Build a Monitoring Portal Using .NET MVC (C#)

Senior Developer / DevOps – Technical Challenge

By Luke Rocco

Objectives

Brief

Objective:
Develop a lightweight monitoring portal using the .NET MVC framework. This portal will display monitoring data and alerts such as the ones provided in the sample csv file shared with this task. The focus should be on clean, maintainable code and creating a user-friendly interface with visualization capabilities.

Requirements:

1. Frontend Features:
    - Dashboard Page:
        - Use widgets, accordions, or collapsible panels to group and display monitoring information by system.
        - Each widget/accordion should be according to the system:
            - Status (OK/Warning/Error).
            - Alert Details
        - Include a color-coded status indicator for quick visualization (e.g., green for OK, yellow for warning, red for error).
    - Allow users to expand an accordion to see more details of alerts for a system.
    - Ensure the UI is responsive and works well across different screen sizes. You may use libraries like Bootstrap for styling and layout
2. Alert Details:
    - Each expanded view should display:
        - Timestamp of the alert.
        - Severity (Critical/Warning/Info).
        - Message describing the issue.
    - Implement a sorting feature for alerts (e.g., by timestamp or severity).
3. Data Source:
    - Alerts should be read from the provided flat file (i.e., CSV). Further alerts/dates/systems can be added to the provided file as desired. Should it be more comfortable to set a local database from which alert details are extracted, feel free to do so and provide us with migration data to set up environment.
    - Provide a method to simulate real-time updates (e.g., re-read the file every minute, as example).
4. Code Quality:
    - Write clean, maintainable, and modular code.
    - Use appropriate design patterns and techniques where applicable such as Repository Pattern, Interfaces, and DTOs
    - Follow .NET coding standards.
5. Scalability and Performance:
    - Ensure the design allows for easy addition of more systems or data sources in the future.

Optional Enhancements

- Ability to define 'alerts':
    - Solution can allow a user to define 'alerts' which essentially do a simple task (like a basic query) according to a given schedule to produce the result set.
- Search and Filtering:
    - Add a search bar to filter by date or alerts by keywords or criteria.
- Export Data:
    - Add functionality to export the displayed alerts to a CSV or JSON file.
- Appropriate Error Handling:
    - Implement error handling mechanisms

Deliverables:

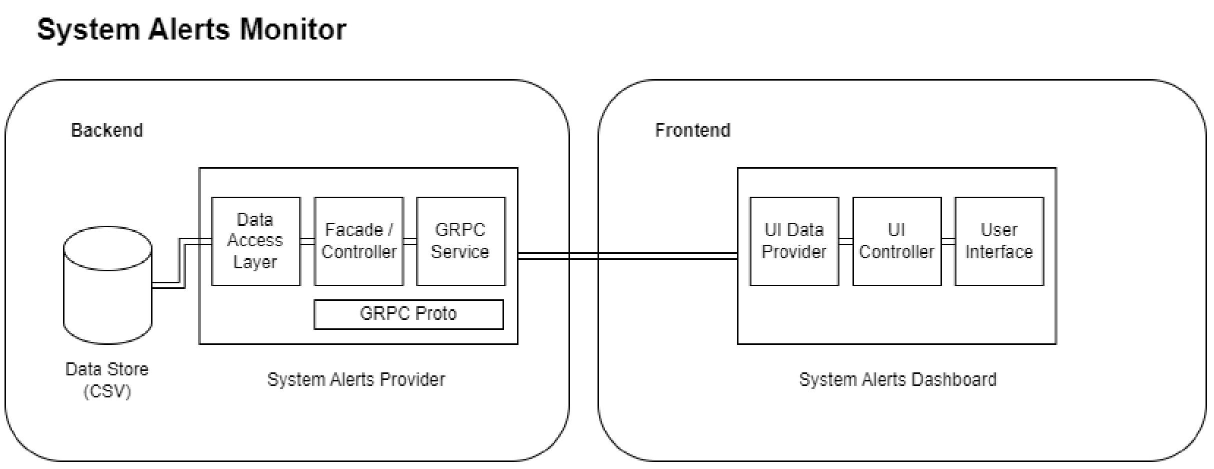
1. A functional .NET MVC application that fulfils the requirements.
2. A short README file with:
    - Steps to run the application.
    - Any assumptions made or additional features implemented.
3. A brief document explaining any architectural decisions such as the entity models implemented, and/or patterns used.

Sample Alerts:

Sample_Monitoring_Alerts.csv

Project Specifications

Project Diagram



The Minimum Viable Product (MVP) of the project will be divided into two sections. The Front End which is named the "*System Alerts Dashboard*" and the Back End which is named the "*System Alerts Provider*".

System Alerts Monitor

For the purposes of this MVP, the backend will be implementing the following features:

Project Data Store

As per the explanation in the above brief, the data store being used is a CSV export containing a snapshot of the System Status Alerts. Upon further revisions or enhancements, however, this backend source can be retrofitted to work with either a standard Relational Database, NoSQL Database or Data Lake datastores. It is even possible to have the System Alerts Provider query directly the Live Status of the systems being monitored.

System Alerts Provider

The backend component of the MVP, called the System Alerts Provider, will serve as an entry point for front end and other systems to query the project's data store. This access point is being envisaged as a GRPC Service, which although seems like an overkill for the time being, leverages the benefits of Scalability, High Performance and Streaming Support which would be useful for real-time monitoring of systems should the system be developed further.

Other alternatives include developing the System Alerts Provider as a REST API which is simpler, easier to debug, while remaining scalable, however might not be suitable for real-time monitoring.

Alternatively, the System Alerts Provider could be developed as a simple DLL Library, and possibly deployed as a NUGET Package, to be referenced and consumed by the front end and/or other systems, thus giving direct access to the datastore. This option would not be viable if the system in question does not have direct access to the datastores being consumed.

System Alerts Dashboard

The frontend component of the MVP, called the System Alerts Dashboard, will serve as the Web Interface a user can make use of to review and investigate the Status of the Systems being monitored.

For the purposes of the MVP, not Login and Authentication will be implemented on the Dashboard, so the information will be immediately accessible to anyone who has access to the URL of the Dashboard. Future enhancements to the project may include the need to Login prior to accessing the Dashboard page, and even Role Based access to the different functionalities available.