Design Description

Project Name: A cloud-native performance monitoring system
Client: Research Institutes of Sweden (RISE)

Emil Larsson: eln10007@student.mdh.se
Gezim Cako: gco20001@student.mdh.se
Gunnar Andersson: pan18010@student.mdh.se
Johan Karlsson: jkn18010@student.mdh.se
Roland Plaka: rpa20001@student.mdh.se
Sandra Eriksen: sen18014@student.mdh.se
Ville Svensson: vsn17004@student.mdh.se

Contents

Introduction

The client for this project is RISE - The research institute of Sweden. They are an organization that works for sustainable growth in Sweden by supporting innovations in the business community. RISE is currently migrating to the cloud, specifically AWS, and 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 and search for specific metric-data. There will also be an admin page for approval of new users and turning off specific data collection.

System Design

System Components

There are three major parts in the system. There is the website frontend (i.e the dashboard) with which the user can view performance metrics in the form of graphs. These graphs will be able to display metrics statistics over a specified period of time. The dashboard will be user configurable, as in which graphs will be visible to the user. While the dashboard will be configurable, the graphs themselves will not.

Dashboard configuration will be user specific, so that any one user cannot change how others will view the dashboard. As of this stage, users will only be able to say whether or not a given graph should be visible, and not *how* they will be displayed on the dashboard. Thus, user configuration can easily be constructed in a standard file format like JSON. An example of this can be seen in figure 1.

```
{
      "config": [
            {"cpu_utilization", true},
            {"ram_usage", true},
            {"network_usage", false}
                ]
}
```

Figure 1: JSON config file example. In this context true would mean visible, and false not

The frontend will only directly communicate with the website backend, through an internal API exposed by it.

Then there is the website backend itself, which will take care of communication between CloudWatch, the website frontend, and the database. Specifically, the backend will provide an API for the frontend to easily read data from both CloudWatch and the database.

The database will contain user information, such as usernames, passwords, and privilege level (regular user or administrator), as well as their personal dashboard configurations. The database will only be used to store user-related data and will not store anything exposed by CloudWatch.

When a user first accesses the front-end, a login page will be shown. The user will enter their user information, and a login request will be made to the backend, which will call a procedure on the database.
After the user is logged in, the front-end will request metric statistics data from the back-end, and the back-end in turn will request the data from CloudWatch. The data will get returned through CloudWatch and the back-end, to the front-end that can now construct its graphs.
An example of this sequence can be seen in figure 2.

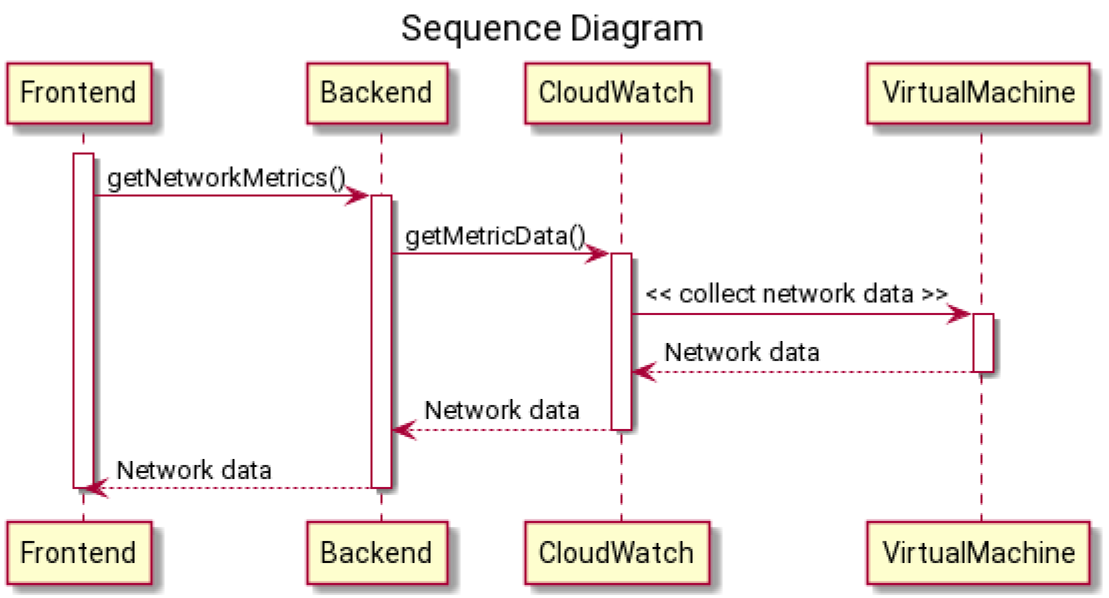


Figure 2: A sequence diagram of a metrics request

Communication with External Systems

At the time, our system will only interact directly with AWS’s CloudWatch; their own performance monitoring system. CloudWatch will handle the communication with the virtual machines running on AWS, which it keeps track of automatically. Metrics that CloudWatch can keep track of include CPU utilization, RAM usage, network throughput, and so on.

Our system will interact 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 can use to fetch whatever data we want from it, including the metrics it collects for itself. In turn, the backend can construct these metrics as a dataset appropriate for constructing graphs in the frontend.

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.

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.

Implementation Structure

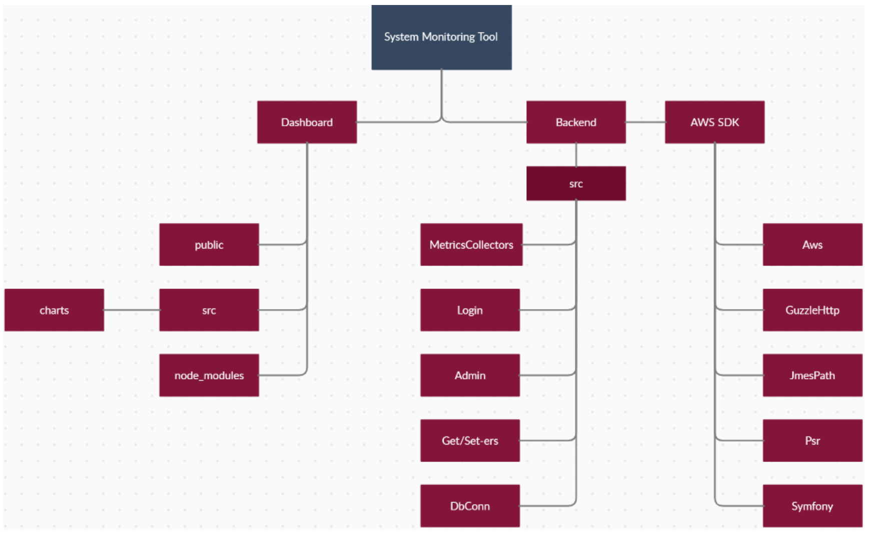


Figure 3: An estimation of how the project is structured.
Note, it is an initial structure and is subject to change.

As can be seen in Figure 3 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 structures 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 nor backend will be written in an object-oriented way, we want to at some degree write it as isolated and modular from each other as possible. Hence for example we'll want to have a MetricsCollectors folder that has 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 should be the API to which the frontend can communicate without having to care at all about the specifics of the communications between CloudWatch API and our server.

Similarly, each folder should include 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 functionality in PHP.

System Usage

GUI Structure

The GUI is structured by web pages. There is a login page, a main page and an admin page. Through the login page the user can enter username and password to login. In the admin page the users with admin status will be able to turn off collection of certain metrics. The mainpage will contain items and tools that will display different metrics collected. Users will be able to manipulate and hide certain search results to get the metrics they want.

Navigation will most likely be made through a global menu bar, present on every page, which contains links to each page.
Changes to the current page might also be made through a context bound menu bar. If the user wants to change which time period the graphs should be displaying, that change can be made with a drop-down menu present in the dashboard's context bound menu bar.
The global menu bar might be located to the left-hand side, while the context bound menu bar might be located on top of the page.

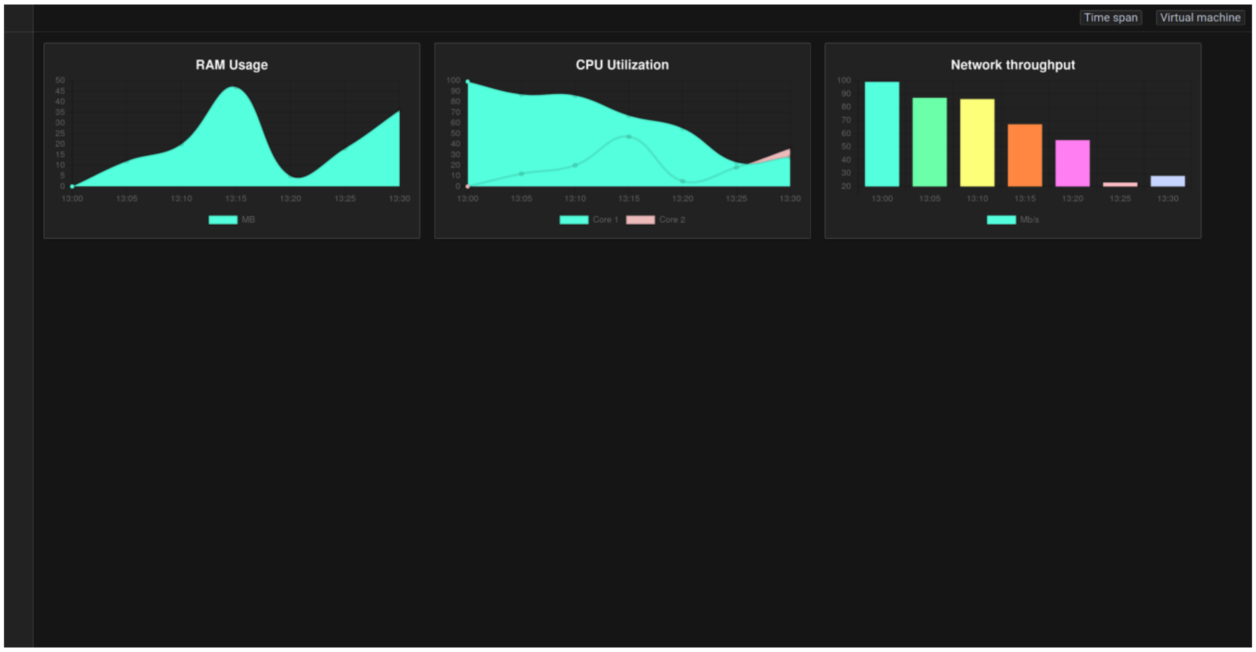
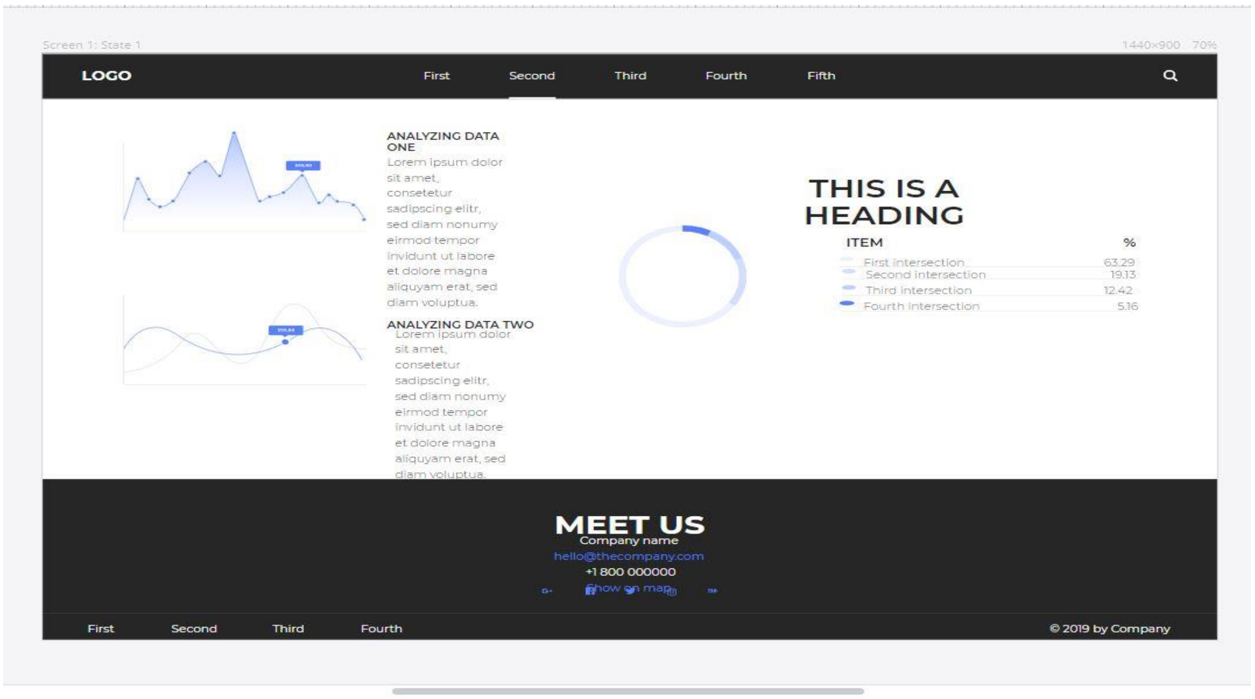


Figure 4: An example a dashboard design

We have created some graphical design examples to base our front-end implementations on. In figure 4 we can see a fairly simple dashboard design, consisting of two menu bars, on the left and one on the top, and a middle area where graphs are shown.
The left-most menu bar is meant to be global and would contain links to every other page. The top-most menu bar is meant to be context-bound and would only contain buttons necessary for the current page (the dashboard would populate the top bar with buttons related to itself).

 In the middle is an area which would contain all the rest of the page information. On the dashboard, it would contain the actual graphs, and on the user settings page it would contain options for configuring which graphs will be shown, and so on



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

ITEM	%
First intersection	63.29
Second intersection	19.13
Third intersection	12.42
Fourth intersection	5.16

Figure 5: An example of the website design

In figure 5 we can see another example. The design of this website is built in a header, body and footer, with this structure users can navigate in a simple way and users will have a good user experience.
The thoughts behind the design is that the user should be able to easily navigate through the web pages with help of a top navigation bar and a footer. The first page will be the dashboard, showing above. More than that, the user can navigate to the other webpages and get a more detailed analysis over that specific metric. For example, one page will only consist of diagrams and graphs for CPU and another webpage for only RAM etc. The links in the navigation bar will be named after what metric to analyze.

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 data will be displayed since the user can decide what to show and hide in the tables.

By logging in the user can access the dashboard and all its functions. A user with admin status can login to an admin page and manage other users and metric collection. Figure 6 shows an activity diagram displaying the flow from different activities in the system.

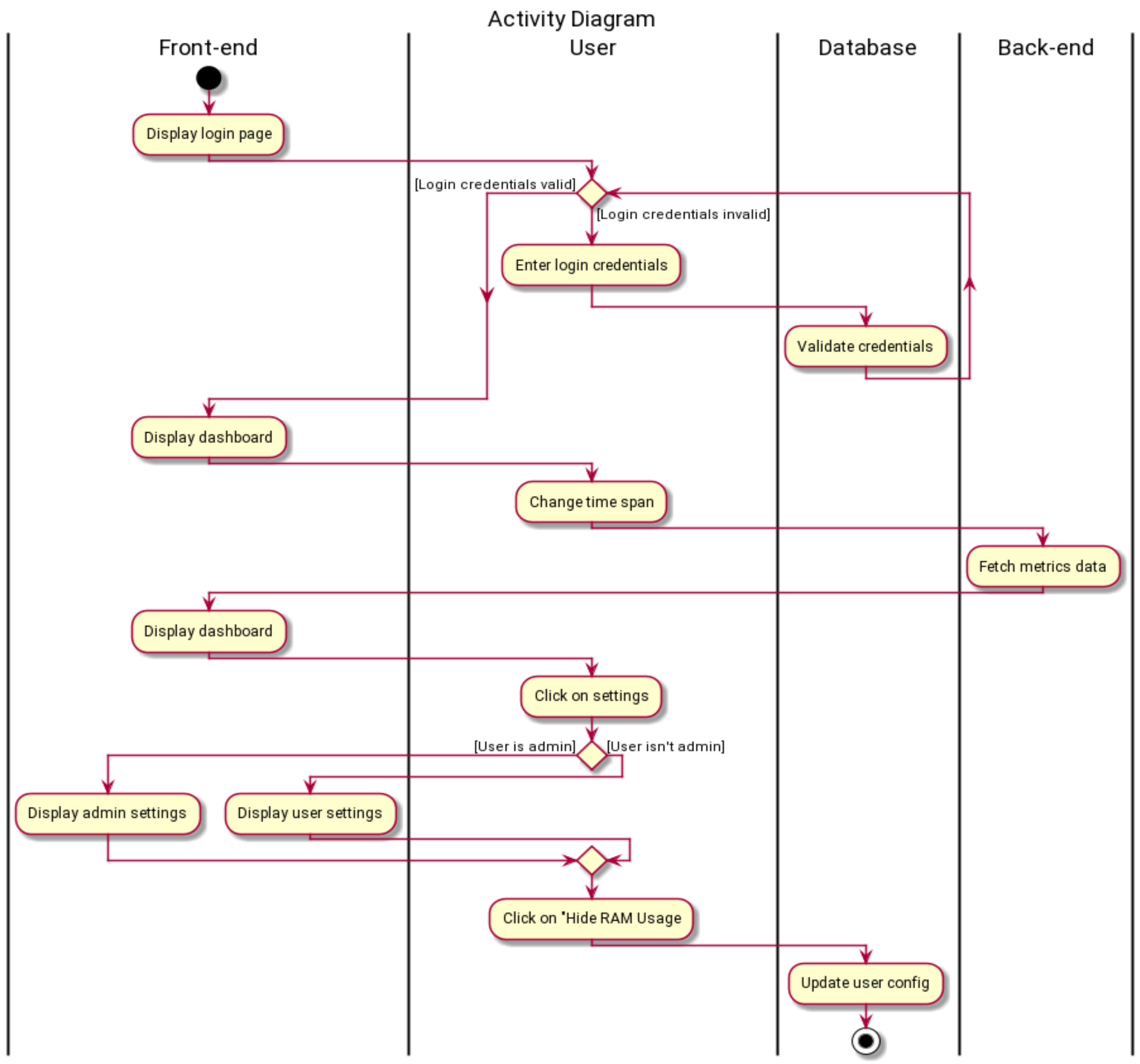


Figure 6: An activity diagram of the initial system design.