logging and Monitoring:** The system should maintain detailed logs of all job executions. Implement a monitoring system that keeps track of job statuses and overall system health.

Technical Requirements

- Your solution should be implemented as RESTful APIs

- Use a reliable database system for managing jobs and their schedules.

- Implement appropriate authentication and authorization mechanisms for job submission and management.

- Design and implement mechanisms for automatic retries.

- The system should be designed keeping scalability in mind. It should be able to efficiently handle an increasing number of tasks.

Assessment Criteria

- Functionality: Does the system work as intended? Does it meet all the requirements stated above?

- Code Quality: Is your code clean, organized, well-commented, and following best practices?

- Design and Structure: Is the system well-designed? Does it demonstrate a good understanding of system design principles and patterns?

- Documentation: Is your report comprehensive and clear? Does it effectively explain the choices made and how to use the project?

- Presentation: Do you effectively demonstrate and explain your system and the decisions made during its development?

Deliverables

1. The final, functional product.
2. README file outlining how to use the system, API documentation, the design decisions and other necessary information.
3. Public link of the Github repository
4. Explainer video demonstrating your project work