

Design Description

Project Name: A cloud-native performance monitoring system

Client: Research Institutes of Sweden (RISE)

List of Contents

1. Introduction	2
2. Design	2
2.1 System Components	2
Frontend	2
Backend	3
Database	3
2.2 Communication With External Systems	4
2.3 Third Party Frameworks	5
2.4 Implementation Structure	6
3 System Usage	7
3.1 GUI Structure	7
3.2 User and Activity flow	9

1. Introduction

The client for this project is RISE - The research institute of Sweden. They are an organisation that works for sustainable growth in Sweden by supporting innovations in the business community. They are relocating their workloads such as data, applications, websites, databases, storage, physical or virtual servers, or entire data centers to a cloud infrastructure. RISE is currently migrating to the cloud, since cloud computing increases efficiency, flexibility, security and offers many more benefits. Specifically they would like to have an easy way to monitor the performance of their AWS virtual machines.

The project is about monitoring the performance-related metrics of a virtual machine and visualizing them by making use of AWS CloudWatch (AWS's built-in monitoring system). Users will be able to view a dashboard on a website where they can monitor specific metric-data. There is also an admin page for turning off specific data collection.

2. Design

2.1 System Components

What are the main parts of your system and how do they communicate?

There are three major parts in the system. There is the website frontend, backend, database.

Frontend

In the frontend the user can view performance metrics in the form of graphs displayed on a dashboard.

These graphs display metrics statistics over a specific period of time that is chosen by the user. There are three static time intervals available for the user to choose. The user can also choose between three different ways of displaying the data which are : line chart, pie chart and bar chart.

The frontend only directly communicates with the website backend, through an internal API exposed by it. It does not communicate with CloudWatch directly, this is because we do not want the frontend to have access to the credentials that are used for the CloudWatch API.

When the user first enters the dashboard page, the frontend asks the backend which metrics are active. For each active metric it requests from the backend to retrieve the data for that metric from CloudWatch in the specified static interval. The graphs are then drawn on the dashboard with the returned data through the backend from CloudWatch. There is also a timer set when the dashboard is loaded to retrieve new data every five minutes because that is how often CloudWatch logs information.

When an user tries to access the settings page, a login page will be shown. This is because the settings page is only meant for admin users. In order to verify their credentials the user has to enter their user information, and a login request will be made to the backend. Which in turn verifies that the credentials match the ones that exist within the database.

Once an admin toggles a metric on the settings page, the action will be reflected inside the database. Which in turn affects the dashboard since the dashboard uses information from the database in order to know which metrics to display.

After the admin is logged in, the front-end will request the status of each metric from the back-end, it will then provide the admin with a panel that they can use to toggle if a specific metric should be active or not.

Backend

The website backend takes care of communication between CloudWatch, the website frontend, and the database. Specifically, the backend provides an API for the frontend to easily retrieve data from both CloudWatch and the database.

The backend can retrieve data from CloudWatch by using an API provided by Amazon. It allows you to specify which metric you want to collect data from as well as an interval. This design could be more efficient because we do not store the data anywhere. Since the frontend is set to update the dashboard every five minutes, we are effectively querying the CloudWatch API for duplicate data each time it wants to update.

Database

The database contains information about all the metrics that are used in the system, and whether or not they are enabled for data collection. It also contains user-information for the admin accounts. Although it would be very efficient, we are not storing any of the CloudWatch metrics that are being returned by their API.



 name	display_name	 collection_status
StatusCheckFailed_System	Status Check Failed (System)	1
StatusCheckFailed_Instance	Status Check Failed (Instance)	1
StatusCheckFailed	Status Check Failed	1
NetworkPacketsOut	Network Packets Out	1
NetworkPacketsIn	Network Packets In	1
NetworkOut	Network Out	1
NetworkIn	Network In	1
MetadataNoToken	Metadata No Token	1
DiskWriteOps	Disk Write Ops	1
DiskWriteBytes	Disk Write Bytes	1
DiskReadOps	Disk Read Ops	1
DiskReadBytes	Disk Read Bytes	1
CPUUtilization	CPU Utilization	1
CPUSurplusCreditsCharged	CPU Surplus Credits Charged	1
CPUSurplusCreditBalance	CPU Surplus Credit Balance	1
CPUCreditUsage	CPU Credit Usage	1

Figure 1: The metric table inside the database, it stores the metric name that is used when querying the CloudWatch API, as well as a display name that is shown in the dashboard. Lastly it also stores the collection status which means whether or not it should be displayed in the dashboard.

username	password
admin	\$2y\$10\$B5CvGN.JXgB03EZxjQIIHeszMYkF85ec6sJXSr...

Figure 2: The user table inside the database, it stores the login information.

2.2 Communication With External Systems

Our system only interacts directly with AWS's CloudWatch; their own performance monitoring system. CloudWatch handles the communication with the virtual machines running on AWS, which it keeps track of automatically. Metrics are collected from the administrator that CloudWatch can provide and the users can watch or include CPU utilization, RAM usage, network throughput, and so on.

Our system interacts with CloudWatch through Amazon's public API, which we get access to through one of their several language-specific software development kits (SDK). These SDK's expose an API for CloudWatch which we use to fetch whatever data we want from it, including the metrics it collects for itself. In turn, the backend constructs these metrics as a dataset appropriate for constructing graphs in the frontend.

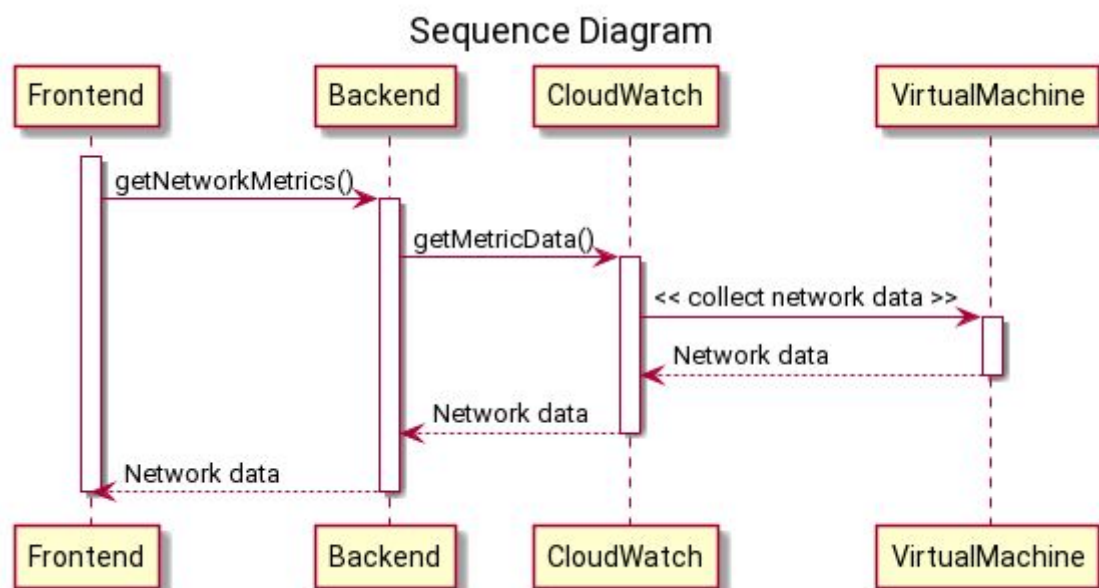


Figure 3: A sequence diagram of a metrics request

2.3 Third Party Frameworks

The major 3rd party framework used in this project is React. React is a JavaScript, front-end library that is used for creating interactive user interfaces.

In order to use React efficiently we need to know that it is a component-based library. Component-based means that React divides web pages into several components. A component is a bit of code that represents a certain part of a web page.

React differs from “traditional” web development, in the sense that it’s only concerned with rendering data to the DOM, thus we can’t go the normal route of creating HTML-pages, and later on adding JavaScript functionality on top. What this means for us, is that we will have to write our HTML as smaller components, that we can then tell React to render directly to the DOM.

2.4 Implementation Structure

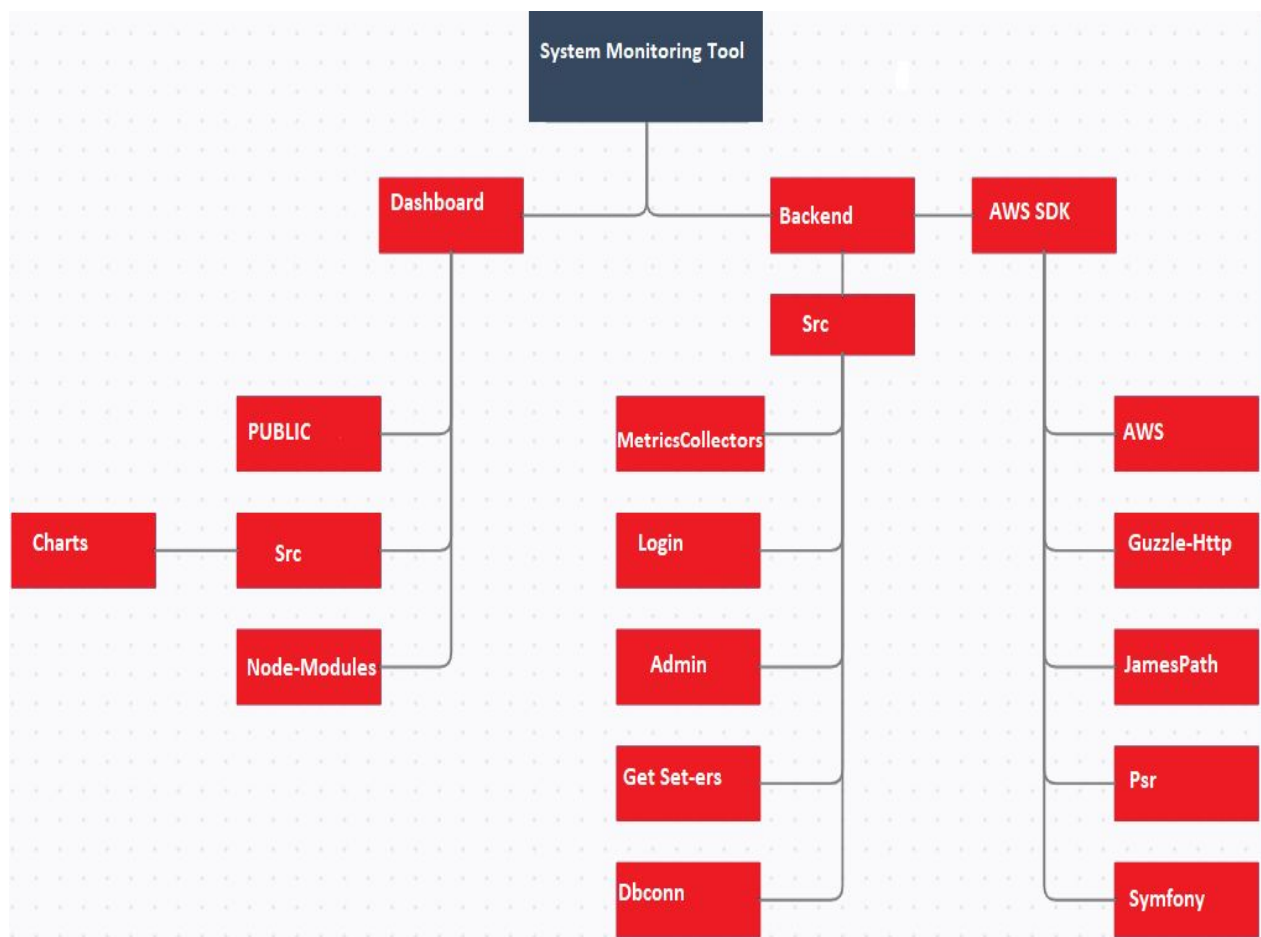


Figure 4: An overview of the folder structure.

As can be seen in Figure 4 the Dashboard is the frontend part of the project structure while the Backend holds all PHP and the AWS SDKs. As we are using React for our frontend the structure's already predefined by the standard React setup. We have on the backend chosen to retain the same structure setup that React uses for the sake of keeping fidelity/consistency throughout the project structure.

While neither frontend or backend is written in an object oriented way, we wanted it to at some degree have it as isolated and modular from each other as possible. Hence for example we wanted to have a **MetricsCollectors folder** that stored each type of request to the CloudWatch API separate from the front end requests. This is why there's also a folder specifically for getters and setters as they act as the API to which the frontend communicates without having to care at all about the specifics of the communications between CloudWatch API and our server.

Similarly each folder includes a **functions.php** that contains all functions for each module. The reasoning behind this is mainly to reduce the size of each source file to only contain the most pertinent information instead of being littered with functions performing large tasks. This is something that usually isn't done in other languages but is something that many of us have found to be very effective while writing PHP code. This retains code readability (amongst other things), something that we've found to quickly degrade when writing complex systems.

3 System Usage

3.1 GUI Structure

The GUI is structured by web pages. There is a login page, a dashboard and a settings page. Through the login page the admin can enter username and password in order to login. In the admin page the users with admin status will be able to turn off collection of certain metrics. The dashboard contains items and tools that displays various metrics collected.

Navigation is made through a global menu bar, present on every page, which contains links to each page.

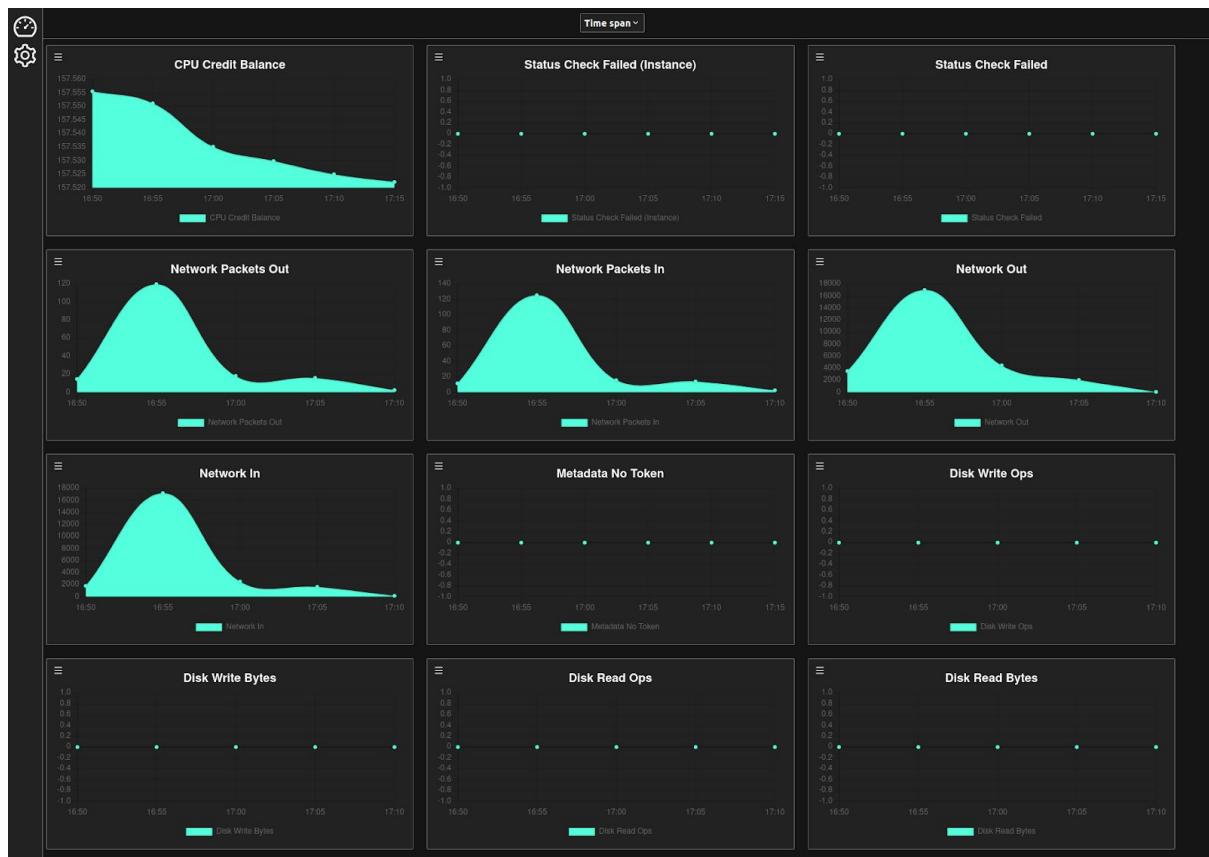


Figure 5: The dashboard page

Figure 5 shows the final design for the front end and the graphs show the metrics collected from CloudWatch. The left-most menu bar is global and contains links to the dashboard and the settings page. The top-most menu bar on the dashboard is used for setting the interval in which you want the data to be displayed. Each graph also has a hamburger menu where the user can choose to display the graph in a different way or hide it.

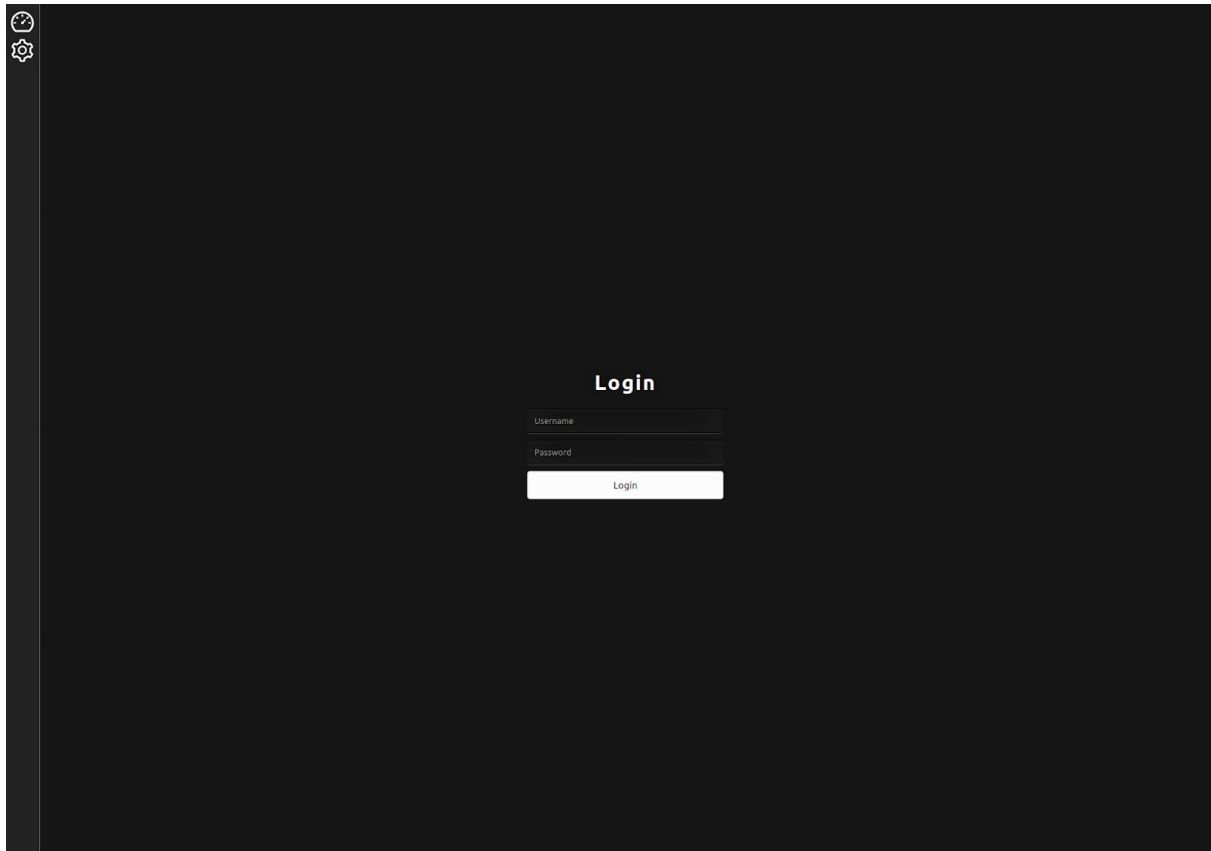


Figure 6: The login page that is displayed when an unauthorized user tries to access the settings page.

The login page is displayed when an user is not logged in to the system and they attempt to access the settings page. In order to login the user enters their username and the password and presses the login button. If the login was successful the settings page will show, if not the login page will still be presented.

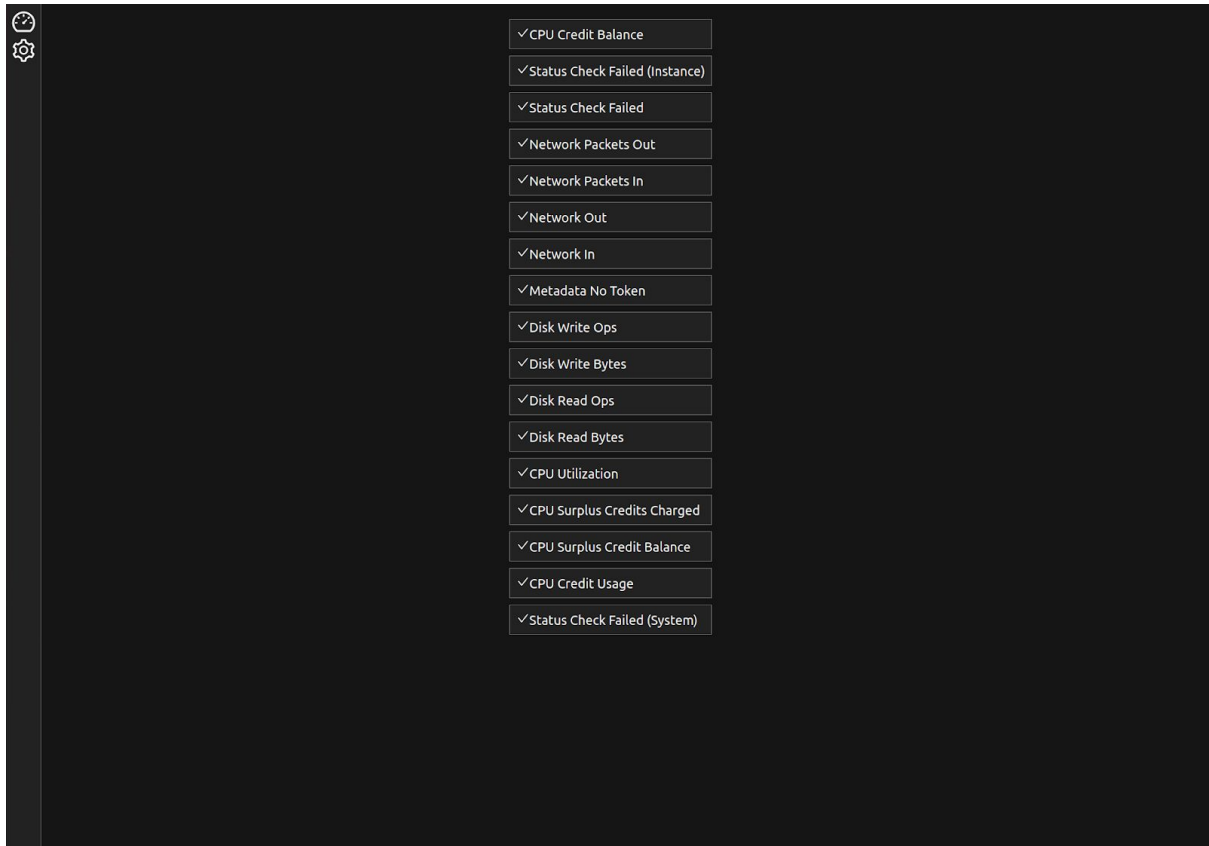


Figure 7. The settings page that only admins are able to access. Through the panel shown in the figure, they can toggle the collection of data for specific metrics.

The admin is able to toggle the collection of data for specific metrics through a panel that resides in the settings page. If a metric is toggled off by the admin, it would no longer show up inside the dashboard for the users of the system. In order to toggle a metric the admin simply clicks on it because they act as buttons.

3.2 User and Activity flow

The user interacts with the application through the website. From the website the user can access different kinds of graphs and tables with information about metrics usage of other applications. It is up to the user which graphs will be displayed since the user can decide to hide graphs on their dashboard. This also assumes that the metric is enabled by the admin. The graphs are updated automatically through timed API calls to the backend.

A user with admin status can login to an admin page and manage the metric collection. Normal users do not require any kind of authentication, they can simply go to the dashboard page and look at the graphs.

Figure 8 shows an activity diagram displaying the flow from different activities in the system.

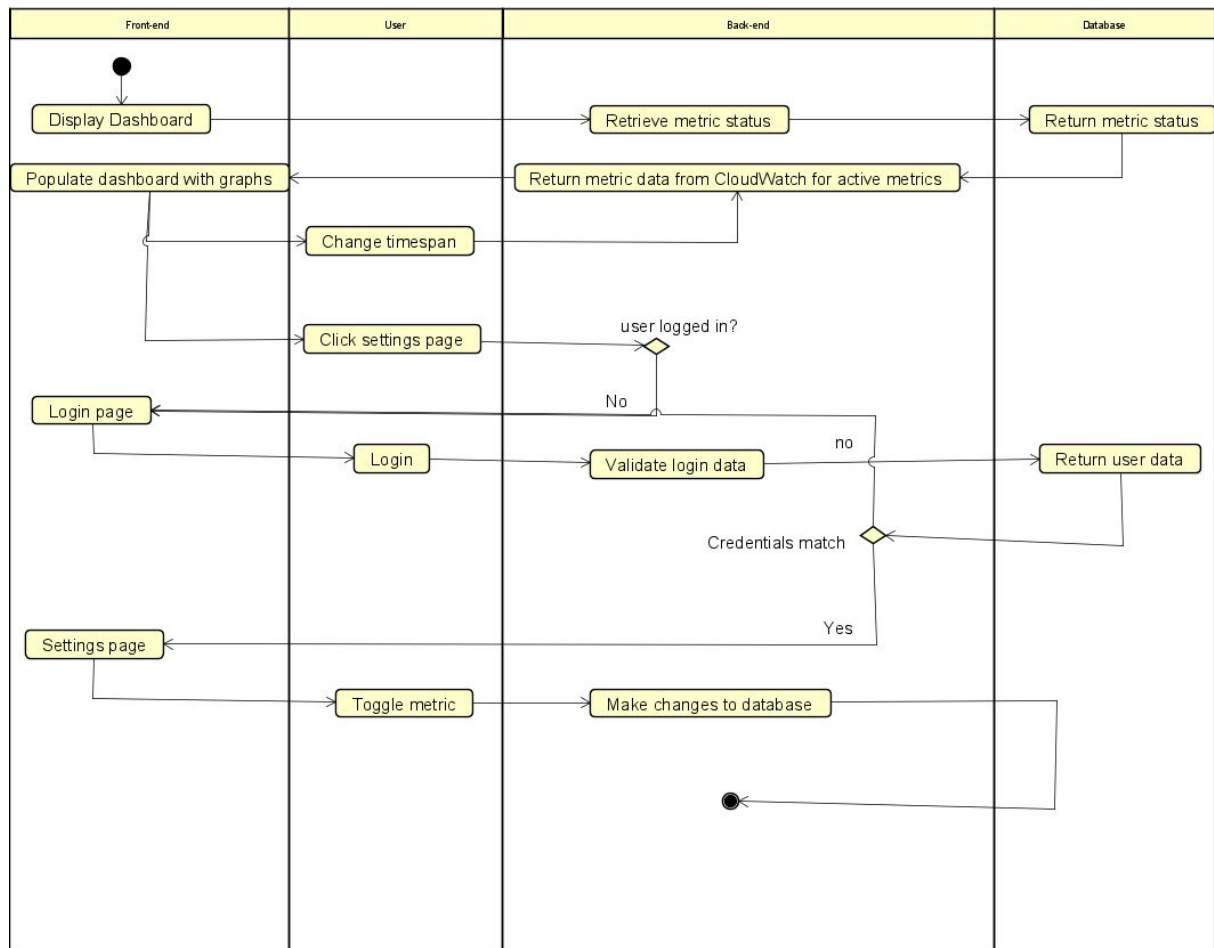


Figure 8: An activity diagram of the system.