Scalable multi-platform medical system in a distributed environment

Caliang Xie

TU Delft

Scalable multi-platform medical system in a distributed environment

by

Caliang Xie

to obtain the degree of Bachelor of Science
in Computer Science and Engineering
at the Delft University of Technology,
to be defended publicly on Thurday March 7 March at 1:30 PM

Project duration:	November 26, 2018 – February 2, 2019	
Thesis committee:	PhD. L. Chen	TU Delft, Coach
	MSc. K. Rassels,	TU Delft, Supervisor
	MSc. O.W. Visser	TU Delft, supervisor

An electronic version of this Bachelor thesis is available at
http://repository.tudelft.nl/.

TU Delft

Preface

This report describes the completion of Bachelor End Project by Jason Xie. This project was completed in the second quarter of 2018-2019. Over the course of two and a half months, a backend distributed solution and a front-end for an administrator was developed for the multi-platform app that was developed by the Delft startup Eya Solutions. I wish to thank my coach Lydia Chen from Distributed Systems at the Delft University of Technology for her guidance and Kianoush Rassels from Eya Solutions for his help and guidance with the project.

Caliang Xie
Delft, January 2013

Summary

Eya Solutions commissioned a project to convert the, at the time, standalone system to a distributed system. This system should have been scalable so as to deal with an greater amount of users and clients. The system should also have been dependable as it supports an app that is to be used in the medical sector. If the system could not be relied upon then that might have major consequences such as the death of patients.

Furthermore Eya Solutions also commissioned the creation of a front-end for an admin along with several tools and functionalities that an admin should have access to for controlling the system.

The project was developed using as few external dependencies as possible so it costs less for migration.

Contents

1

Introduction

Eya Solutions is a start up in the medical field who have developed a multi-platform app for monitoring the health and well being of patients such as prematurely born babies and elderly. This app is capable of displaying data related to the patients that each caretaker is in charge of. The data include general data about a patient's personal information as well as periodic data that is recorded by a device worn by the patient.

The back-end system receives periodic data from these devices are stores it so users can query these data later on. The device is also capable of detecting whether an incident has occurred. When an incident has occurred, an alert is sent to the caretakers that are in charge of this patient, notifying them that an incident has occurred and thus a checkup is required.

All the data and alerts go through the system. It is essential that the system is reliable as a mistake that could lead to delay of several seconds might have dire consequences for a patient. The same should apply if number of users and patients increase.

At the beginning of this project, the system was only a standalone system. This would not be able to scale as the number of users *and* patients increases. An increase of patients will lead to an increase of write requests into the databases whereas an increase of users will lead to an increase of read requests of data in the ever growing databases.

To ensure that the system can scale up while maintaining its reliability, the current standalone system has to be changed into a distributed system using multiple web servers.

2

Related work

This chapter will present several works related to creation of a back-end system for the remote monitoring of patients. For instance, [6] discusses the possible opportunities and challenges that could with designing such a system.
An actual health monitoring framework is presented by [7]. In that paper they present a monitoring framework where they watermark the healthcare data before it is sent to the cloud. This would enable secure, safe and high-quality health remote health monitoring of patients.
This paper [4] discusses the possibility of using fog computing in health monitoring. They propose using smartphone devices as sensors. They see however that their proposal would lead to open access to all data in their system which would violate data privacy.
Another system is presented by [3]. In this paper the presented E-Health Care System uses cloud computing and web services for remote monitoring and controlling. Alerts are sent by SMTP (Simple Mail Transfer Protocol).
The authors of this paper [5] do not propose any new solution but opt to benchmark three different open source time series databases for scalability and reliability in a conceptual architecture that they designed. Their conceptual architecture is quite similar to ours we also have clients that will access the system through the web, make use of a time series database to store data and have devices that need to be monitored.

3

Problem overview

This chapter will define the problem that is to be solved in section 3.1. Additional requirements that the system is required to full fill are shown in section 3.2 This problem will be analyzed in section 3.3 to derive the practical requirements that are needed to solve the problem as well as the research questions that need to be answered. Finally, in section 3.4

3.1. Motivation

Eya Solutions old system 3.1 is running on a standalone. This is will not be able to satisfy the needs of Eya Solutions as the number of users and patients grow in the system. They would like a new system that can replace the old standalone system that they have. This new system should be scalable to support a greater amount of users and patients. The scaling of the system is not merely dependent on the number of users of patients. Each patient has to potentially wear a device that will periodically submit data to the system. This data needs to be stored and be query-able later.

Furthermore the system has to be dependable as a lapse in communication might have disastrous consequences for patients. The system should be able of coping with (partial) grid failures and able of having near 100% uptime for the clients.
These requirements can be fulfilled by designing a system that uses a load balancer to distribute traffic across nodes in the grid.

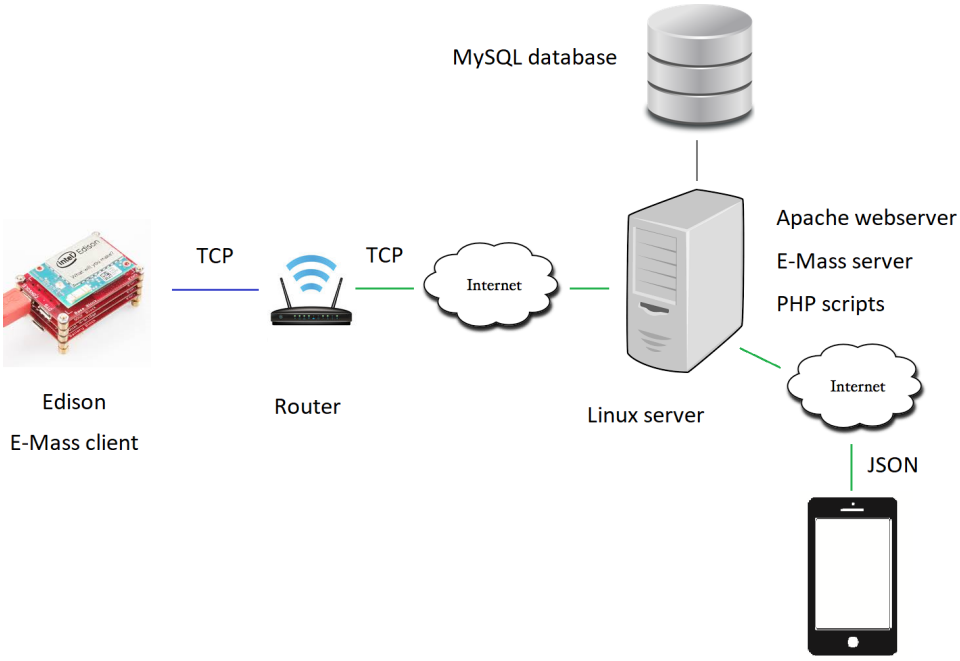


Figure 3.1: Old system diagram

3.2. Practical requirements

The requirements will be listed based on their priority using the MoSCoW method.

3.2.1. MUST haves

1. The system has to also work on a standalone
2. The system has to be able to spread the load across the nodes on the grid
3. The system should be able to monitor and show the grid's total storage, CPU-load, RAM-load and network usage
4. The system can display the same metrics of each individual node in the grid
5. The system has redundancy options in case a node fails
6. The system must have secure and reliable (medical standards) communication and data storage/processing

3.2.2. SHOULD haves

1. The system can switch between multiple load balancing algorithms
2. The system should be able to automatically expand the grid based on metrics of the system

3.2.3. COULD haves

1. The system should be able send an alert to a admin(s) in case the entire or part of the network fails

3.3. Problem analysis

The problem seems fairly straightforward initially. The current system needs to be expanded towards a distributed grid system. The major difference between the solution that is needed for this problem and existing solutions is that the central component of the system will need to be able to handle both read requests from clients as well as write requests from the devices.

3.4. Research questions

Based on the problem definition, requirements and problem analysis, the following question was determined to be the main research question to be answered:

3.4.1. How to load balance both read and write traffic?

Currently there are already many existing solutions to load balancing traffic across a grid but the majority of these load balancers usually deal with traffic of a single type, either read or write and not both.

3.4.2. How can scalability be achieved?

What will the system do in case it starts to reach its ceiling on CPU load, memory load or disk space? Will there be an automated system to that automatically adds in new nodes and assigns their roles?

3.4.3. Which algorithm will be used to distribute the traffic across the nodes?

There are current many different load balancing algorithms that vary in speed, complexity and resource usage. Which algorithm would be the best fit for this system?

3.4.4. How will the system be tested to prove it's capabilities?

The system needs to be able tested through simulations as bugs that occur during deployment could have disastrous results.

4

Research

This chapter will provide an overview of the research has been done for creating the back-end system. The first section 4.1 will discuss several load balancing algorithms. The next section 4.2 discusses the shortcomings of only relying on research.

4.1. Load balancing algorithms

To load balance means to distribute workloads across multiple computing resources. These computing resources can be computer processors, disk drives, computers or even clusters of computers. Using a load balancer can lead to several improvements such as improving response times, better resource utilization, etc. The algorithms that are used in load balancing can differ a lot from each other depending on their use cases.

This section will describe how several load balancing algorithms work and what their strengths and weaknesses are.

4.1.1. Round Robin

Round Robin is a very simplistic load balancing scheme. The first node for the first task can be selected at random but every task afterwards will be processed by the next node. This is very simple to set up and it is easy to understand how it works.
This algorithm could have fast response time in case every task takes the same amount of resources to complete and every node in the grid has the same specifications. Furthermore it does not take the current state of the grid/node into account.

4.1.2. Weighted Round Robin

Weighted Round Robin is very similar to regular Round Robin. The difference is that with Weighted Round Robin, each node is assigned a weight which is determined by the specifications of that node. Nodes with better processors, more memory or more disk storage space would get a higher weight which would lead to these nodes receiving more tasks [12].

4.1.3. Least Loaded

Another simplistic load balancing algorithm. This one simply takes the node that currently has the lowest load. The metric used for determining which node has lowest node could be the CPU load, memory load or the storage load.

4.1.4. Fuzzy Logic

Fuzzy Logic is the counterpart to binary logic. With binary logic a value can be only true or false. Fuzzy Logic on the other hand allows for multiple values. An example would be temperature. With binary logic you could only assign the value hot or cold based on a given temperature value. The transition between hot and cold would be abrupt and hard to place on the temperature scale. With Fuzzy Logic however you could have a range of possible outcome values as as cold, slightly cold, warm, slightly warm and hot. These values would be

placed on the same temperature scale but it would contain more gradual transitions.

Fuzzy Logic can be used in load balancing by applying it to several load metrics. These load metrics are fuzzified using a membership function [9] which returns the fuzzified value, see figure 4.1. An example of a membership function for the CPU load could return the values low, medium or high load. These membership functions can have multiple shapes such as triangular [11], trapezoidal [8] or Gaussian [10]. Other membership functions exist besides the three aforementioned ones.

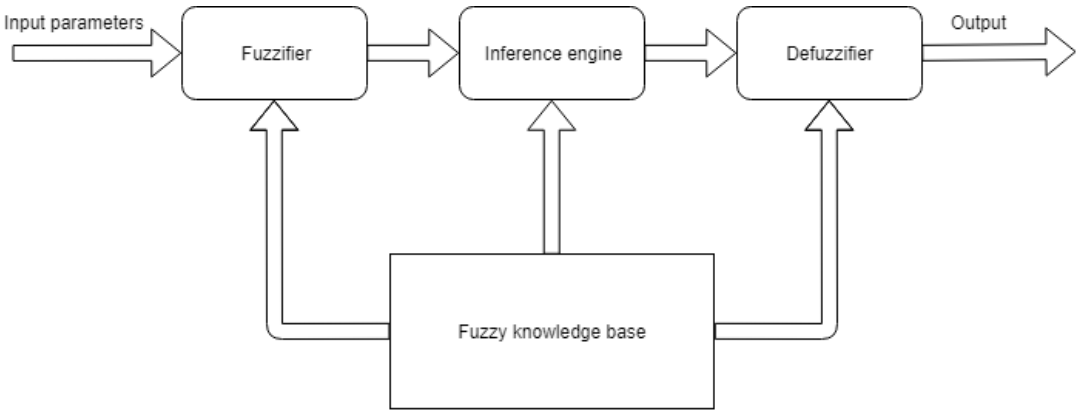


Figure 4.1: Fuzzy logic

All these values are defuzzified by applying rules from the fuzzy knowledge base to obtain a final crisp value between 0 and 1 which indicates the current load of the machine. This final value is used by the load balancing algorithm for deciding which node should process a given task. These rules are essentially If...Then statements that return a certain value for certain combinations.

An example:
If the *low_memory_load* AND *low_cpu_load* Then *low_node_load*
Else if the *low_memory_load* AND *medium_cpu_load* Then *low_node_load*
Else if the *medium_memory_load* AND *high_cpu_load* Then *medium_node_load*

While Fuzzy Logic is well suited to dealing with nonlinear systems, the performance and effectiveness of this algorithm depends on how it is fine tuned. Specifically which values correspond to which fuzzy values in the fuzzifying step and vice versa in the defuzzifying step.

4.2. System testing

Implementing a system based on a theoretical model is not always without problems. The new system needs to be able to be tested on a low level, through unit testing, as well as on a high level, by simulating real world simulations and checking how the system response to these simulations. Automated unit tests can allow one to quickly spot errors in the code without having to spend time going through the code. Unit testing by itself is not enough. It is also necessary to be able to test the entire system against a real world scenario (outside production). Example would be to test whether the system is capable of handling load from a thousand connections simultaneously sending requests to the system. Another thing that could be tested is whether the system will respond in a correct manner when a node fails. What happens to the rest of the requests? These issues will be discussed more in depth in chapter 8

5

Experiment results

This chapter will cover the results from experiments. The first section 5.1 will cover the setup for the experiments. The next section 5.2 will discuss and show the results of the experiments.

5.1. Experiments setup

The experiments were performed on nine VMs on the TU Delft super computer. The specifications for each vm were:

- Ubuntu 18.04.2 LTS
- 2 cores from Intel Xeon CPU E5-2683 v3
- 4GB memory
- 20 GB storage

Lastly a load generator was written for generating the load and measuring the latency of the requests. The load generator could generate load that were all read requests, write requests of mixed. The load generator would log in on ten user accounts that and randomly send requests to any resource that is normally available to a regular user. Each user would send out ten requests per second except when the load consisted of both read and write requests. In the case of mixed load, five requests of each load type would be send out each second.

A tool called stress was used for artificially increasing the CPU and memory load on the VMs.

5.2. Results

The goal of the experiments was to compare the performance of each the algorithms Fuzzy Logic, Round Robin and Random against each other across multiple scenarios. These scenarios are firstly split up across different load types. These load types were read only, write only and mixed. The load types indicate what kind of requests are sent to the load balancer to check the performance of.
Then each load type is split up into multiple subcategories that represented the different load levels of the grid. These subcategories are no stress and n stress - CPU/Memory. No stress meaning that there are no nodes that have been put under any artificial/external stress. N stress means that there are N nodes in the grid being stressed with N being an even number and evenly distributing the number of stressed nodes across the data node (relational and sensor data). CPU/memory indicates what kind stress is being applied to the machine.

The initial thoughts before experimentation started were that fuzzy logic would perform better when the grid is unbalanced. This could be either be because of non homogeneous system specifications, uneven distribution of tasks or other reasons.

Initial prediction before the experimentation actually started was that Fuzzy Logic would perform better than the other two algorithms if the grid was balanced. Fuzzy Logic would then perform the same or slightly worse on a grid with no nodes being stressed.

The charts for the results of the experiments can be seen below 5.2, 5.1, 5.3

5.2.1. Results analysis

Starting with the chart of the write only load 5.1 we can see that remarkably that the differing variables do not seem affect the latency of the write requests. The chosen algorithm and the load of the grid do not seem to affect the latency of write requests. This seems to indicate that InfluxDB is capable of consistently handling requests regardless of the current load of the machine.

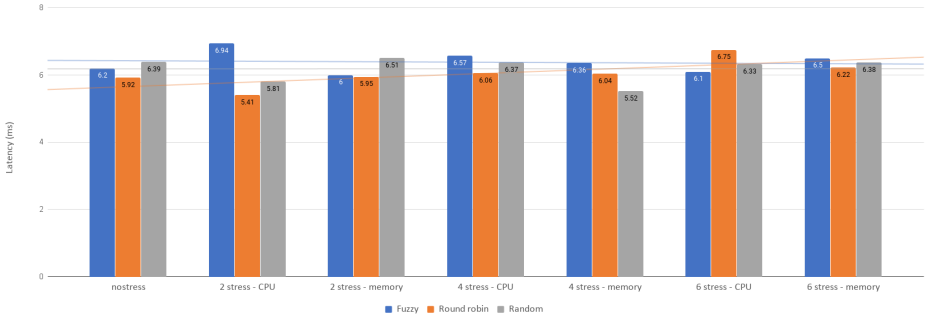


Figure 5.1: Chart of latency for write only load

The chart for read only load 5.2 paints a different story compared to the chart of write only load. In this chart the algorithm used for when the grid is not under any stress does not matter either. When the grid is under stress however, Fuzzy Logic seems to always perform better than the other two algorithms. The gap in latency of responses keeps widening as the number of stressed nodes increases.

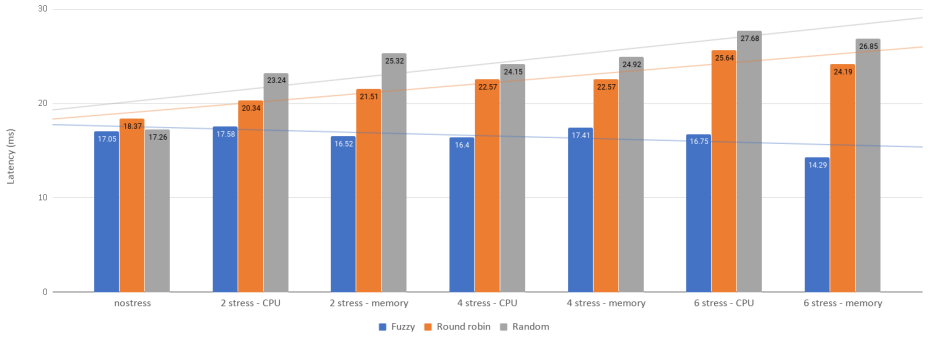


Figure 5.2: Chart of latency for read only load

The chart of the mixed load 5.3 paints a similar story to the chart of read only load on the right side and otherwise it looks similar to the chart of write only load. Initially it does not seem to matter which algorithm is being used for load balancing the request but when a large portion of the grid is being stressed, Fuzzy Logic comes out ahead.

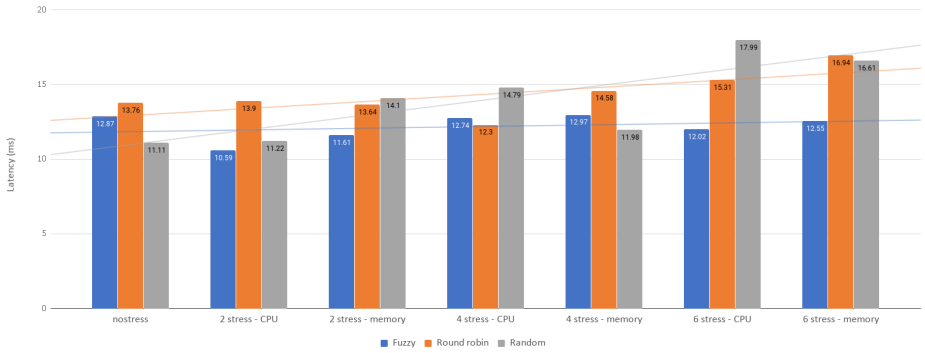


Figure 5.3: Chart of latency for mixed load

In conclusions, it seems that Fuzzy Logic performs considerably better than the other two algorithms when a large portion of the grid is being stressed and the load consists of read requests or a mix of read and write. In the case of write requests, it does not which algorithm is used but this could change in the future if the data to be written requires some preprocessing.
In future experimentations it is probably a better idea to take the variation of latency in the same situation into account as there indeed variations during experimentations. Furthermore, additional metrics could be compared besides only the latency.

6 Design

After the initial research was performed, designs for how the different parts of the project would end up looking like were made. While reading the papers for the research should have led to a general idea of what to do, it is generally better to first spend some time to clarify what things are needed. Section 6.3 what went into designing the GUI for the admin front-end. Section 6.5 will discuss and show how the design of the network topology came to be. Finally there will be an entire section dedicated to the design of the load balancer in section 6.6.1 as this component is instrumental to the entire system.

6.1. Development environment

All designs were made with running on a Lubuntu 18.04 machine in mind.

6.2. Database

The first thing that had to be done at start of the project was the migration of data from the MySQL to InfluxDB. This meant that a new database schema had to be designed as InfluxDB and MySQL do not have similar schema structures. MySQL is a relational database whereas InfluxDB is a time series database that does not have joins. The new database design had to keep most of the old data fields while being compatible with InfluxDB and taking advantage of it's strengths and weaknesses.

InfluxDB uses different terminology for certain concepts in MySQL. Refer to appedix F.1 for an small overview of the terminology from MySQL to InfluxDB.

InfluxDB is a time series database and is optimized for high write and query loads and for storing large amounts of time series data. Examples of times series data include sensor data, monitoring data and any kind of data about an topic that spans a (large) period. These data are typically indexed by their timestamp. In InfluxDB each point has a time value which is indexed. The timestamp used by InfluxDB uses nanoseconds precision. Besides the time column, each point (and thus each measurement) must also have at least one tag value and one field value. A tag is indexed while a field is not. InfluxDB recommends to not have too many unique tags as that may impact the performance.

The old database schema had two databases, one called eya_patient 6.1 and the second one called 6.3. The first database contained mostly data relating to users and patients whereas the second database contained mostly data relating to sensors from devices and the devices themselves.

After discussion with the client about what needed to be displayed on the admin front-end, a decision was made to add a third database to the new database schema. This third database would contain private data that are not supposed to be seen by users. The other design of the other two databases had to be modified in order to reduce the need of joins in the new database schema.

6.2.1. Design of new eya_patient

The first database called eya_patient was renamed to the EyaUsers for simplicity sake. The changes that were made from the old design to the new one were as follows.

The previous login table was renamed to user_logins with the username and an additional account_type as the tag fields and the rest as fields. The reason for the naming of user_logins instead of user or users is that both are already reserved keywords in InfluxDB. If that naming is desired, then it has to be escape by double quotes like ”user” or ”users” in every query and write operation. This was avoided by renaming the measurement to something else with the same meaning. With the inclusion of the account_type in this table, the previous role_type and role tables could be discarded.

The next change to be made was in the table of patients. In the old design the a patient only contained information about the patient itself whereas in the new design there are several newly added fields. These fields are the contact_ids fields which takes the place of the patient_medicalrelation table and medical_condition_ids which takes the place of medicalpatientinfo. Furthermore an additional field was added for the distinction between intramural and extramural patients. In both cases the residence_id field is used for referring to their residence. In the case of an extramural patient, the residence_id will refer to a point in the extramural_patient_addresses measurement where as in the case of an intramural patient it will refer to a room. A room in the new database design also has a field for the institution in which that room is located.

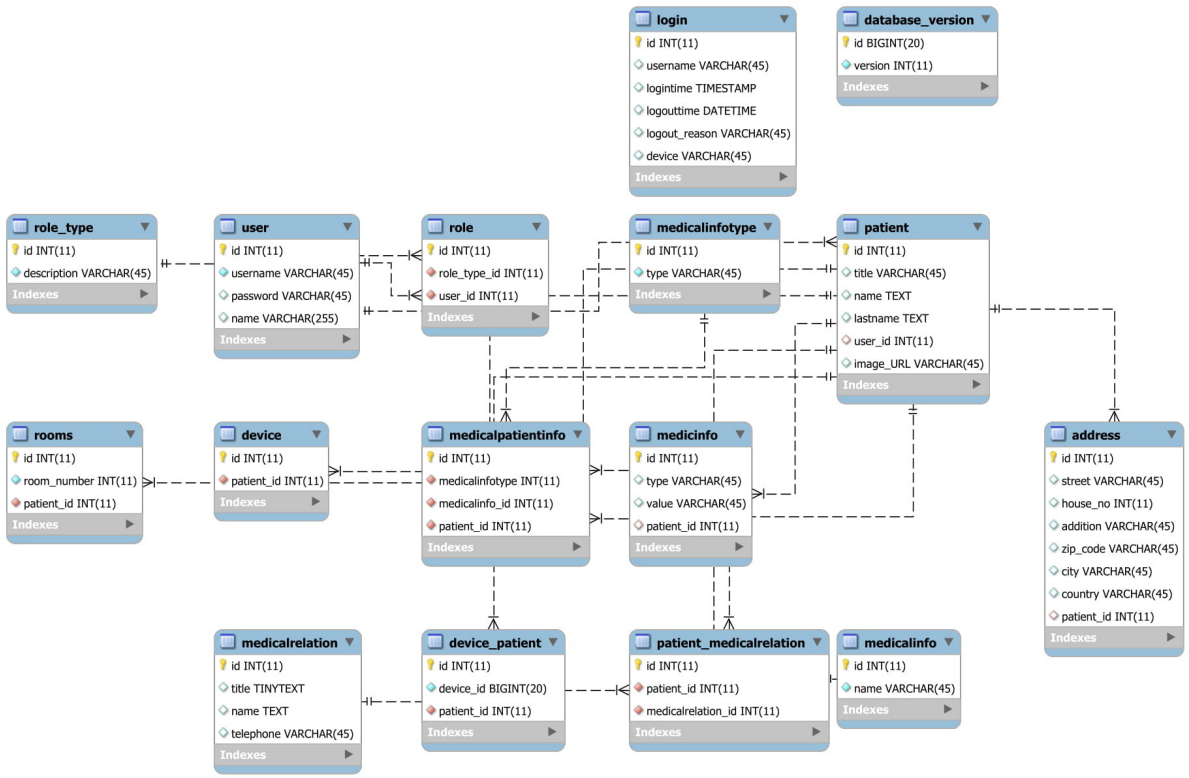


Figure 6.1: Database design of eya_patient

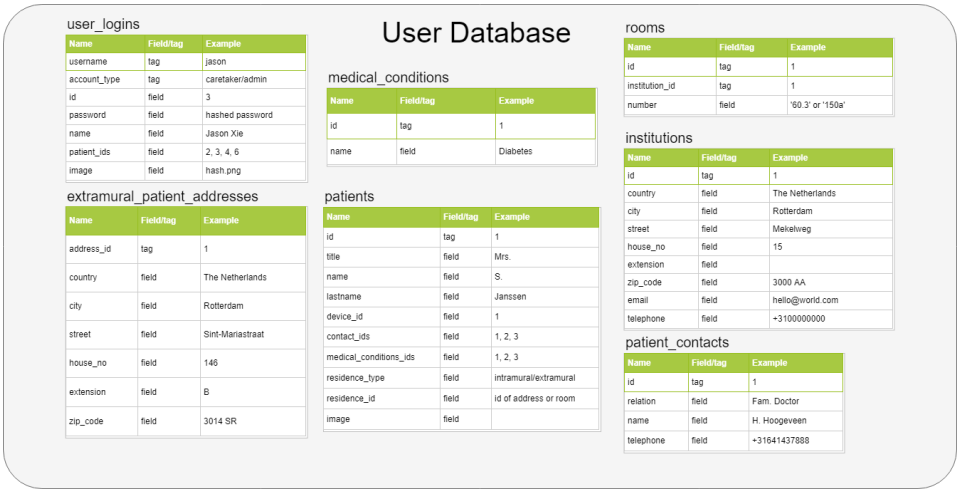


Figure 6.2: Database design for user data

6.2.2. Design of EMassDev

The second database was renamed to EyaSensor. After clarification from the client it was made clear that the tables in this second database contained data that were generated by the devices worn by a patient. The device sends two types of data: heartbeats and incidents. Heartbeats are send periodically and contain the temperature, WiFi signal strength, battery level, etc. These values were stored in the status and temperature table. Incidents on the other hands are only send by the device if the device detects that an incident has occurred. An incident can be that the patient fell or the patient's temperature dropped below a threshold. The data that is send in case of an incident are the same as of a heartbeat with the addition of the latitude (lat), longitude (lon), gyro and accel values. The gyro and accel values are measured in a three second window around the time of the incident and are thus not a single value but an array.

Based in this information, the decision was made to consolidate the status, temperature, gyro, accel, incident, incident_type and geolocation tables into the heartbeats, incidents and sensor_data measurements. Heartbeats would contain points that were originally in the status and temperature tables. Incidents would also contain points that were in the status and temperature tables the same timestamps as entries in the incident table. Furthermore the incidents measurement would also include fields for the lat and lon from the geolocation table. The gyro and accel tables would be merged together to form the sensor_data measurement.

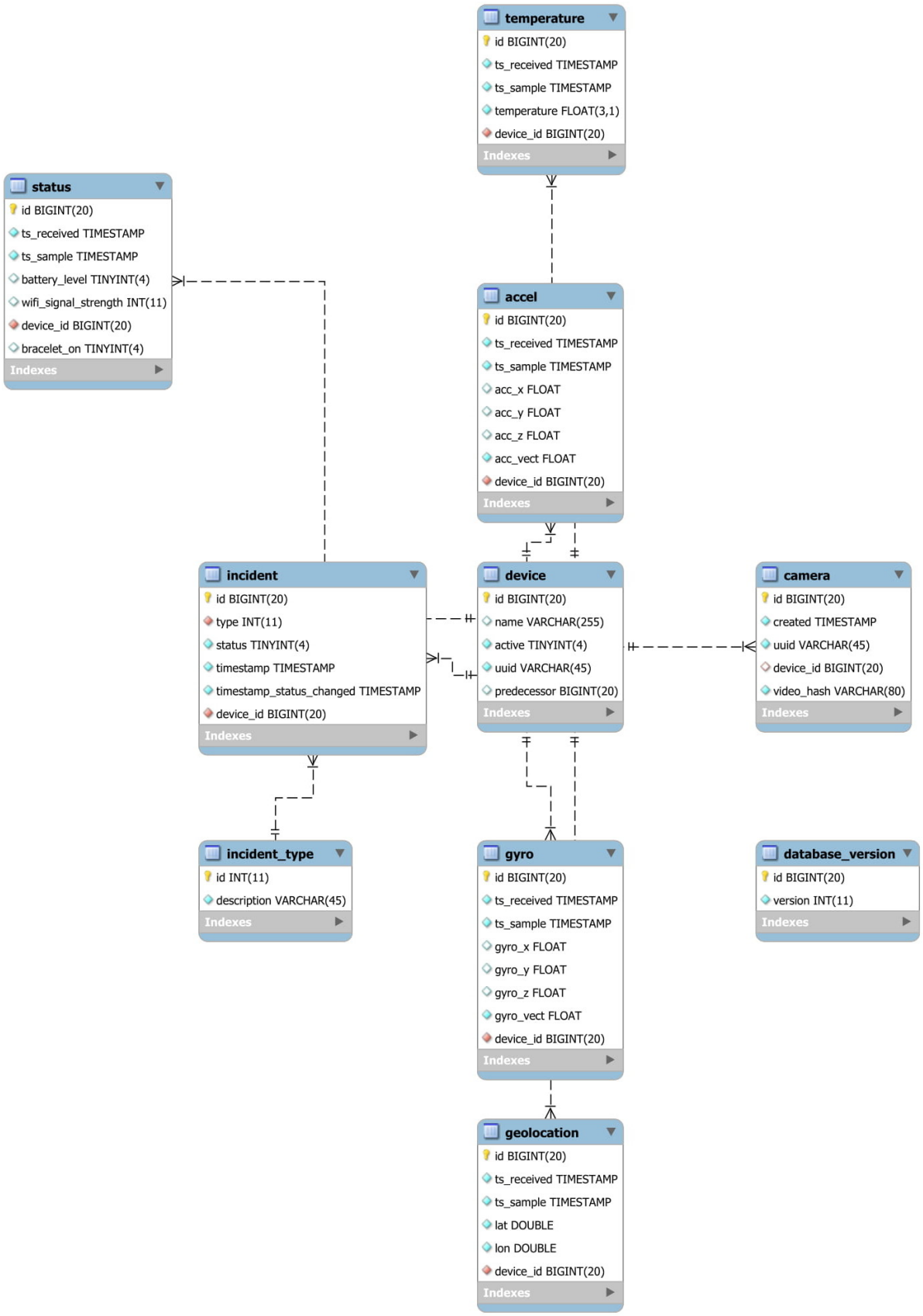


Figure 6.3: Database design of EMassDev

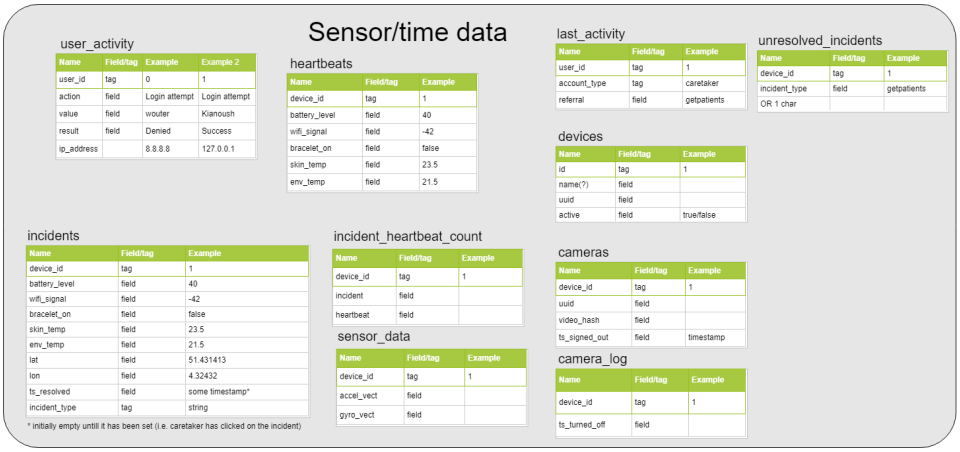


Figure 6.4: Database design for sensor data

6.2.3. Private/additional data

The design of the third database was fairly straightforward as it would contain additional information about clients (patients) and caretakers which are not shown to users. This data is instead only visible to the admin.

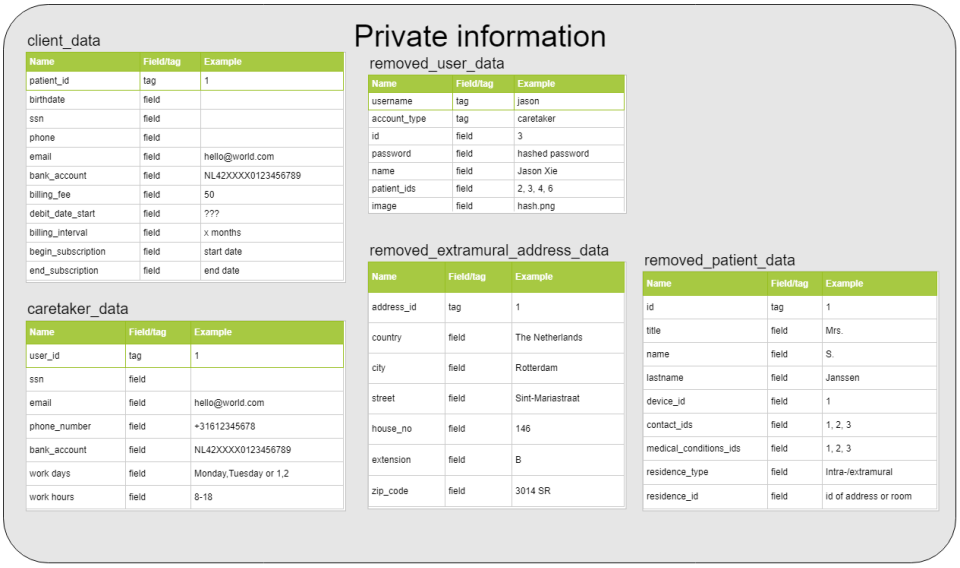


Figure 6.5: Database design for private data

6.3. Admin front-end

At the start of this project, the front-end for the client/user side already existed. One of the requirements for the admin front-end was that the Look and Feel of the admin front-end should be the same as the Look and Feel of the client front-end. There was thus not much time spend on writing the CSS rules and some JavaScript functions as these were already defined. What remained was listing what features were required in the admin front-end, how to display these features and finally how implement these features to give them functionality.

6.3.1. Requirements

From the original project description in appendix A, "... so a system administrator is able to add, remove or modify the patients or caretakers' information as well as his/her own information." Immediately two groups that shared similar functionalities could be found, namely patients and caretakers. After discussion with the client, it was clarified that there would

also be a third similar group, namely users. Both users and caretakers would be able to login domain and they also shared many similar level of access to data. The main difference being that a caretaker also has some additional information about oneself and a caretaker will also receive an alert through push notifications which they are able to sign off. This distinction did lead to affecting the client front-end and the database structure.

Besides being able to add/modify/remove these three groups, the client would also like that "The administrator should be able to perform real-time analytics, backup and restore of the data ..." through the admin front-end. Additionally the client would have also liked that an admin is able to view more general information about the three groups in the database(s). The metrics to be displayed were, number of online people per group, number of mutations (added/removed) since a given timestamp and number of incidents since a given timestamp. The choice was made to display all these data table. As for the ability of real-time analytics and the backup and restore of data, this would be shown in a tab of server tools.

6.3.2. GUI design

For the admin front-end, the client wanted tab button menu for selecting which information to display. The landing page shows a table of information about the number of online and offline people per group (users/caretakers/patients) as well as mutation data that can be queried by a calendar date. The other tabs were to show system tools, list of all patients, list of all caretakers and list of all users. The system tools page had to contain functionality for creating back ups of the database as well as the ability to restore a backup, changing the admin's information and displaying real-time monitoring data of the system. Each list of entities had to display all entities of that type. On each list it should be possible to add a new entity as well as remove the current entities. Each entity shows the profile picture and some basic information initially. Clicking on an entity would expand the tile and reveal all information about the entity. The designs of these pages can be seen in appendx G.

6.4. Push notifications

The push notifications were also part of the code that had to be migrated from the only MySQL database to the new one. The old push notifications was part of another older system however. As it was part of an older system, it seemed that the code relied on libraries that were deprecated and used multiple external libraries. For the new push notifications service system it was thus decided to just rewrite it from 'scratch' instead. The task of the original push notification service was to send push notifications to the client front-end when an incident has occurred. For this, it checked a table in the MySQL database for whether there were any incidents that were currently ongoing. This happened every second. The query made use of an indexed column to only retrieve incidents that were not yet resolved. That column field however is never ever used again afterwards. The flow of incidents data can be seen in 6.6. Devices (bracelets) send send a special message to the bracelet service when it detects that the patient has gone through an incident. From the bracelet service two entries are inserted into the database. One entry for the incident itself and another for for unresolved incidents. This second table is only used by the push notification service. At the push notification service, it will receive messages from the bracelet service. Upon receiving an incident, it will broadcast that incident data to all connected clients. Those clients will receive a push notification, if they are in charge of that patient, and will see that the patient in question will be highlighted. *Caretakers* are able to click on the patient to sign off the alert signal for that patient to all other users. Users on the other hands will not trigger this behaviour.
When an incident has signed off by a caretaker, its entry will be removed from the unresolved incidents table. The main use case of this table is in case the push notification service crashes, the table can be queried to see which incidents are currently still ongoing.

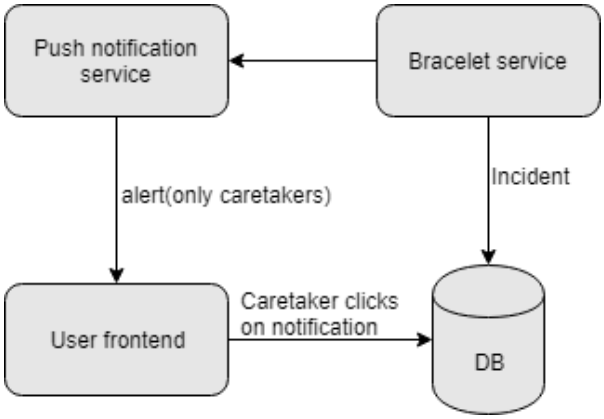


Figure 6.6: Flow of incidents and push notifications

6.5. Network topology

The load balancer is central in the network topology of the new system. Requests arrive at the load balancer and responses leave through the load balancer. These requests are forwarded to the data node that contain the data relevant to the request.
Inside the system there are services for push notifications and for receiving messages from bracelet devices. New heartbeats or incidents are submitted to the bracelet service from the device. The data is forwarded to the load balancer. A second separate message is sent from the bracelet service to the push notification service in case of incidents. Once this message arrives at the push notification service it is once again propagated to all connected clients that are in charge of the patient that was involved in the incident.

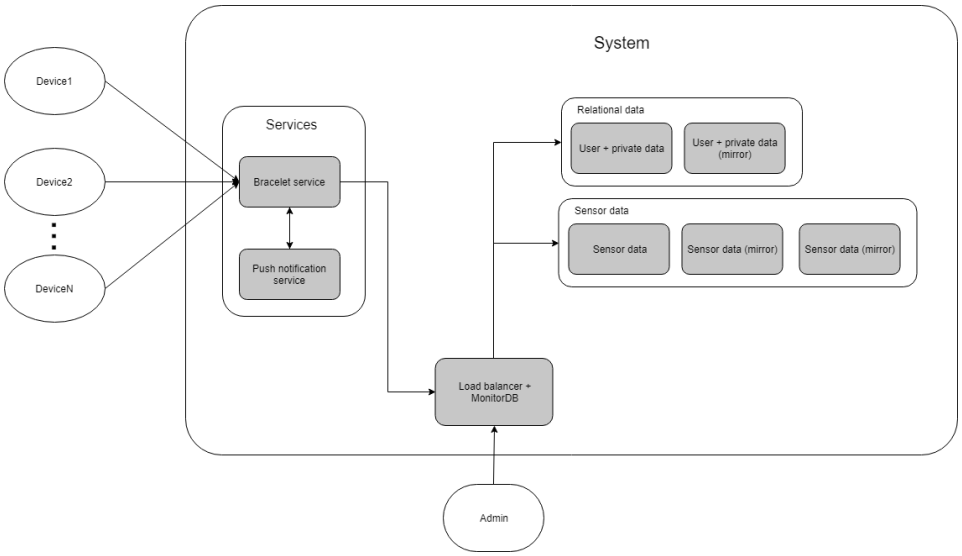


Figure 6.7: Network topology

6.6. Load balancing

The load balance is an integral part of the system. All read requests from the clients and write requests from the devices go through the load balancer first. The details of the design of the load balancer are discussed in section 6.6.1. In order for the load balancer to be load aware, the nodes need to be monitored. This will be discussed in section 6.6.2.

6.6.1. Load balancer

A request enters the load balancer where it will go through several steps, see figure X. The first step validates the request to see whether it is valid. The validation checks whether the client is logged in in case of query requests and whether the path leads to a content that is supposed to be served.
The next step is selecting the node that will process the request. At this step the path of the request is taken into account. First the database that is required for the request is determined. Afterwards the least loaded node will be selected to handle the request. This node is returned and passed on to the next step.
The request forwarding is straightforward. It will forward the client request from the load balancer to the actual node. The response is returned to the load balancer and then in turn returned to the client.

6.6.2. Node monitoring

The load balancer keeps track of the load of each node in the system. These load values are calculated locally by using fuzzy logic on the metrics of the node. The calculation of the node load has been kept on the node to alleviate the load on the load balancer. The nodes are monitored continuously but the load values are only sent to the load balancer if they have been changed to lower bandwidth use.

7 Implementation

This chapter will build upon the designs shown in chapter 6 for the implementation of each part.

7.1. Database migration

The migration of MySQL to InfluxDB was initially performed using a python script that parses the SQL file into a text file with a format that is compatible with InfluxDB. After this initial conversion, a PHP script was executed that properly restructured the data according to the design in section 6.2 and figures 6.2, 6.5 and 6.4. This required the use of the InfluxDB-PHP module that was installed through Composer.

7.2. Middleware

After the data had been migrated from MySQL to InfluxDB according to the new schema in 6.2, the middleware scripts had to be modified. The old PH scripts used mysqli to query the data from the database. In order to query the data from the new InfluxDB database a similar library had to be used. The scripts were then modified to query the data from the new database. The queries stayed largely the same as the query language of InfluxDB is very SQL like. The major difference being that there are no joins and data is returned in what InfluxDB calls *series*. A series is simply a collection of data with the same tags [1].

7.3. Admin front-end

Since the client front-end was already in place with a CSS file, the issue of Look and Feel of the admin front-end was somewhat already taken care of.

7.3.1. Dashboard table

The dashboard table contains a mix of 'static' and mutation data. The static data is data returned by queries which do not change where as mutation data use an extra timestamp as input for the query. The timestamp for the query is selected by a date picker that shows up when an user clicks on the calendar icon.

As soon as a date has been selected, the query will be executed and update the values in the cells belonging to that date picker.

7.3.2. System tools

For the remote backup and restore functionality on the admin GUI, the backup and restore commands of InfluxDB were used E. These commands have have several options which are displayed on the admin GUI. Since InfluxDB cannot create restores to an existing database (with the same name) the name of the new restore had to be taken into account by checking for a response from InfluxDB.

The modification of an admin's information is pretty much the same as modifying an entity in the list of entities (users, caretakers or patients).

Displaying real-time monitoring data is done by creating a websocket connection between the admin's web browser client and each node in the grid. Much like the entities, each node can be expanded to display more details so the page is not immediately clogged with information.

7.3.3. List of entities

Each entity is displayed as a card that can be expanded by clicking on it. Each tab contains it's own template for filling in the fields belonging to that entity.

7.4. Push notification

The push notification was written in JavaScript and runs in the Node.js environment. The reason for picking Node.js is because this service needs to be able to send messages towards a client and be able of opening many connections simultaneously.

The push notification service starts up by querying the database for all entries in the unresolved_incidents measurement. The service also starts up a websocket server which will send a list of unresolved_incidents to new connections. A client can also submit messages to the service for signing off incidents. These messages will only be processed if they have submitted the proper handshake. If they are caretakers then a hashed value is returned to the client which has to be send to the service in order to be able to sign off incidents.

The push notification also has a HTTP server that listens for requests send by the bracelet service in case of new incidents. New incidents are added to the in memory list of current unresolved incidents. These new incidents are also broadcast to all open connections.

A point that has to be made with regards to the implementation, currently the push notifications can only be send if there is an open connection. Closing the tab or app will result in the connection being broken (not sure yet for mobile). There is no service running in the background that handles the push notifications. The reason for this is that the existing Push API is not supported by all modern web browsers. The Push API furthermore requires the use of a *push service*. The push notification service on the server side sends push notifications to this push service and in turn this push service sends the push notifications to the clients. The choice was thus to make use of a third party push service, like OneSignal, Firebase Cloud Messaging or others, implementing a push service ourselves or create a separate service in Java that can handle the push notifications on mobile. The former was immediately denied by the client as he does not wish to make use of a third party service due to the confidentiality of the data and the former two options were not feasible duo to the time constraints of this project.

7.4.1. Client side

On the client side, websockets from the WebSocket API were used for opening a duplex connection with the push notification service. Looking at the websockets on the caniuse.com website, it is shown that the vast majority of modern web *and* mobile browsers support this feature. The only difficulty that was found during implementation, was figuring out how to make sure that only a caretaker would be capable of signing off alerts send by the push notification service. This was later resolved by adding in a handshake. Upon logging in, a hash value is generated and send to the client and the push notification service (if the user is a caretaker). If the handshake is correct then the caretaker will be able to sign off push notifications by sending messages over the websocket connection. Otherwise any message send by the client over the connection is ignored.

7.4.2. Server side

On the server side, the ws library was used for opening duplex connections with the client. There two most popular Node.js libraries for websockets were the ws [2] module and socket.io. The former was chosen as the extra functionality provided by that library was not needed and

the extra performance of the latter was more desirable.
Another alternative was the µWebSockets.js module. This module had significantly higher performance than any other Node.js modules. The reason of the significantly higher performance is because it is written in C and C++. The µWebSockets.js module makes use of the C++ module through Google V8 bindings. Due to concerns regarding the dependability of the module it was decided to use the ws module instead. The concerns were a result of the developer removing the source from npm resulting in all projects that depended on that module to break.

7.5. Load balancing

Just like the push notification service, the two components that make up the load balancing; the load balancer 7.5.1 and the node monitoring 7.5.2 are Node.js applications written in JavaScript.

7.5.1. Load balancer

The load balancer makes use of the https module for receiving the requests as well as for creating the forward requests to the nodes in the grid.
The load balancer also makes use of the ws module which is used for opening connection between the nodes and the load balancer. This connection is for transmitting the calculated load values from the node to the load balancer and for sending over the response time of the requests from the load balancer to the node.

7.5.2. Monitoring

Initially the plan was to use the monitoring function of PM2, a process manager for managing Node.js applications. Upon investigating the documentation of PM2 it was revealed that PM2 does indeed have monitoring functionality of a node but these monitoring values could not be extracted for processing. Upon checking the repository of the monitoring module however, it was shown that internally PM2 makes use of several Node.js modules as well as some commands to retrieve the metrics shown in monitoring. That is thus what was also done for the monitoring of nodes in the back-end system.

The metrics that are being monitored are the:

- CPU load
- Memory load
- Network bandwidth
- Disk usage

For the monitoring the CPU and memory the built in OS module of Node.js was used. Specifically the functions os.cpus(), os.freemem() and os.totalmem() were used.

For monitoring the network bandwidth and disk usage external commands were used. These commands were executed by the exec command of the child_process module of Node.js The commands that were executed were ’df -kP’ and ’ifconfig’.

8

Software Quality and Testing

The platform that is going to be developed will be used to test the performance, data integrity and maximum load of a distributed version of a standalone system. The platform itself is to be used as load balancer and central monitoring and managing system. The environment of the platform consists of the web servers, that can either be used in standalone mode, or used by the controller in a distributed system.

8.1. Software quality

During the project, the software quality of the code was kept in mind by focusing on the the maintainability through documentation and reliability through reliance of fewer external modules.

8.2. Testing

Another important part of software quality is testing. Testing allows one to discover bugs that were introduced through the addition of new code or modification of existing code. Writing automated tests allows one to discover bugs before they can be shipped to production.

8.2.1. Unit testing

When testing software, one has to start at the smallest component of the program. Writing tests for these components is called unit testing and revolves around testing the small components such as methods and classes.
Since the main programming languages used in this project are JavaScript and PHP it is easy to find a good testing framework as they are both very mature languages. For PHP the testing framework PHPUnit was used to test code and Mocha JS was used for unit testing frontend JavaScript.

8.2.2. System testing

Unit tests alone are not sufficient for testing a system. Therefore there will also be several test cases for simulating real world traffic. This will done through the use of a load generator that will simulate requests from clients and data send by devices. This has already been discussed in a previous chapter 5.

9 Reflection

This chapter discusses how the project went. In the first section 9.1 there will some insights on how the project went. The next section 9.2 will discuss the cooperation with the client while the section that follows after that discusses the cooperation with the coach 9.3. The final section 9.4 will list several ideas for future work or research.

9.1. General reflection

I am happy with how the final product turned out and I managed to learn new things during the project. I learned PHP and improved my knowledge of JavaScript, Node.js and Web APIs. Because the the client's request for the project to stay easy to migrate as well as simple, I tried to have as few external dependencies as possible. For things that I had to develop for the front-end I always checking whether there were any existing Web APIs existed and whether these APIs were supported by enough browsers on both desktop and mobile. On the back-end I stuck to mainly using popular dependencies for Node.js. Furthermore I also learned quite a lot about general system architectures and how components are supposed to communicated with each other.
One area that could have been improved is the planning. In the beginning I was a bit lost as to how to start the project which led to some time being lost. Thankfully I managed to pull things together and prevent myself from falling behind schedule for the rest of the project.
Another thing that could have been improved was testing as I did not have any tests up until the first feedback from SIG. This was a major mistake on my part. I did manage to write some tests in the end but there was not enough time to be able to write automated tests for the entire system.
It was unfortunate for me that I was late in looking for a group and thus had to do this project on my own. I could not afford to spend too much time researching all parts in-depth. This led to the scope of the research becoming quite narrow. It would have been really nice if I had a team member with whom I could discuss things.

9.2. Cooperation with the client

The cooperation with the client was really good and I am very happy with him. Because this project was originally intended for a group, it was modified at the last minute so a single student could work on it. Kianoush was very cooperative and we often had meetings in person, over Skype or TeamViewer. He was thus usually aware of what my current state was.

9.3. Cooperation with the coach

My coach was a bit busy at the start of the project so the meetings were not regular but they became regular after things settled down.
Our coach was really helpful is directing the research part of this project. At the beginning I

was trying to tackle a problem that was too broad and the coach helped narrow the scope of the research a lot.

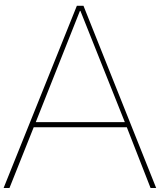
9.4. Future work / research

This section will discuss ideas for future work and research that have not been done yet (in this project).
While the final result turned out to be okay, there are still areas for improvement. Writing automated tests for every component of the system would be among one of the things with the highest priority. Another would be to add the ability to auto-scale to the system in case the system starts to hit a threshold in performance or storage.
Another idea would be to implement a service worker for the push notifications as currently the push notifications will not work if the browser or app is closed.

10

Conclusion

The goal at the beginning of the project was to develop a distributed back-end system along with creating a front-end for an administrator and modifying the old middleware to make use of a new database. These goals have been achieved. The load balancer still has room for improvement and there are still some functionalities that could be added to the admin front-end but the baseline has been achieved. Because of the small number of external dependencies, the product is easy to migrate and deploy. The steps to deploying a new node has been documented. The code in general has also documented so it is easy to maintain.
I hope that this product will be a positive contribution to the medical field in helping with monitoring patients. I have learned how important it is to document things extensively as well as to communicate clearly.



Original project description

The student will create a HTML 5 front-end platform supporting popular browsers and mobile devices, so a system administrator is able to add, remove or modify the patients or caretakers' information as well as his/her own information. The administrator should be able to perform real-time analytics, backup and restore of the data without losing data integrity and performance in a distributed environment. The system should not introduce any downtime for the system serving the clients.

The student will migrate the backend (mysql database) to an InfluxDB database along with the middleware. The middleware and the database need to be improved from the performance and efficiency point of view and support distributed environment en grid computing for analytics.

The student will investigate and implement the concurrent connections, load balancing, the response time of the system and push notification with respect to server CPU load, available internal memory and hard disk along with number of active connections with the database.

The patients and the caretakers must be able to access their data based on their credentials regardless of where their information has been stored physically (on another hardware) in the grid. The application must be easy to move/migrate from one hardware to another in real-time.

Summarized, the end product should work intuitive and efficient in a distributed environment, which supports secured data transmission, is scalable in real-time and easy to migrate to another hardware. The system has to be tested for performance, load balancing and data access speed.

Documentation of the code, deployment and test results is part of this assignment!

B

Infosheet

Title of the project: Scalable Multi-platform administrator tool in a distributed environment
Name of the client organization: Eya Solutions
Date of the final presentation: 7 March 2019 at 1:30PM

Description
At the end of this project the original data was migrated over to InfluxDB from MySQL, middleware scripts were fixed, a front-end for administrators was created and a load balancer has been implemented.

Challenge
The main challenge of this project was to create a load balancer with high throughput while capable of being load aware.

Research
During the research phase I learned that I should try to tackle broad topics. Research problems should be defined crisp and clearly so that research.

Process
The process went well. There was a lot communication between the client and the user over Skype, Teamviewer or WhatsApp. There were also weekly meetings with the client and the coach. The client expected more communication in the beginning which stayed for the remainder of the project.

Product
The final product consists of a distributed back-end system and an admin front-end. Part of the distributed back-end were the load balancer, push notification service, web servers with the middleware and node monitoring. The front-end consisted of the admin-frontend, with functionality, and some scripts for the user front-end for receiving push notifications.
The majority was manually tested unfortunately. The only part that has written test are some middleware scripts.

Outlook
There were some suggestions made for the system such as changing the flow of data from sensors to the system in case of emergencies. This suggestion was partially implemented on the push notification service but it was not complete as the device was not part of the project. The client took note of the suggestion however and plans on implementing the suggestion when the device gets reworked.
The product will end up being used.

Member of the project team
Name: Caliang Xie
Interests: Reading
Contribution and role: Researcher and programmer

Client

Name: Kianoush Rassels
Affiliation: Eya Solutions

Coach

Name: Lydia Chen
Affiliation: Distributed Systems

Contact

Caliang Xie, caliangxie@gmail.com

The final report for this report can be found at: `http://repository.tudelft.nl`

C

Software Improvement Group (SIG)

During the project, a zip containing the source code of some files in the project were submitted to SIG. They responded with the following feedback.

C.1. First feedback

[Analyse]

The source code of your project scores about 3 stars in our quality model for maintainability, meaning that it is around market average. You did not receive a higher rating due to lower scores for Unit Size and Unit Complexity.

For Unit Size we look at the number of lines of code within a method. Large methods tend to contain too much functionality and too many responsibilities. By splitting up these responsibilities you can make the individual parts easier to understand.

In your project we see that little code structure is used within the PHP code. Files like remove_caretaker.php perform multiple tasks, but all of the code is put inside the root of the file without creating any classes or functions. This makes it impossible to test the different parts individually. Different parts of the code, like the part that populates $userfields, should be moved into a separate method (e.g. getUserFields()) to keep the code maintainable.

With Unit Complexity we consider the number of decision points within a method. If the method has many possible outcomes it becomes increasingly hard to determine the impact of changes. Also, unit test cases will need to create for all possible paths.

In your project, the problem of files lacking structure also leads to high complexity. In other cases the files do contain structure, but the methods are too generic and therefore still contain a lot of functionality. An example is submit() in admin_framework.js. This method contains two clear blocks of functionality that are separated by comments: performing input validation on the form, and then using the data from the form to send an AJAX request. By separating these functionalities you would improve both the Unit Size and Unit Complexity.

As a last comment, we have not found any unit test code within your upload. Both PHP and JavaScript have mature test frameworks, so it would be good to add tests in the remainder of the project. This will also help you to avoid introducing bugs when you change the code in the future.

In summary, your code is currently around market average, but if you spend some time to focus on these improvement areas in the remainder of the project should be able to achieve significant improvements without spending too much effort.

C.2. Addressing the feedback

There was a lot of room for improvement in improving the quality through adding more structure in the files and code "... the problem of files lacking structure also leads to high complexity." as well as just writing tests in general.
Firstly the point made about unit size and complexity would be addressed by splitting up and restructuring function where applicable. After that would come the writing of tests. These tests could be split up into unit tests on the back-end and application tests for the front-end.

C.3. Second feedback

> [Remeasurement] In the second upload we can see that your project has grown significantly, the codebase is now 50% larger when compared to our first analysis. We also see a clear and significant improvement in the maintainability of the code.
>
> We previously mentioned Unit Size and Unit Complexity as areas that could be improved. For both we see that you implemented our recommendations, while also ensuring that the new code did not lead to new violations.
>
> We can also see that you followed our recommendation regarding test code, and added unit tests. Obviously, because you did not create tests from the beginning of the project the amount of test code is still quite low when compared to the production code. Still, we again see clear improvements in this area when compared to the first uploads.
>
> From these observations we can conclude that you have followed our recommendations during the development process.

D

Deployment procedure

The deployment of the program requires the installation of software on a (virtual) Ubuntu machine.

D.1. Installing InfluxDB

sudo apt-get install influxdb

D.2. Installing nginx

First, install the web server nginx

```
apt-get update
apt-get install nginx
```

D.3. Installing PHP

Afterwards install a PHP processor for nginx

```
sudo apt-get install php-fpm
```

After installation we need to configure php-fpm. The configuration file is located at /etc/php/<version>/fpm/ph
Open this file with a text editor and modify the cgi.fix_pathinfo parameter by uncommenting it (removing at the beginning of the line) and setting it's value to 0.

After saving, the PHP processor will have to be restarted with the following command:

```
sudo systemctl restart php<version>-fpm
```

D.4. Configuring nginx

Nginx' config file is located at /etc/nginx/site-available/default. The resulting config should look like this.

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html/;

        index index.php;

        server_name _;
```

```
        location / {
                try_files $uri $uri/ =404;
        }

        location ~ \.php{
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
        }

        location ~ /\.ht {
        deny all;
    }
}
```

After making your changes, check whether there are any errors in the config file by running

```
sudo nginx -t
```

If there aren't any, then restart nginx to load the new config with the following command.

```
sudo systemctl restart nginx
```

D.5. Installing composer and InfluxDB-PHP

Install composer through.

```
apt-get install composer
```

Following that install InfluxDB-PHP *in* the client frontend directory. This directory contains the composer.json file which tells composer which dependencies to install.

```
composer install

sudo apt-get install ethtool
```

D.6. List of used ports

Insert table of applications and the used ports (and notes/description?)

Application	**Port**
InfluxDB queries	8086
InfluxDB backup and restore	8088
Push notification websocket	8080
Push notification incidents receiver	8081

E

InfluxDB backup and restore commands

E.1. Backup

To create a backup of one or multiple databases one needs to execute a command with the following syntax.

```
influxd backup
    [-portable]
    [-database <db_name>]
    [-host <host:port>]
    [-retention <rp_name>] | [-shard <shard_ID> -retention <rp_name>]
    [-start <timestamp> [-end <timestamp>] | -since <timestamp>]
    <path-to-backup>
```

Square brackets indicate optional arguments where as angled brackets indicate that it is required. Example of some valid backup commands.

```
influxd backup <path-to-backup>
influxd backup -portable <path-to-backup>
influxd backup -portable -db mydb <path-to-backup>
influxd backup -portable -db=mydb <path-to-backup>
influxd backup -db mydb -portable <path-to-backup>
```

portable If this argument is set, then the backup will be created using a newer format that is compatible with InfluxDB Enterprise. It is highly recommended to use this argument.

database Use this argument if a backup of a specific database has to be created. If it is not specified, then all databases will be backed up. -db can be used instead of -databases

host The host and the port of the InfluxDB instance. By default it is set to '127.0.0.1:8088' meaning that it will connect to the localhost on port 8088, the default port for requesting backups and restores.

retention The retention policy for the backup. If it is not specified then the default behaviour is to use all retention policies. If this argument is specified, then -database has to be specified as well. -rp can be used instead of -retetion

shard Shard ID of the shard to be backed up. If specified, then -retention is required.

start Include all points after the given timestamp. The timestamp has to be in RFC3339 format.
Example: -start 2019-01-31T12:12:12Z

end Exclude all points after the given timestamp. The timestamp has to be in RFC3339 format.
Example: -end 2019-01-31T12:12:12Z

since

path-to-backup The path to where the backup will be placed alwas has to be specified. The path can be relative or absolute.
Example: ./backup1 or /var/www/html/backups/backup2

E.2. Restore

To restore one or multiple databases one needs to execute a command with the following syntax.

```
influxd restore
    [-portable]
    [-database <db_name>]
    [-newdb <db_name>]
    [-host <host:port>]
    [-retention <rp_name>] | [-shard <shard_ID> -retention <rp_name>]
    [-newrp <rp_name>]
    [-start <timestamp> [-end <timestamp>] | -since <timestamp>]
    <path-to-backup>
```

Square brackets indicate optional arguments where as angled brackets indicate that it is required. Example of some valid backup commands.

```
influxd restore <path-to-backup>
influxd restore -portable <path-to-backup>
influxd restore -portable -db mydb <path-to-backup>
influxd restore -portable -db=mydb <path-to-backup>
influxd restore -db mydb -portable <path-to-backup>
```

portable If this argument is set, then the backup will be created using a newer format that is compatible with InfluxDB Enterprise. It is highly recommended to use this argument.

database Use this argument if specific database has to be restored. If it is not specified, then all databases will be restored.

newdb Name of the database into which the backup data should be imported to. If not specified, then the value of -db will be used. The new database name must be unique to the target InfluxDB instance, meaning that it cannot already exist.

host The host and the port of the InfluxDB instance. By default it is set to '127.0.0.1:8088' meaning that it will connect to the localhost on port 8088, the default port for requesting backups and restores.

retention The retention policy for the backup. If it is not specified then the default behaviour is to use all retention policies. If this argument is specified, then -database has to be specified as well.

newrp The name of the retention policy that will be created on the target InfluxDB instance. Requires -rp to be set. If -newrp is not set then the value of -rp will be used.

shard Shard ID of the shard to be backed up. If specified, then -db and -rp are required.

path-to-backup The path to where the backup will be placed alwas has to be specified. The path can be relative or absolute.
Example: ./backup1 or /var/www/html/backups/backup2

F

Glossary of terms

F.1. MySQL versus InfluxDB terminology

MySQL	**InfluxDB**
Row/entry	Point
Table	Measurement
Unindexed column	Field
Indexed column	Tag

G

GUI

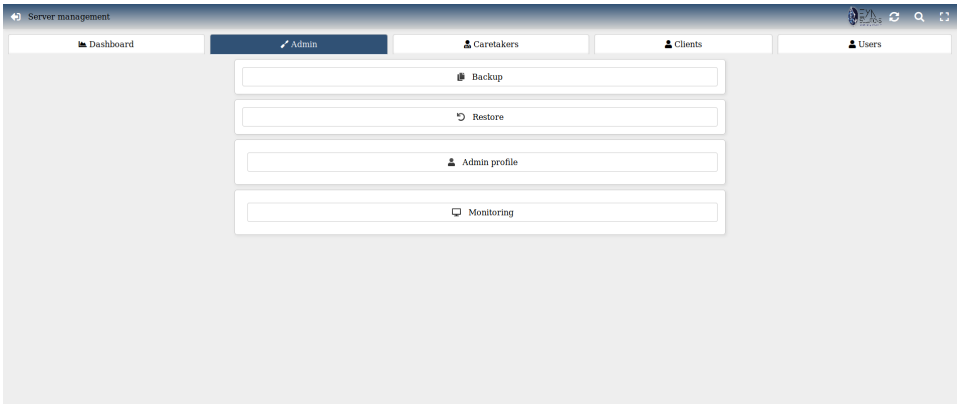


Figure G.1: Admin tools page

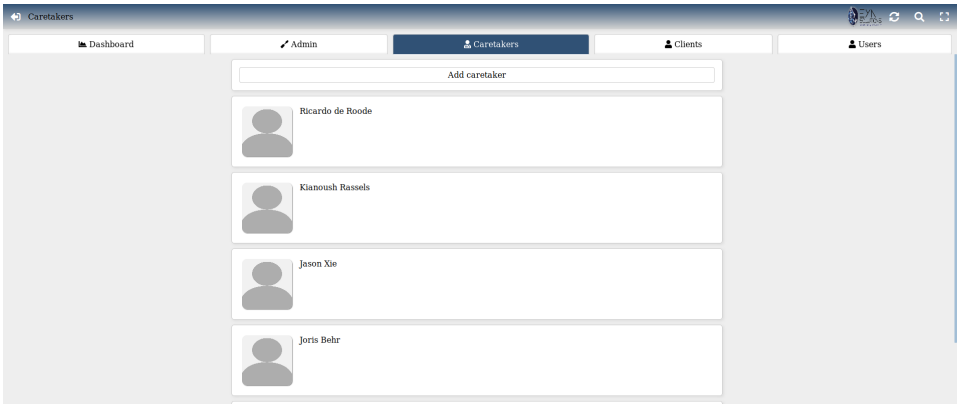


Figure G.2: List of caretakers page

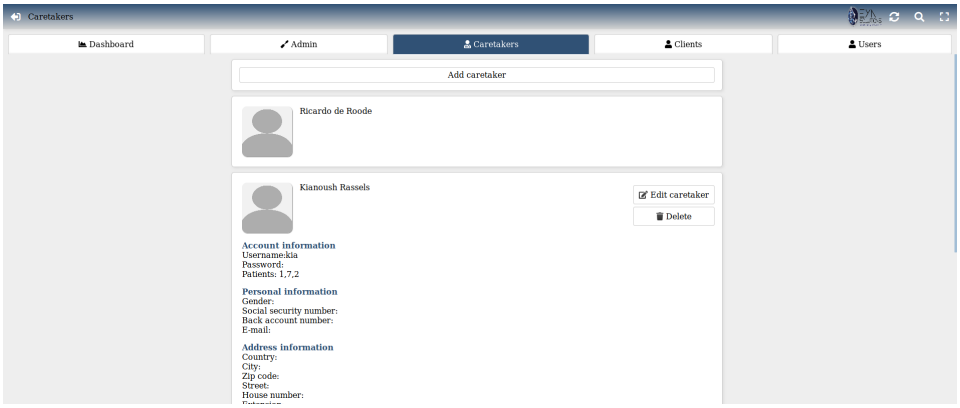


Figure G.3: Caretaker's profile expanded

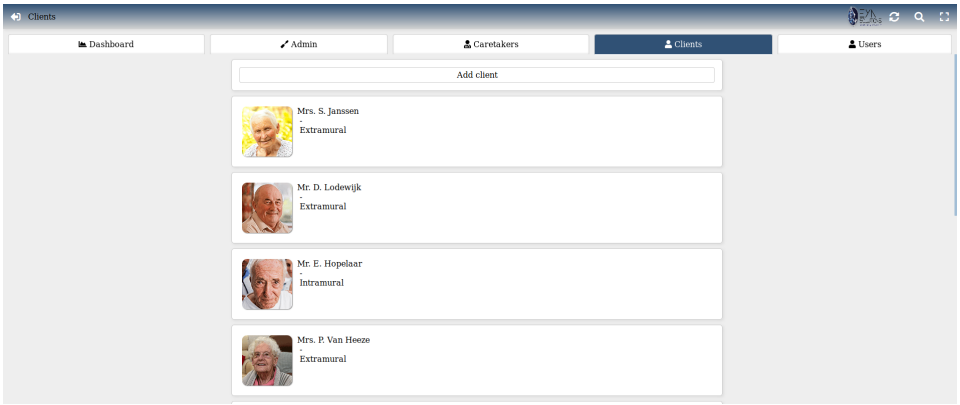


Figure G.4: List of clients page

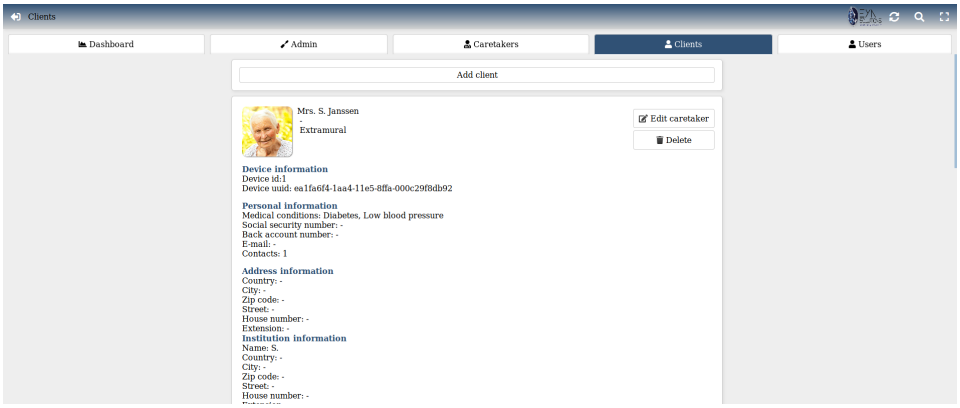


Figure G.5: Client's profile expanded

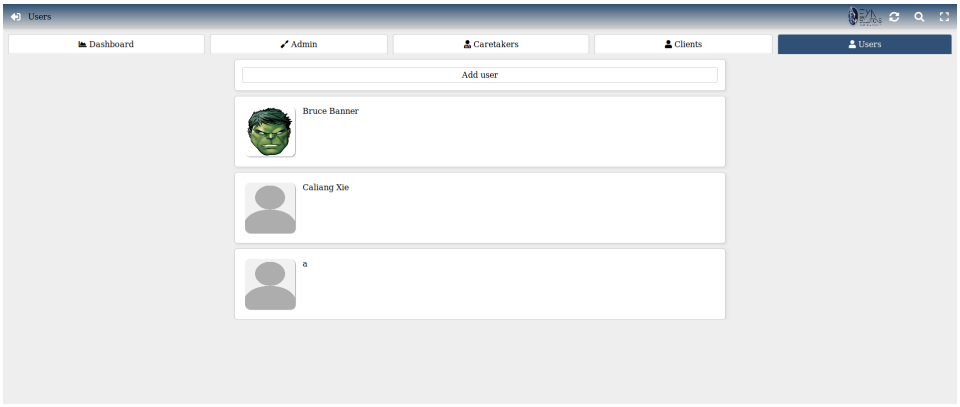


Figure G.6: List of users page

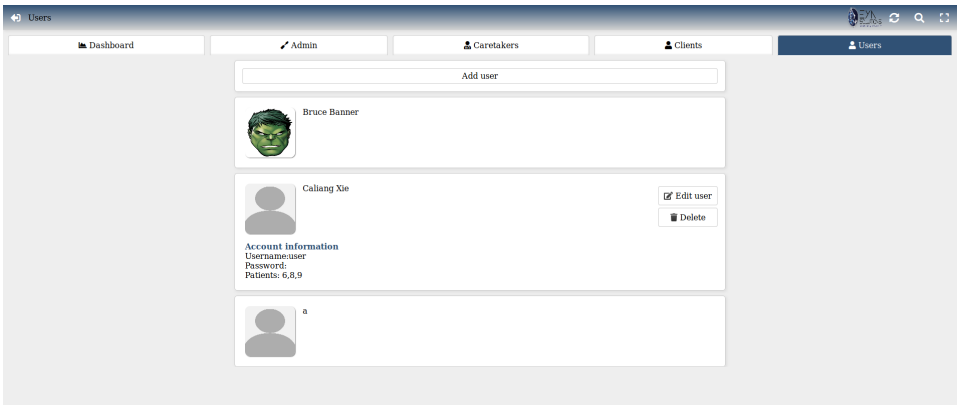


Figure G.7: User's profile expanded

Bibliography

[1] Influxdb glossary of terms.

[2] Node.js ws module. `https://www.npmjs.com/package/ws`.

[3] Unnati Dhanaliya and Anupam Devani. Implementation of e-health care system using web services and cloud computing. In *Communication and Signal Processing (ICCSP), 2016 International Conference on*, pages 1034–1036. IEEE, 2016.

[4] Anita George, Hindhuja Dhanasekaran, Jagrithi Palachanda Chittiappa, Lavanya Aishani Challagundla, Shreya Satish Nikkam, and Omar Abuzaghleh. Internet of things in health care using fog computing. In *Systems, Applications and Technology Conference (LISAT), 2018 IEEE Long Island*, pages 1–6. IEEE, 2018.

[5] Thomas Goldschmidt, Anton Jansen, Heiko Koziolek, Jens Doppelhamer, and Hongyu Pei Breivold. Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 602–609. IEEE, 2014.

[6] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges. In *2015 IEEE International Conference on Services Computing*, pages 285–292. IEEE, 2015.

[7] M Shamim Hossain and Ghulam Muhammad. Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring. *Computer Networks*, 101:192–202, 2016.

[8] Abbas Karimi, Faraneh Zarafshan, Adznan Jantan, Abdul Rahman Ramli, M Saripan, et al. A new fuzzy approach for dynamic load balancing algorithm. *arXiv preprint arXiv:0910.0317*, 2009.

[9] Sameena Naaz, Afshar Alam, and Ranjit Biswas. Effect of different defuzzification methods in a fuzzy based load balancing application. *International Journal of Computer Science Issues (IJCSI)*, 8(5):261, 2011.

[10] Md SQ Zulkar Nine, Md Abul Kalam Azad, Saad Abdullah, and Rashedur M Rahman. Fuzzy logic based dynamic load balancing in virtualized data centers. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*, pages 1–7. IEEE, 2013.

[11] Srinivas Sethi, Anupama Sahu, and Suvendu Kumar Jena. Efficient load balancing in cloud computing using fuzzy logic. *IOSR Journal of Engineering*, 2(7):65–71, 2012.

[12] Nadeem Shah and Mohammed Farik. Static load balancing algorithms in cloud computing: Challenges & solutions. *International Journal Of Scientific & Technology Research*, 4(10):365–367, 2015.