Small Project 2: Alarm Management

Contents

[Basic Problem Statement 1](#_Toc97718418)

[Types of alarms table: 1](#_Toc97718419)

[User Interface 1](#_Toc97718420)

[Database 4](#_Toc97718421)

[APIs/Services 4](#_Toc97718422)

[APIs can be used 5](#_Toc97718423)

[System Testing 5](#_Toc97718424)

[Database Testing 5](#_Toc97718425)

[User Interface Testing 5](#_Toc97718426)

[API Testing 5](#_Toc97718427)

[Summary of technologies and tools used 5](#_Toc97718428)

[Technology 5](#_Toc97718429)

[Tools 5](#_Toc97718430)

This project involves building Database, API and UI and integration and testing of the same for the problem statement and user story described below.

# Basic Problem Statement

A manufacturing factory has a given set of machines, for example, Printer-Slotter machines, Auto-Stitcher machines, Rotary Auto-Feed machines, or Auto-Corrugator Machines. It has designed a sensor-based system to compute the matrix required for Overall Equipment Effectiveness. The system helps measure the parameters such as availability, performance, quality and OEE for any given machine over an interval of time.

We need to extend the system by providing a module to capture Alarms/Alerts.

1. For now, we will have 4 alarms created for each mentioned name having similar conditions for it. Additional conditions can be added as shown in the UI

Types of alarms table:

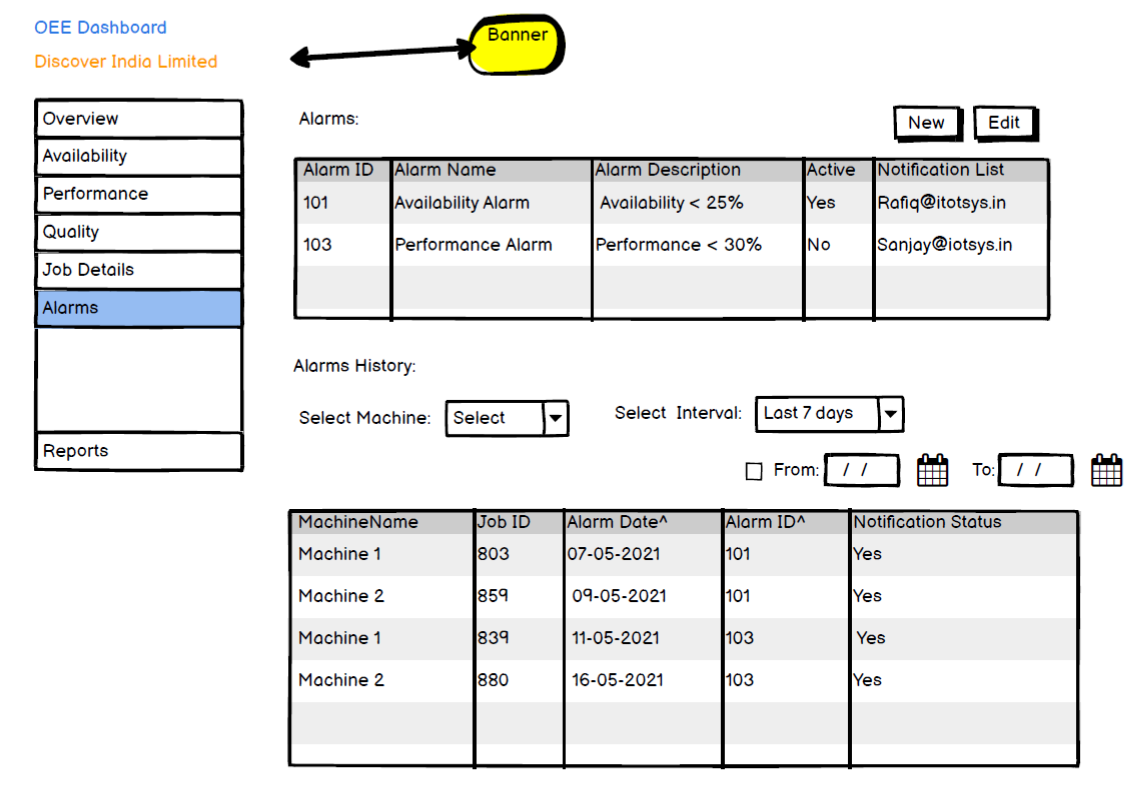
|  |  |  |
| --- | --- | --- |
| SR.No. | Alarm Name | Condition |
| 1 | OEE Alarm | If OEE is under 5% |
| 2 | Availability Alarm | If Availability is under 25% |
| 3 | Performance Alarm | If Performance is under 30% |
| 4 | Quality Alarm | If Quality is under 90% |

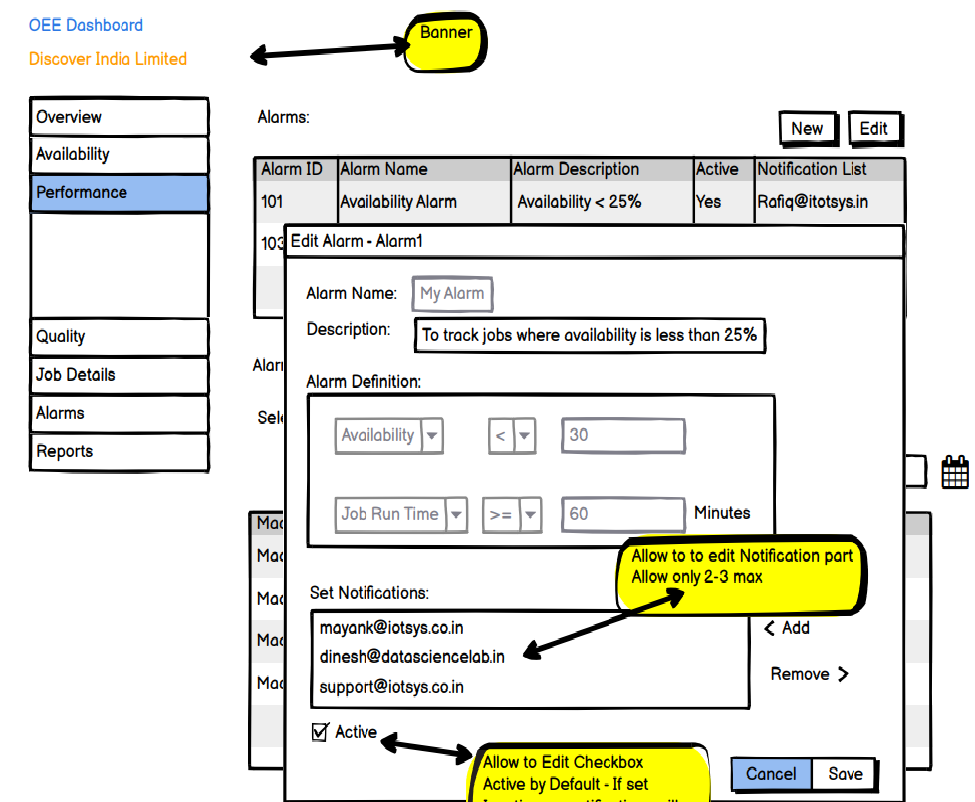
1. For simplicity, each alarm can be assigned to any particular machine and to its running job so that a manager or operator can get notifications regarding current running job.
2. Alarm state can be set to Active/Inactive. Only “Active” Alarms will be considered while creating History/Notifications
3. Alarm history table will have records of all notified alarms with their non-editable job and machine level details. Table should have same access control facilities like other tables.
4. It is desirable to have a separate service to send notification from time to time (say, at a given interval every 15 min or so.) The system should send notifications only if not sent earlier.

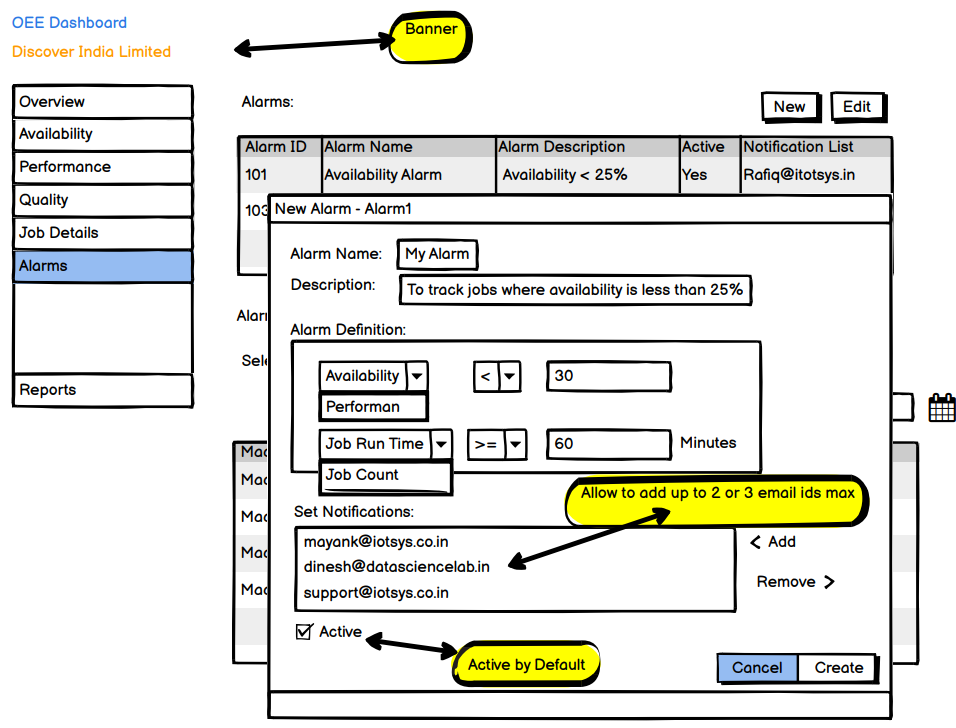
# User Interface

The user interface will consist of WebApp (React App) satisfying minimalistic functional requirements: (Please see the mockups below to get an idea)

1. Supporting UI to carry out crud operations (Create/Edit/Delete) Alarm.
2. Facility to display Alarm History table
   1. This should show which Alarm Id refers to which Alarm Definition
   2. The columns should also include Job Id, Machine ID etc.
   3. Facility to drop Alarm History (in the past until some specific given period)
3. Any other components that will make user interaction more intuitive.







## Database

The database schema and tables should be suitably designed to capture all relevant information like Alarm Definitions, Alarm History details, any other intermediate tables if required etc. in an efficient manner. Any suitable Relational Database such as MySQL or Postgres can be used.

# APIs/Services

The following set of APIs need to be designed and implemented. Any supporting APIs that are required need to be considered and included.

1. 4 main APIs should be there to create [POST], edit [PUT], display [GET], destroy [DELETE] alarm.
2. Alarm History Service – This service will actually process the information available from other APIs and create “Alarm History” as per the constraints explained in the section about ‘Basic Problem Statement’
3. Purge History Service - This service should facilitate drop past history until a given date in the past in ‘Basic Problem Statement’
4. A service to get Alarm History at any given point.
5. Use following API endpoints to get job-status and job-analytics data.
6. E-mail Notification Service (Expanded Scope)

# System Testing

This will involve testing all the components of the system.

## Database Testing

1. Testing all operations, and schema for data integrity and sanctity
2. Any other conditional database tests during development of Services/other UI Components

## User Interface Testing

1. Writing Test cases, reviewing these for correctness
2. Manual Functional Testing
3. Learning and using the tool Cypress for overall application testing

## API Testing

1. Writing Test cases, reviewing these for correctness and effectiveness
2. Running API Tests with Postman
3. Learning and using the tool ‘RestAssured’ for API testing

# Summary of technologies and tools used

## Technology

JavaScript, React JS, Node JS, HTML, CSS etc.

## Tools

Cypress, REST-Assured, Postman, JIRA, GIT etc.