CE301 Final Report: Nurse Activities Score (NAS) Software
Workload statistics for intensive care units

A Dissertation Presented

by

DAN A. R. NORSTRÖM

Supervisor
DR. JON CHAMBERLAIN

Second Assessor
Dr. DELARAM JARCHI

Submitted to the Graduate School of the
University of Essex in partial fulfillment
of the requirements for the degree of

BACHELOR OF COMPUTER SCIENCE

04 2021

UNDERGRADUATE COURSE IN BSc COMPUTER SCIENCE

ACKNOWLEDGEMENTS

I would like to thank my supervisor Jon Chamberlain, other academic staff, and members of the technical and professional services team that have helped in the work on this project.

Particularly I would like to thank DR. Siv Stafseth for all the valuable insights into NAS as well as Anette Herstad at Kirkenes ICU department for the valuable feedback and user-testing during the extent of the project.

ABSTRACT

The purpose of this project is to create a software that can replace manual calculations at hospitals that lack infrastructure for intensive care unit workload evaluation. Effectively becoming a tool that can be utilized to improve hospital humanitarian standards and economy while ascertaining quality of care.

Nurse Activities Score (NAS) is a tool for quantifying intensive care unit workload at hospitals globally. In this project I am developing an automatized web application capable of reporting nurse workload, peer-viewing data, calculating and visually representing its related statistics.

Nurses can swiftly fill in their patients NAS based on their status and basic activities in a form while Matrons may peer-view the data sent in by the nurses. This allows them to perform quality control and pick up on any incorrect reports to remedy them and improve the correctness of all the related statistics.

Management may access the dashboard to view all related NAS statistics on either a per hospital basis or globally. The global dashboard also provides geographical data, showcasing trends over time across all regions in a country visually.

Frond-end is built using ReactJS, a JavaScript framework used to create responsive interfaces.

The backend is built using NodeJS with the ExpressJS framework. NodeJS is a server-side scripting environment used to build modern JavaScript web-applications.

The software is deployed using Docker and Docker Compose, containerizing the software and packaging all dependencies to create a lightweight and standalone executable.

MongoDB Atlas Cloud is a database that holds all the reports in JSON format, the foremost format for effective web and API communication.

Login and Authentication is secured using Auth0, a service platform that specialises in authorization.

Finally, the software is hosted on AWS using their Linux EC2 service, a VM capable of scaling resources after requirements efficiently.

Table of Contents

LIST OF SYMBOLS

AJAX: Asynchronous JavaScript And XML

Auth0: a login security provider

AWS: amazon web services

DB: database

EC2: Amazon Elastic Compute Cloud, VM hosted at AWS.

GUI: Graphical User Interface

ICU: intensive care units

MERN: shorthand for the technologies: MongoDB, Express.JS, React.JS & Node.JS

NAS: nurse activities score

VM: Virtual machine

WAMP: shorthand for the technologies: Apache web server, OpenSSL, MySQL, PHP

XML: Extensible Markup Language

LITERATURE REVIEW

The workload tool Nursing Activities Score was validated and published in 2003 [1]. The tool has a scale from 18.3-176.8% representing the workload and activities performed per shift or per day in intensive care units. A score of 100 (%) indicates the work of one nurse per shift around the clock i.e., a nurse-patient ratio of 1:1.

After implementation in several countries an up-dated version was launched in 2015 [2]. During the last decade three reviews [3, 4, 5] have confirmed the validity and reliability of the tool, allowing data and trends in National quality registers. Furthermore, workload and activity in critical care were described by Greaves et al. and found that “NAS is the most extensively examined workload tool, with generally reliable results” [6].

In recent years, the costs of nursing staff have been of interest for society and politicians. NAS could easily describe costs with exceptional accuracy [7]. Worldwide, a total of over 50 studies on ICU patients and nursing workload have been published with the NAS tool. Two of the newest studies are from the Netherlands and Belgium, the first NAS based on time measurements [8] and impact of COVID-19 on nursing time in the ICUs [9].

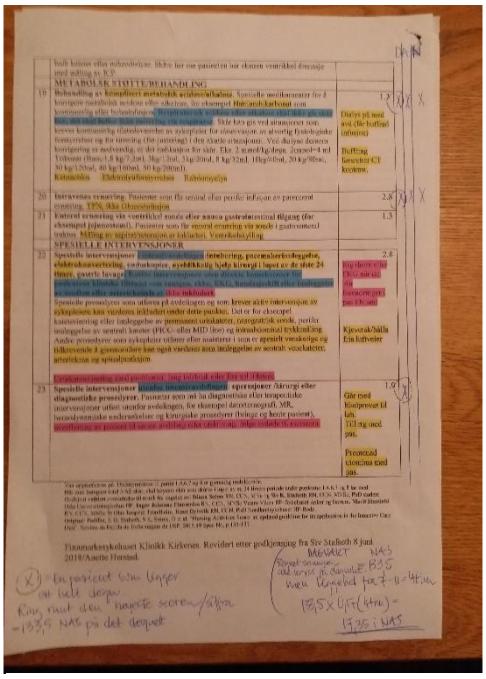
The software has been updated to follow the NAS descriptions publicised in 2020 [10], increasing the detail and scope of the categories originally from 2015 [2].

CONTEXT

In 2019 I had the valuable opportunity to assist an ICU at the Norwegian hospital in Kirkenes with their report to the Northern Norway Regional Health Authority (Helse Nord). This report consisted of the last years NAS (a type of workload statistics) calculations collected at the hospital's ICU, as well their summary and corresponding graphs.

The NAS numbers was saved in paper format, five papers per patient per day, placed in folders marked by date. All this data had to be processed by the nurses by hand, consuming a lot of time out of their busy schedule.

I was responsible for converting the numbers into data science, following the strict guidelines of the NAS-model to calculate statistics such as patient-workload by categories, weekend nurse workload, as well as a NAS-Index that describes how many days the ICU has been worked above its capacity. I produced the data required in the form of a excel workbook.



	[illegible] kateter eller mikrodialyse. Ikke her om pasienten har ekstern ventrikkel drenasje med måling av ICP		D A N
	METABOLSK STØTTE/BEHANDLING		
19	**Behandling av komplisert metabolsk acidose/alkalose.** Spesielle medikamenter for å korrigere metabolsk acidose eller alkalose, for eksempel Natriumbikarbonat som kontinuerlig eller bolusinfusjon. [illegible] Skal kun gis ved situasjoner som krever kontinuerlig tilstedeværelse av sykepleier for observasjon av alvorlig fysiologiske forstyrrelser og for titrering (fin-justering) i den akutte situasjonen. Ved dialyse dersom korrigering er nødvendig, er det indikasjon for skår. Eks. 2 mmol/kg/døgn. 2mmol=4 ml Tribonat (Baby: 1,8 kg/7,2ml), 3kg/12ml, 5kg/20ml, 8 kg/32ml, 10kg/40ml, 20 kg/80ml, 30 kg/120ml, 40 kg/160ml, 50 kg/200ml). Ketoacidose Elektrolyttforstyrrelser Rabdomyolyse	1.2 Dialys på med avd (får buffrad infusjon) Buffring fører/etter CT kontrast.	X X X
20	**Intravenøs ernæring.** Pasienter som får sentral eller perifer infusjon av parenteral ernæring. TPN, ikke Glukoseinfusjon	2.8	X X X
21	**Enteral ernæring via ventrikkel sonde eller annen gastrointestinal tilgang (for eksempel jejunostomi).** Pasienter som får enteral ernæring via sonde i gastrointestinal traktus. Måling av aspirat/retensjon er inkludert. Ventrikkeltømming	1.3	
	SPESIELLE INTERVENSJONER		
22	**Spesielle intervensjoner** [illegible] **intubering, pacemakerinnleggelse, elektrokonvertering, endoskopier, øyeblikkelig hjelp kirurgi i løpet av de siste 24 timer, gastric lavage.** [illegible] ikke inkludert. Spesielle prosedyrer som utføres på avdelingen og som krever aktiv intervensjon av sykepleiere kan vurderes inkludert under dette punktet. Det er for eksempel kateterisering eller innleggelse av permanent urinkateter, nasogastrisk sonde, perifer innleggelse av sentralt kateter (PICC- eller MID line) og intraabdominal trykkmåling. Andre prosedyrer som sykepleier utfører eller assisterer i som er spesielt vanskelige og tidkrevende å gjennomføre kan også vurderes som innleggelse av sentralt venekateter, arteriekran og spinalpunksjon. [illegible]	2.8 [illegible] Kjevetak/hålla frie luftveier	
23	**Spesielle intervensjoner utenfor intensivavdelingen: operasjoner /kirurgi eller diagnostiske prosedyrer.** Pasienter som må ha diagnostiske eller terapeutiske intervensjoner utført utenfor avdelingen, for eksempel datortomografi, MR, hemodynamiske undersøkelser og kirurgiske prosedyrer (bringe og hente pasient), [illegible]	1.9 Går med blodprøver til lab. Til rtg med pas. Prosessed [illegible] med pas.	X

[illegible]

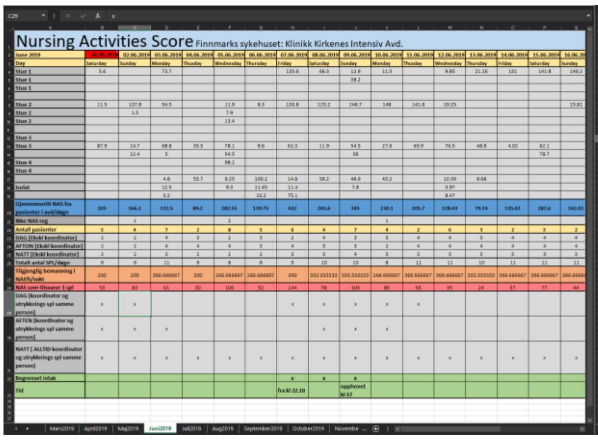
Finnmarkssykehuset Klinikk Kirkenes. Revidert etter godkjenning fra Siv Stafseth 8 juni 2018/Anette Herstad.

(X) = En pasient som ligger ut hele døgnet
Ring rundt den høyeste scoren/siffra
= 133,5 NAS på det døgnet

DAGVAKT NAS
Regnet sammen alt scores på dagvakt = 133,5
men Utskrevet fra 7-11 = 4t
13,5 x 0,17 (4t) =
17,35 i NAS

Figure 1: Manual NAS Form

Later the same year, I was tasked with automating their NAS procedures because their awaited NAS-system had yet to be implemented. I developed an excel codebook capable of summarising yearly data and produce a report with a dynamic timeframe. This allowed the nurses to skip every step of the calculations except for the base patient NAS-workload, which at the time was still done by paper. The amount of time the ICU nurses spent on registering their patients NAS numbers decreased because of my codebook, allowing them to focus their efforts on caretaking instead, effectively freeing up their working hours.



Nursing Activities Score Finnmarks sykehuset: Klinikk Kirkenes Intensiv Avd.

June 2019	01.06.2019	02.06.2019	03.06.2019	04.06.2019	05.06.2019	06.06.2019	07.06.2019	08.06.2019	09.06.2019	10.06.2019	11.06.2019	12.06.2019	13.06.2019	14.06.2019	15.06.2019	16.06.20
Day	Saturday	Sunday	Monday	Thusday	Wednesday	Thursday	Friday	Saturday	Sunday	Monday	Thusday	Wednesday	Thursday	Friday	Saturday	Sunday
Stue 1	5.6		73.7				135.6	66.3	12.9	11.3		9.85	21.26	131	141.8	146.2
Stue 1									38.2							
Stue 1																
Stue 2	11.5	137.8	54.5		11.9	8.5	133.8	125.2	146.7	148	141.8	19.25				15.81
Stue 2		1.3			7.9											
Stue 2					13.4											
Stue 3																
Stue 3	87.9	14.7	68.8	35.5	78.1	9.6	61.3	11.9	54.5	27.6	69.9	78.5	48.8	4.02	62.1	
		12.4	5		54.5				36						78.7	
Stue 4					58.2											
Stue 4																
			4.8	53.7	9.25	100.2	14.8	58.2	48.9	43.2		10.59	9.68			
Isolat			12.5		9.3	11.45	11.4		7.8			3.97				
			3.2			10.3	75.1					8.47				
Gjennomsnitt NAS fra pasienter i avd/døgn	105	166.2	222.5	89.2	282.55	139.75	432	261.6	345	230.1	205.7	128.43	79.74	135.02	282.6	162.01
Ikke NAS reg		1			2					1						
Antall pasienter	3	4	7	2	8	5	6	4	7	4	2	6	3	2	3	2
DAG [Ekskl koordinator]	2	2	4	3	2	3	2	4	3	3	4	4	3	4	4	4
AFTON [Ekskl koordinator]	2	2	4	4	4	3	4	3	3	2	4	4	4	4	4	4
NATT [Ekskl koordinator]	2	2	3	2	2	2	3	3	4	3	3	3	3	3	3	3
Totalt antal SPL/døgn	6	6	11	9	8	8	9	10	10	8	11	11	10	11	11	11
Tilgjengelig bemanning i NAS%/vakt	200	200	366.666667	300	266.666667	266.666667	300	333.333333	333.333333	266.666667	366.666667	366.666667	333.333333	366.666667	366.666667	366.6666
NAS som tilsvarer 1 spl	53	83	61	30	106	52	144	78	104	86	56	35	24	37	77	44
DAG [koordinator og utrykknings spl samme person]	x	x					x	x	x	x	x					
AFTEN [koordinator og utrykknings spl samme person]	x	x	x					x	x	x						
NATT [ALLTID koordinator og utrykknings spl samme person]	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Begrenset intak							x	x	x							
Tid							fra kl 22.10		opphevet kl 17							

Figure 2: Data Processing

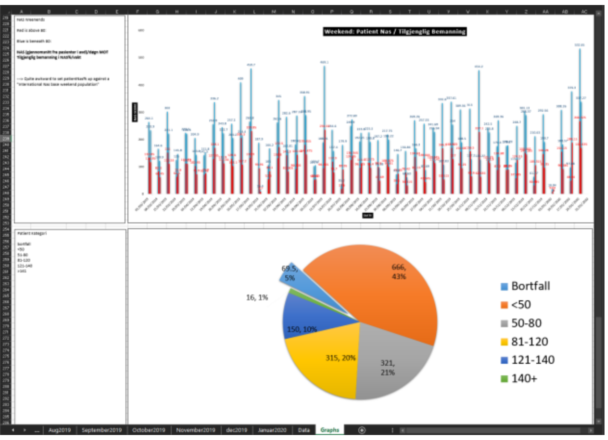


Figure 3: Part of the report

Seeing how much work went into these reports at an ICU that was already understaffed was surprising to me. The number of hours that went into bureaucracy instead of caretaking was enough to put staff under a lot of undesirable pressure. While developing the last codebook I racked my brain for ways to remove the biggest time-consuming step of the report, registering the patient's workload on paper. This became a big hurdle as the paper consists of three different shifts and a multitude of different fields that are filled out over the day by up to three different nurses. There was no clear way of alleviating this at the time, so the manual registration still had to be done.

Going forward from there I decided that it would be a good idea to solve this real-world problem by creating a NAS-software as my bachelor project. Iterating on top of what I already learned to create a software capable of increasing the humanitarian standards at ICUs by lowering their workload.

AIMS & OBJECTIVES

Intensive care units can find themselves overrun and overworked at times, creating situations where they cannot provide the required aid to each patient. As ICU’s usually handle more volatile patients compared to other departments, there is a higher chance of severe complication in correlation with workload.

If situations like these happen frequently, these departments usually employ a workload statistics tool, like NAS, to collect information that can act as proof of failure for quality of service. These tools can show management that they need to make adjustment to personnel or protocol to improve their service to follow the governmental guidelines.

Intensive care units do not always have access to workload measurement software to report NAS, therefore they must dedicate a sizeable sum of time to report their NAS-numbers manually. This requires excessive learning and protocol for intensive care nurses who already have a stressful work-environment.

In this project I will develop a user-friendly NAS software, where:

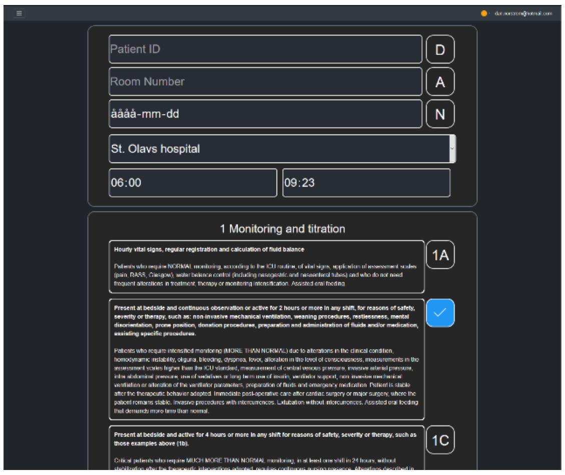
- Nurses can report their NAS swiftly
- Matron can report personnel numbers and inspect forms sent in by nurses
- Management can supervise statistics in a dashboard

It will follow the NAS-model to the point, keeping up with the required standards to be able to provide accurate and useful reports at hospitals. The main dashboard will include statistics on personnel count, patient count, patient workload categories, NAS resistance versus workload, NAS, and finally geodata representation of the average NAS across a country’s regions.

THE PRODUCT

❖ *This chapter explains the software's utilization*

Patient & Personnel Registration Form



St. Olavs hospital

06:00 09:23

1 Monitoring and titration

Hourly vital signs, regular registration and calculation of fluid balance

Patients who require NORMAL monitoring, according to the ICU routine, of vital signs, application of assessment scales (pain, RASS, Glasgow), water balance control (including nasogastric and nasoenteral tubes) and who do not need frequent alterations in treatment, therapy or monitoring intensification. Assisted oral feeding.

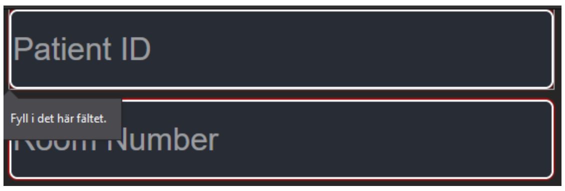
✓ **Present at bedside and continuous observation or active for 2 hours or more in any shift, for reasons of safety, severity or therapy, such as: non-invasive mechanical ventilation, weaning procedures, restlessness, mental disorientation, prone position, donation procedures, preparation and administration of fluids and/or medication, assisting specific procedures.**

Patients who require intensified monitoring (MORE THAN NORMAL) due to alterations in the clinical condition, hemodynamic instability, oliguria, bleeding, dyspnea, fever, alteration in the level of consciousness, measurements in the assessment scales higher than the ICU standard, measurement of central venous pressure, invasive arterial pressure, intra abdominal pressure, use of sedatives or long term use of insulin, ventilator support, non invasive mechanical ventilation or alteration of the ventilator parameters, preparation of fluids and emergency medication. Patient is stable after the therapeutic behavior adopted. Immediate post-operative care after cardiac surgery or major surgery, where the patient remains stable. Invasive procedures with intercurrences. Extubation without intercurrences. Assisted oral feeding that demands more time than normal.

Present at bedside and active for 4 hours or more in any shift for reasons of safety, severity or therapy, such as those examples above (1b).

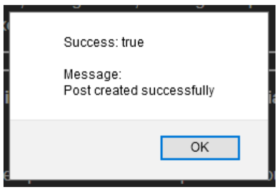
Critical patients who require MUCH MORE THAN NORMAL monitoring, in at least one shift in 24 hours, without

Figure 2: Patient Registration Form



Fyll i det här fältet.

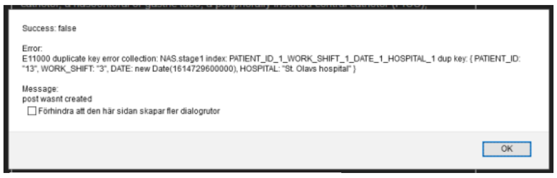
Figure 5: unchecked field



Success: true

Message:
Post created successfully

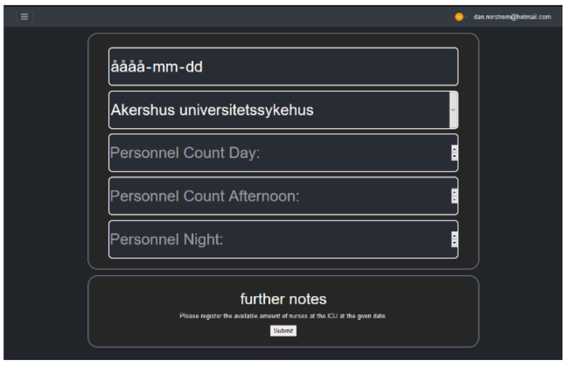
Figure 6: successful alert



```
Success: false

Error:
E11000 duplicate key error collection: NAS.stage1 index: PATIENT_ID_1_WORK_SHIFT_1_DATE_1_HOSPITAL_1 dup key: { PATIENT_ID:
"13", WORK_SHIFT: "3", DATE: new Date(1614729600000), HOSPITAL: "St. Olavs hospital" }

Message:
post wasnt created
```

☐ Förhindra att den här sidan skapar fler dialogrutor

Figure 7: unsuccessful alert



åååå-mm-dd

Akershus universitetssykehus

Personnel Count Day:

Personnel Count Afternoon:

Personnel Night:

Please register the available amount of nurses at the ICU at the given date.

Figure 8: Personnel registration form

Nurses can fill in their patient details at the end of each shift. The treatments listed in this form corresponds to about 80.8% [1] of an ICU nurse's total workload over the course of a shift.

The form is mobile friendly with big custom checkboxes as requested by the nurses.

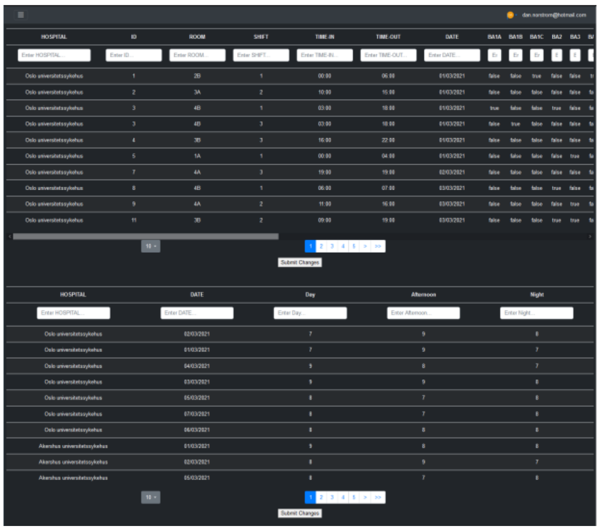
If a nurse attempts to submit the form without filling out all the required fields, the form automatically returns the user to said field, marks it red and prompts them to fill it out before continuing.

Figure 6 shows a successful prompt when the form has been sent

Figure 7 displays an unsuccessful prompt when the form has been sent. It includes an error message derived from MongoDB's Indexes as well as a text interpretation sent from the Servers API. In this example a duplicate post has been sent, which is against the index rules, hence an error is returned. For the nurse this means that someone has already reported this patient at this hospital during this shift.

The personnel registration form is used by Matrons to document the daily nurse attendance at an ICU. This form shares its traits with the patient registration form, they have the same capabilities to handle errors.

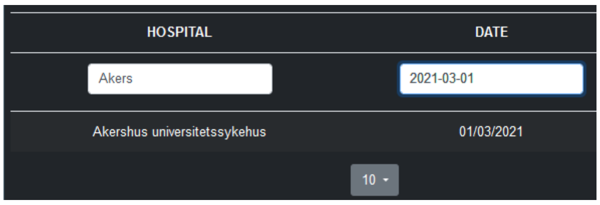
Peer view Controller



HOSPITAL	ID	ROOM	SHIFT	TIME-IN	TIME-OUT	DATE	BA1A	BA1B	BA1C	BA2	BA3
Enter HOSPITAL...	Enter ID...	Enter ROOM...	Enter SHIFT...	Enter TIME-IN...	Enter TIME-OUT...	Enter DATE...					
Oslo universitetssykehus	1	2B	1	00:00	06:00	01/03/2021	false	false	true	false	false
Oslo universitetssykehus	2	3A	2	10:00	15:00	01/03/2021	false	false	false	false	false
Oslo universitetssykehus	3	4B	1	03:00	18:00	01/03/2021	true	false	false	true	false
Oslo universitetssykehus	3	4B	3	03:00	18:00	01/03/2021	false	true	false	false	false
Oslo universitetssykehus	4	3B	3	16:00	22:00	01/03/2021	false	false	false	false	false
Oslo universitetssykehus	5	1A	1	00:00	04:00	01/03/2021	false	false	false	false	true
Oslo universitetssykehus	7	4A	3	19:00	19:00	02/03/2021	false	false	false	false	false
Oslo universitetssykehus	8	4B	1	06:00	07:00	03/03/2021	false	false	false	true	false
Oslo universitetssykehus	9	4A	2	11:00	16:00	03/03/2021	false	false	false	false	true
Oslo universitetssykehus	11	3B	2	09:00	19:00	03/03/2021	false	false	false	true	true

HOSPITAL	DATE	Day	Afternoon	Night
Enter HOSPITAL...	Enter DATE...	Enter Day...	Enter Afternoon...	Enter Night...
Oslo universitetssykehus	02/03/2021	7	9	8
Oslo universitetssykehus	01/03/2021	7	9	7
Oslo universitetssykehus	04/03/2021	9	8	7
Oslo universitetssykehus	03/03/2021	9	9	8
Oslo universitetssykehus	05/03/2021	8	7	8
Oslo universitetssykehus	07/03/2021	8	7	8
Oslo universitetssykehus	06/03/2021	8	8	8
Akershus universitetssykehus	01/03/2021	9	8	8
Akershus universitetssykehus	02/03/2021	8	9	7
Akershus universitetssykehus	05/03/2021	8	7	8

Figure 9: Peer view controller



HOSPITAL	DATE
Akers	2021-03-01
Akershus universitetssykehus	01/03/2021

Figure 10: Sorting mechanism

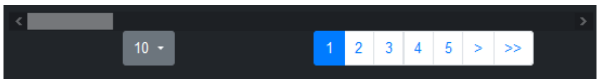


Figure 11: result and page sorting

Matrons can use the peer view controller to view all forms sent in by the nurses, correct mistakes in real-time and submit their changes.

This acts as a quality control where hospitals can allow Matrons to do routine checks of the forms by comparing it to the hospital journal systems, reducing the risk of incorrect reports and hence strengthening the correctness of the data in the dashboard.

The top controller handles patient data while the bottom controller handles personnel data

Both controllers have sorting elements, allowing matrons to quickly navigate and access data of interest. It also supports lexical sort by clicking the header of each field.

For ease of use a paginator has been implemented, it allows the matrons to select how many rows of data to show on each page as well as navigate said pages.

Report Dashboard

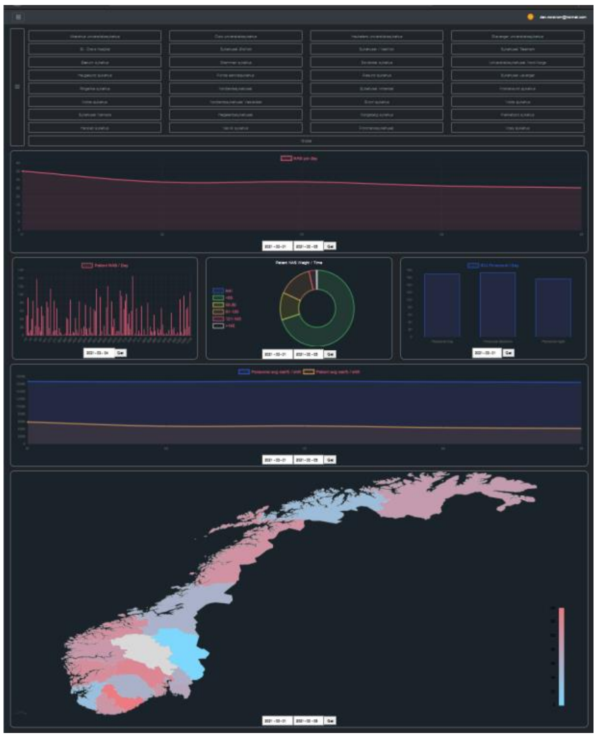


Figure 12: The Report Dashboard

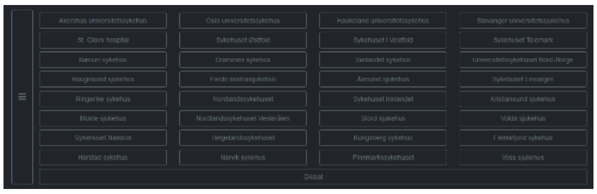


Figure 13: Hospital Menu 1

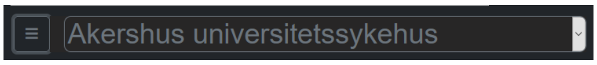


Figure 14: Hospital Menu 2

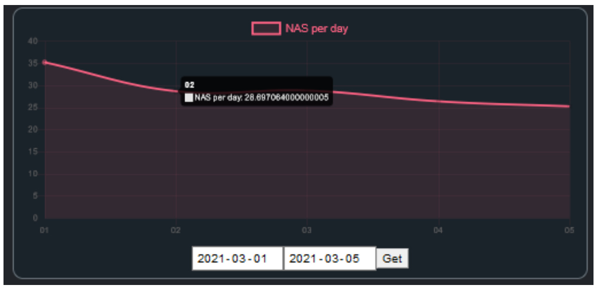


Figure 15: NAS per day

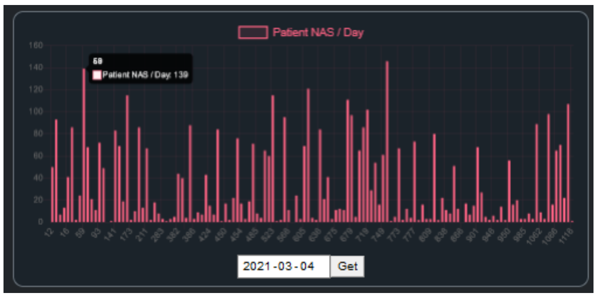


Figure 16: Patients NAS per day

The core of this software. The Dashboard presents important NAS statistics based on the data sent in by the nurses.

At the top of the page health management personnel can select which hospital they would like to research; upon selection the dashboard renders with animations to reflect that hospitals data.

The dates can be selected individually for the different graphs,

Hospitals may be selected either from a button-grid layout or from a scroll down list

The dashboard components have four different viewport breakpoints for how they render, hence they render well with small or big monitors.

There are multiple different graphs in the dashboard that assists management in taking actions to improve hospital standards.

NAS per day insinuates the total NAS score obtained at the end of each day, here management can observe trends in workload capacity across hospitals and take actions if a specific hospital performs above capacity for an extended period.

Patient NAS per day insinuates the NAS score of each individual patient per day. This gives management an overview of the count and workload of patients on a specific date. Here management can spot trends of high workload patients on specific dates, for example the day after Christmas might have a higher count of small workload patients while Octoberfest may have fewer patients but each with a higher workload.

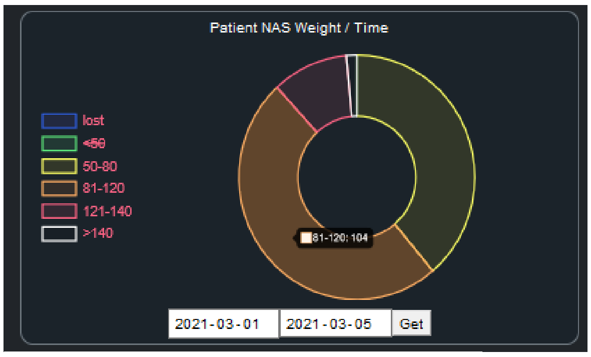


Figure 17: Patient NAS Categories

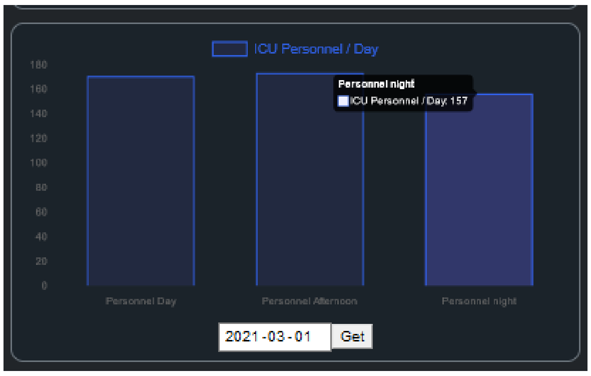


Figure 18: Personnel NAS per day per shift

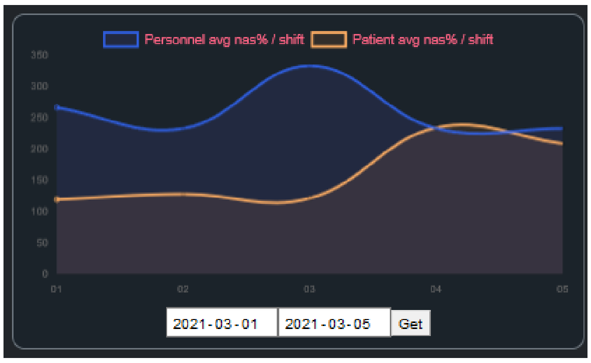


Figure 19: Workload resistance versus workload

Patient NAS weight / Time insinuates the workload sum of patients categorized into severity of treatment. A higher NAS per patient means that the patient required more attention from the nurses compared to one with a low NAS. Management can use these categories to spots trends regarding when and where patients requiring more attention appears. A hospital that on average has a higher risk of receiving heavy workload patients might have to change their protocol to match the demand in their region.

ICU Personnel reports the count of nurses in each shift at one date per hospital or region (in the figure the global region is reported). Management can use this to ascertain how many nurses they have on location at any given time.

One of the staples in NAS is to report the average NAS percentage of Personnel and patients against each other. This shows management the workload resistance (the nurses) allocated to their workload (the patients). This is the base data required to calculate the first graph we looked at; NAS/Day. If management notices a trend with workload higher then resistance, action must be taken, else quality of service or nurse health may falter.

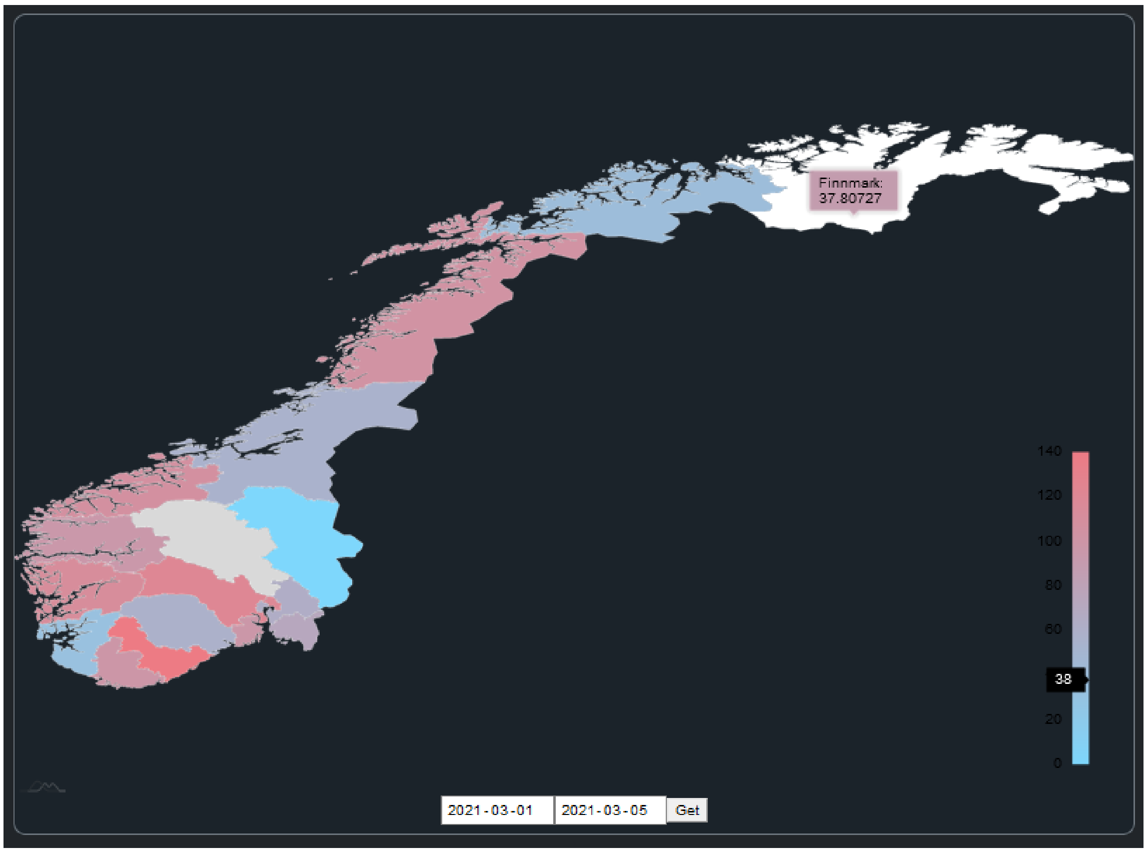


Figure 20: Countrywide Geographical Average NAS per Region

Geographical interpretation of each hospitals data divided upon their regions of availability is present on the global dashboard. Each region is colorised based on their average NAS per day index, hovering over a region allows management to inspect the name and values of different regions.

Management may use this geodata to notice trends bound by geographical locations. This allows management to react to changes afflicting certain regions of the country.

For example:

- ‘A pandemic starting from the Oslo area of Norway before spreading outwards.’
- ‘The northern Norway hospitals having a higher NAS because fewer nurses are willing to move to the area, resulting in a lack of nurses.’

By noticing these trends early, management may have more time to react to events, leading to:

- Fewer severely sick patients
- Limiting the spread of contagious diseases by contacting other government departments in time.
- Quicker response for workforce relocation patching to afflicted regions
- Better humanitarian standards for Nurses, reduced burnouts, and overtime.

Legal & Ethical

NAS is a globally accepted and free to use model for workload evaluation and is not limited by any intellectual properties.

If this software is deployed at a real hospital, patient ID must be removed, as in most countries storing data about the patient ID together with their afflictions outside of the government is illegal.

This software security is based on Auth0. If NAS data is labelled to be secured within the government or department where this software is deployed and Auth0 does not provide the correct ethical level of security, it will have to be replaced. This also goes for transferring data using MongoDB atlas as well as an EC2 instance from AWS.

TECHNOLOGIES

❖ *This chapter explains the technologies used to create the software*

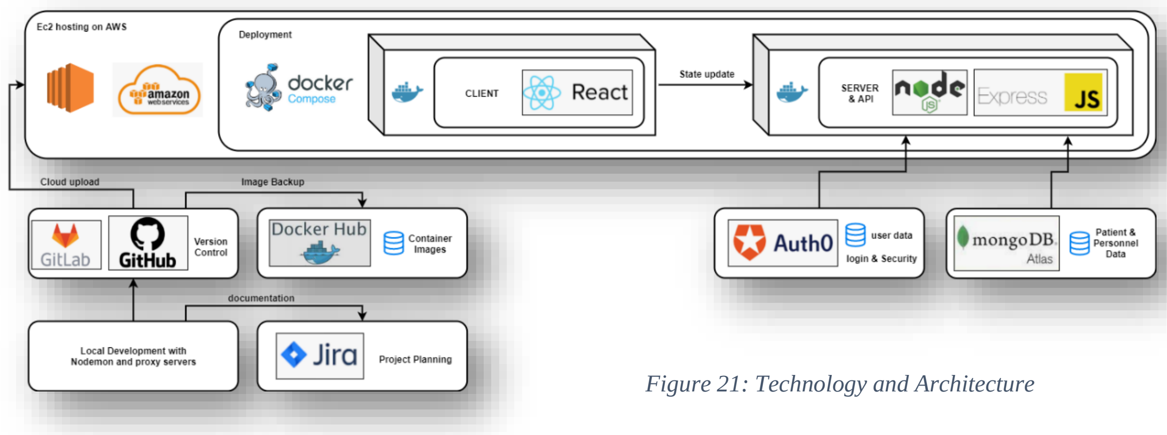


Figure 21: Technology and Architecture

The client is built using React.js, a frontend JavaScript library specialised in creating responsive one-page applications. The Server and its APIs are built using Node.js with the Express.js framework, the standard backend used together with react.js to create apps using full stack JavaScript.

Login and user information is secured using Auth0, a login service provider specialised in secure data transfer.

The database of choice is MongoDB Atlas, a NoSQL cloud cluster service with great functionality hosted on AWS.

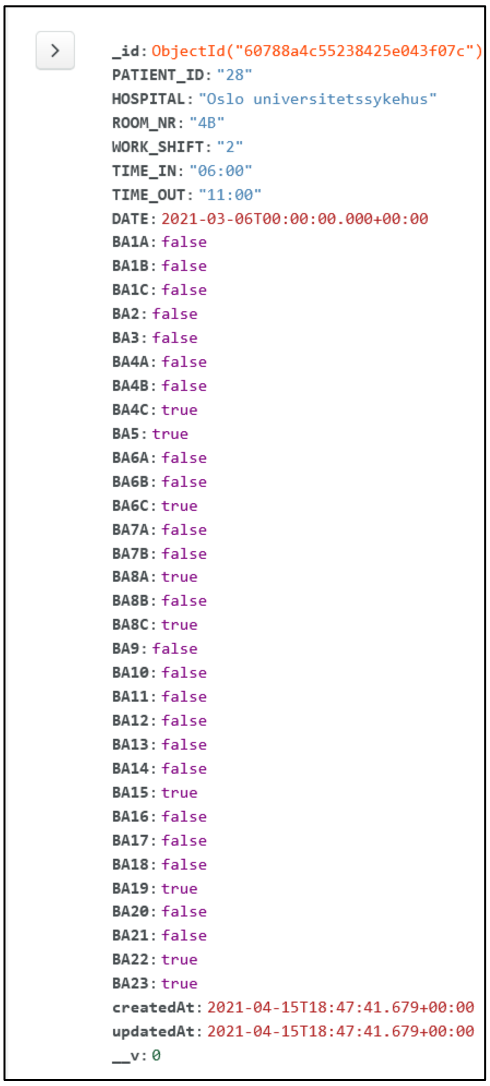
The client and server are running in docker containers, standalone executables capable of running in any environment. We alleviate the need of connections between the client and server using Docker-Compose, a tool to define, configure and deploy multi-container docker applications

At this stage, the application can be cloned from git to any device and deployed with a single command. Hence the application is hosted on Amazon Elastic Compute Cloud (EC2), a rentable virtual computer service capable of scaling its resources to my needs.

During development I commonly used Nodemon, Visual Studio Code, Postman, and the MongoDB Shell.

MongoDB

MongoDB is the document database deployed for this project. There are two different types of collections implemented, one for patient data and one for personnel data.



```
_id: ObjectId("60788a4c55238425e043f07c")
PATIENT_ID: "28"
HOSPITAL: "Oslo universitetssykehus"
ROOM_NR: "4B"
WORK_SHIFT: "2"
TIME_IN: "06:00"
TIME_OUT: "11:00"
DATE: 2021-03-06T00:00:00.000+00:00
BA1A: false
BA1B: false
BA1C: false
BA2: false
BA3: false
BA4A: false
BA4B: false
BA4C: true
BA5: true
BA6A: false
BA6B: false
BA6C: true
BA7A: false
BA7B: false
BA8A: true
BA8B: false
BA8C: true
BA9: false
BA10: false
BA11: false
BA12: false
BA13: false
BA14: false
BA15: true
BA16: false
BA17: false
BA18: false
BA19: true
BA20: false
BA21: false
BA22: true
BA23: true
createdAt: 2021-04-15T18:47:41.679+00:00
updatedAt: 2021-04-15T18:47:41.679+00:00
__v: 0
```

Figure 22: Patient JSON

The Patient document has fields related to identifying the patient such as patient id, room number, hospital, work shift, date as well as time-in and time-out.

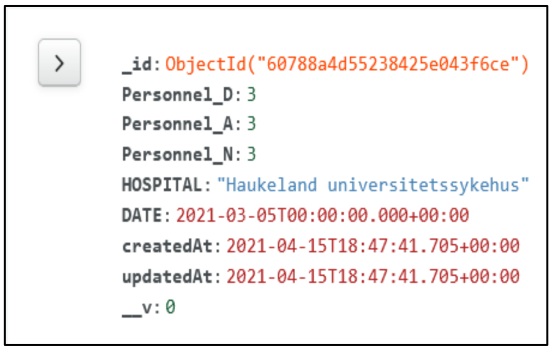
These fields are supported by a Database Index. Indexes state relationships between fields in documents, deciding which combinations are legal or illegal. The index in the patient collection (also called Stage1) is a Unique compound index. It states that there may only be one document with the same field combinations of PATIENT_ID, HOSPITAL, WORK_SHIFT and DATE.

This acts as a security layer, returning errors back to the server which passes it along to the client, reporting that the combination is invalid. Then matrons can peer-view the data for details if a mistake is suspected. Allowing the database to stay cleaner and less prune to incorrect data.

The fields labelled as "BA<xx>" are booleans stating if the selected NAS Basic Activity has been taken with the current patient or not.

The createdAt and updatedAt fields are added from the mongoose framework in the server's backend, allowing us to keep track of changes through its BSON format (Binary JSON).

Mongoose also adds a "__V" field called versionKey. It contains any internal revisions made to the documents.



```
_id: ObjectId("60788a4d55238425e043f6ce")
Personnel_D: 3
Personnel_A: 3
Personnel_N: 3
HOSPITAL: "Haukeland universitetssykehus"
DATE: 2021-03-05T00:00:00.000+00:00
createdAt: 2021-04-15T18:47:41.705+00:00
updatedAt: 2021-04-15T18:47:41.705+00:00
__v: 0
```

Figure 23: Personnel JSON

The Personnel document holds information about the daily count of nurses at the ICU spread between the three different shifts. There are also enough fields to correctly identify the unit using hospital name and date.

There is a Unique compound index stating that there may only be one unique combination of HOSPITAL and DATE present in the collection. This ascertains that the daily numbers are not reported more than once per hospital.

Just like the patient documents, the createdAt, updatedAt and versionKey are present.

There are limitation using a cloud database such as MongoDB Atlas. Because the DB is off-location there are fees regarding usage. The free tier that is currently active could run into limitations when being implemented for a bigger corporation, even more so as this software was built to connect all hospitals in a country together.

While the free tier might be resilient in the beginning, if a corporate would implement this software I would recommend using one of MongoDB's monthly subscriptions to alleviate any data, RAM, or storage limitation. One such service is Multi-Region Clusters, allowing low-latency access and better connectivity over multiple regions.

The current data in the DB has been simulated using the ranges of possible outcomes in each form. In this projects GitLab, two DB population scripts are available in the "nas_data_population" folder (Link).

Datascript.js builds patient and personnel documents based on random variables to create diverse and real looking data. I am basing these variables upon the one-year NAS data I collected from Kirkenes Hospital in Norway during my Freelance work. The simulated data acts on:

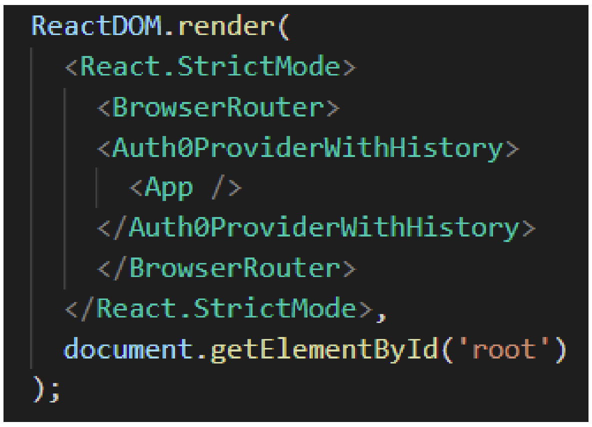
1. variable number of patients per day (simulated workload)
2. variable number of nurses per day (simulated workload resistance)
3. variable chances of certain NAS basic activities to be taken per hospital. (simulates workload based on regions)

Datascript.js then sends the created documents to their collections and reports any issues or errors returned from MongoDB.

DeleteScript.js cleanses the database of all data, this is an important script as MongoDB has no service to clear databases except by dropping them. Dropping a DB removes any indexes present which means that the index would have to be remade every time. There is also the factor of creating new indexes during development to consider, as indexes can only be created when their rules can be applied to all documents currently in the collection. If there is a single document that does not adhere to the index, the index fails to bind to the collection. Hence it is great to be able to empty the DB during development to ascertain that indexes are bound correctly.

React

React is our frontend framework, it decides when and what to render as well as what it looks like. React specializes in creating Components, they are code pieces that can be compared to a HTML version of a class or method but controlled with JavaScript code. Components are either functional or class based, while their structure can vary depending on their usage, a common structure includes

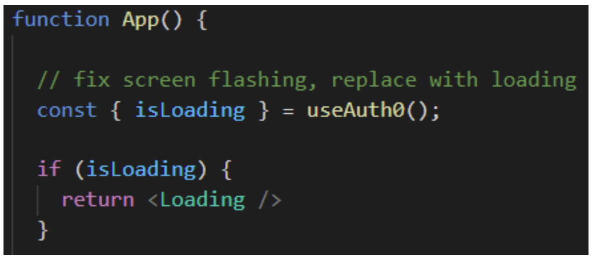


```
ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
    <Auth0ProviderWithHistory>
      <App />
    </Auth0ProviderWithHistory>
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

Figure 24: ReactDOM

React renders the application from the React DOM found in index.js. This is a one-page application, which means that index.js is the parent of all our components.

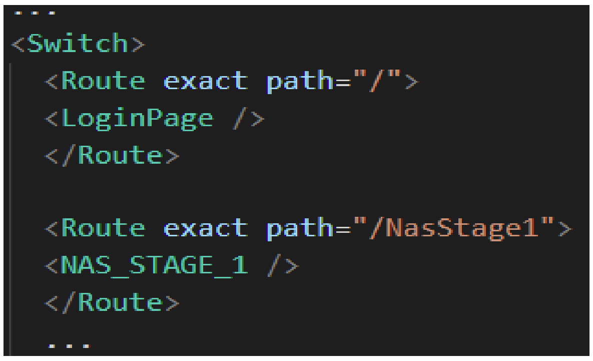
BrowserRouter and Auth0ProviderWithHistory are wrappers to the App component, they have been built to add a security layer with access protection as well as application routing. (Details in Components chapter)



```
function App() {

  // fix screen flashing, replace with loading
  const { isLoading } = useAuth0();

  if (isLoading) {
    return <Loading />
  }
```

Figure 25: Loading Implementation

in App.js the loading screen is called while authenticating user’s login. This removes the white flashes from the software as it tries to change domain.



```
...
<Switch>
  <Route exact path="/">
  <LoginPage />
  </Route>

  <Route exact path="/NasStage1">
  <NAS_STAGE_1 />
  </Route>
  ...
```

Figure 26: Routing

There is also a switch that decides which main components should be rendered when certain paths are used in the app. This allows React to simulate and look like a multi-page application while keeping the velocity of a single-page application.

There are many ways of styling components in React, component based .css files, global .css files, React styles and any other styling available to regular HTML static webpages. In this project I am using a Global .css file, React styling and bootstrap. One thing to consider while using bootstrap together with React is that they are both using JavaScript for certain operations. This can create issues with React DOM while using bootstraps JavaScript methods (such as dropdown menus or the like). There is the possibility of using a React library called React-Bootstrap to solve these issues, but I decided not to use any of the bootstrap JavaScript methods instead.

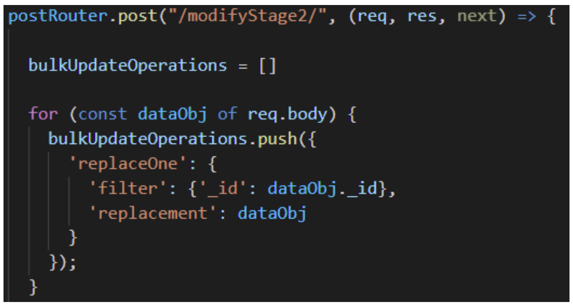
There are multiple different frameworks and packages to use and import for react. The ones used in this project are collected in a file called “package.json” (Link)

Node & Express

Node.js is the backbone of this application. It is an open-source JavaScript runtime Environment that allows execution of JavaScript code. With Node.js we may execute our client.js and server.js files using the command:

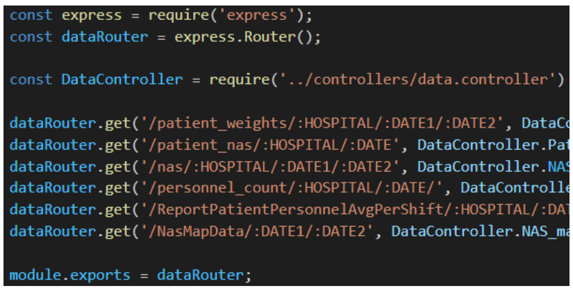
```
Node server
```

This creates an instance of the server; it has two express routers mounted on top of it to listen to client requests made to the server's API endpoint.



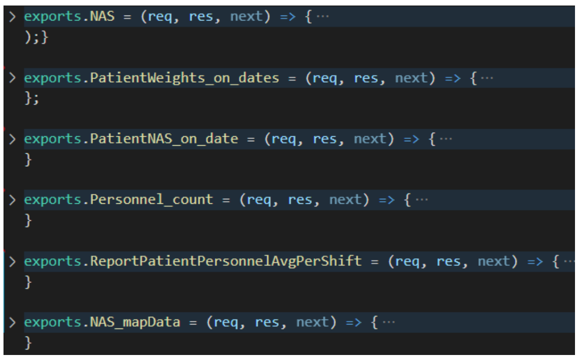
```
postRouter.post("/modifyStage2/", (req, res, next) => {

  bulkUpdateOperations = []

  for (const dataObj of req.body) {
    bulkUpdateOperations.push({
      'replaceOne': {
        'filter': {'_id': dataObj._id},
        'replacement': dataObj
      }
    });
  }
```

Figure 27: Bulk writer



```
const express = require('express');
const dataRouter = express.Router();

const DataController = require('../controllers/data.controller')

dataRouter.get('/patient_weights/:HOSPITAL/:DATE1/:DATE2', DataCo
dataRouter.get('/patient_nas/:HOSPITAL/:DATE', DataController.Pat
dataRouter.get('/nas/:HOSPITAL/:DATE1/:DATE2', DataController.NAS
dataRouter.get('/personnel_count/:HOSPITAL/:DATE/', DataControlle
dataRouter.get('/ReportPatientPersonnelAvgPerShift/:HOSPITAL/:DAT
dataRouter.get('/NasMapData/:DATE1/:DATE2', DataController.NAS_ma

module.exports = dataRouter;
```

Figure 28: Express Router



```
> exports.NAS = (req, res, next) => {…
  );}

> exports.PatientWeights_on_dates = (req, res, next) => {…
  };

> exports.PatientNAS_on_date = (req, res, next) => {…
  }

> exports.Personnel_count = (req, res, next) => {…
  }

> exports.ReportPatientPersonnelAvgPerShift = (req, res, next) => {…
  }

> exports.NAS_mapData = (req, res, next) => {…
  }
```

Figure 29: Data Controller

The postRouter has multiple endpoints for simple data addition or modifications. Allowing the client to call upon them to receive or send data to our server that passes it along to the DB using mongoose schemas.

The dataRouter is far more complex and carries out calculations on behalf of the requests. Once it receives a request it calls a method inside the data.controller.js that gets the required data from the DB and computes it to return it in the format required in the client

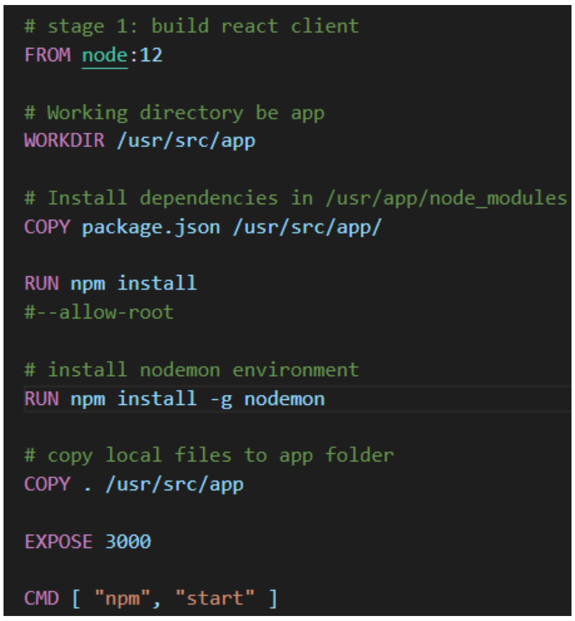
As mentioned, the data.controller holds multiple methods of computation required for the different NAS graphs. The biggest by far is the result_date_to_daily_patient_nas_collection method that calculated the sum of NAS from n amounts of patients in a timeframe, its used in a versatile way across most of the API endpoints to transpose the basic activities into NAS. (more about this in the API section)

All the communication with the MongoDB is carried out using the schemas created by the mongoose framework. Mongoose schemas is a way of modelling JSON (BSON) document structure to adhere to certain rules before being posted to a DB. By using these exported schemas bound to a certain collection we can ascertain that we have the correct format when we are querying the DB.

Docker

Docker role is to ease the communication and deployment of the project. By using a containerized client and server we can deploy them together using docker-compose. The main perks are:

- swift deployment
- robust environment, the containers can be deployed on multiple devices and be guaranteed to run the same way.
- The ability so use docker desktop to supervise each of the images consoles. Giving the developer one console that reports statuses, errors and logs for both the client and server.



```
# stage 1: build react client
FROM node:12

# Working directory be app
WORKDIR /usr/src/app

# Install dependencies in /usr/app/node_modules
COPY package.json /usr/src/app/

RUN npm install
#--allow-root

# install nodemon environment
RUN npm install -g nodemon

# copy local files to app folder
COPY . /usr/src/app

EXPOSE 3000

CMD [ "npm", "start" ]
```

Figure 30: Client Dockerfile

Dockerfile's are the basic building blocks that holds the instructions of how to build certain images. They decide which files to include, folder structure, network connections, ports exposure and much more. Here is an example of the client running in the development environment.

Once this image is initialized it attempts to build a container in the following way:

It gets Node version 12 as the main driver for the container, then it creates a local work directory and copies the react frameworks to it. Then it runs the install command for all the packages included, installing them on the container. Finally, it copies the application files of the client, exposes port 3000 on the container and runs the start command. It is worth noting that exposing a port on a container is different from exposing a port on a device. These inner ports act like network bridges between the containers, allowing them to communicate. In this case these ports are bound to the real device's ports, which essentially makes them the same as regular ports.

In the development environment Nodemon has been installed, it allows the container to update its files as code is being written. In this way a developer does not have to re-build the image and containers every time the code changes, creating a smooth development environment.

The servers dockerfile works much in the same way as the client, except that it exposes another port that will act as our API instead.

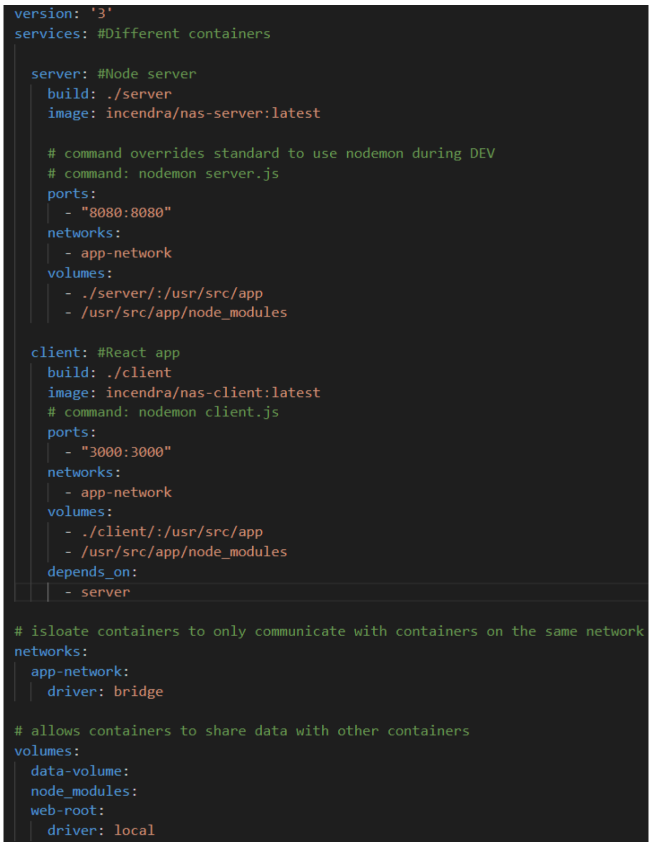

```
version: '3'
services: #Different containers

  server: #Node server
    build: ./server
    image: incendra/nas-server:latest

    # command overrides standard to use nodemon during DEV
    # command: nodemon server.js
    ports:
      - "8080:8080"
    networks:
      - app-network
    volumes:
      - ./server/:/usr/src/app
      - /usr/src/app/node_modules

  client: #React app
    build: ./client
    image: incendra/nas-client:latest
    # command: nodemon client.js
    ports:
      - "3000:3000"
    networks:
      - app-network
    volumes:
      - ./client/:/usr/src/app
      - /usr/src/app/node_modules
    depends_on:
      - server

# isloate containers to only communicate with containers on the same network
networks:
  app-network:
    driver: bridge

# allows containers to share data with other containers
volumes:
  data-volume:
  node_modules:
  web-root:
    driver: local
```

Figure 31: Server Dockerfile

These images can launch containers on their own, but to make them communicate with each other and interact with the devices ports we bind them together using docker-compose.

This docker compose file states that a server and a client should be built.

The image name is bound to my personal GitLab account. This exposes the files to my dockerhub account, allowing it to copy my images and keep them safe on its cloud image repository. Dockerhub was set up to ease deployment of the app into cloud (more on this in the EC2 section).

Ports are pairing the inner ports in the container to an outer port on the device or router. Allowing the containers to interact with the real world.

Both containers should be limited to only contact each other using the allocated network. This makes sure that no other container running on the device may interfere with these containers.

Volumes are used to share or persist data between containers. This allows the developer to extract and inspect data in individual containers. In this project this is only used to grant Nodemon access to the files so that it can signal changes and update its content accordingly.

Auth0

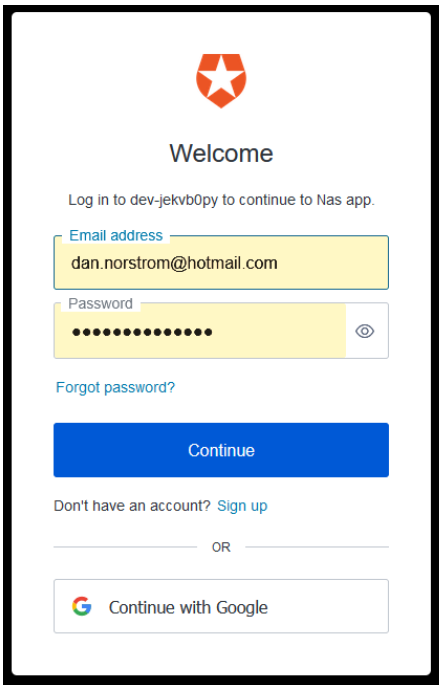
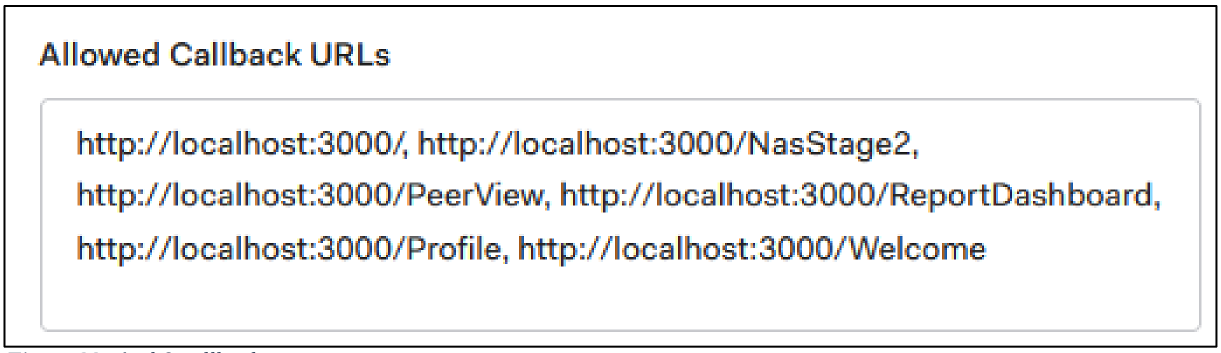


Figure 32: Auth0 Universal Login

Auth0 is a secure login authentication service that handles user data on behalf of applications. This service acts as a showcase for how access to this application can be secured using modern technology.

The purpose is to securely transfer, store and extract user information using a token system. The tokens are JSON web tokens (JWTS) which are meant for application usage. They allow this application to securely access user metadata stored on Auth0's servers.

Login services for this application will differ depending on which hospital and which country are using it. Governments in different countries use different security measures, hence it is difficult to create a login system without tailoring it to the region. It is worth noting that the signing up options is only enabled as a showcase, at a hospital, accounts will be made by admins.

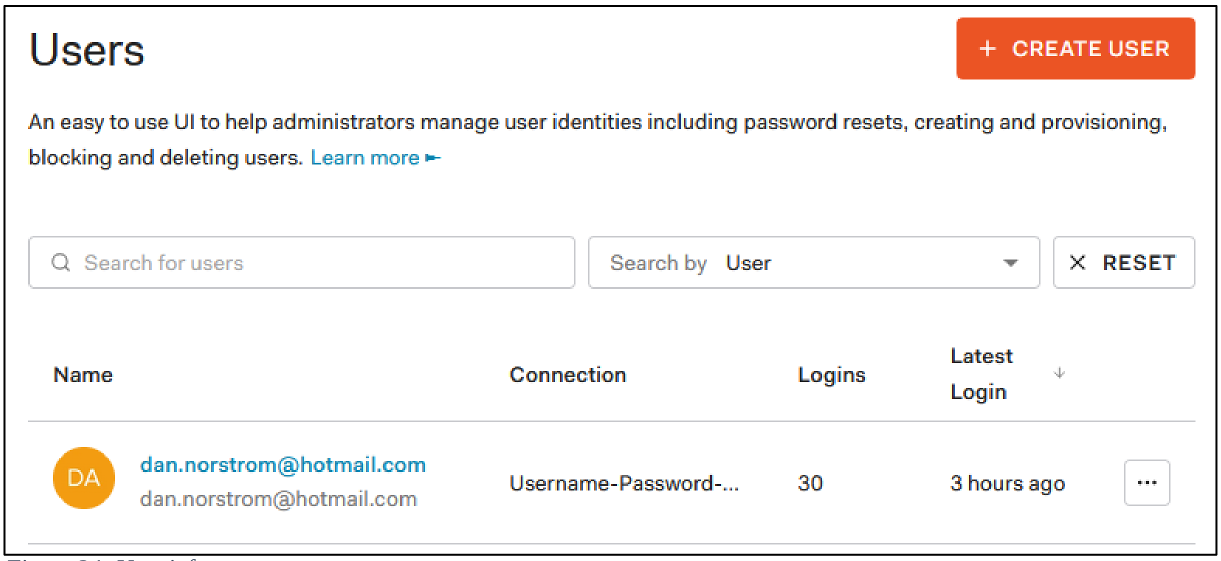


Allowed Callback URLs

http://localhost:3000/, http://localhost:3000/NasStage2, http://localhost:3000/PeerView, http://localhost:3000/ReportDashboard, http://localhost:3000/Profile, http://localhost:3000/Welcome

Figure 33: Auth0 callback

The purpose of Auth0 was to limit user access to certain components in the application. As of now only an authenticated user can view data related components. Auth0 can also control from which addresses it may be queried from. This adds another layer of security where the application cannot be accessed off-premises. Users that attempt to connect to the application with valid details but from an invalid IP will be rejected.



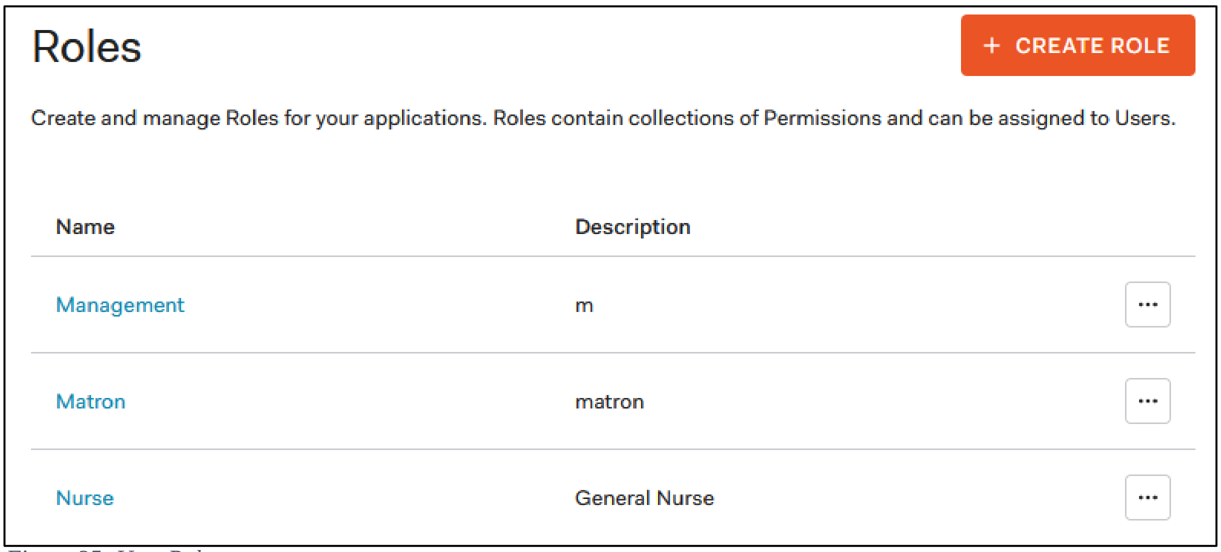
Users

An easy to use UI to help administrators manage user identities including password resets, creating and provisioning, blocking and deleting users. Learn more

Name	Connection	Logins	Latest Login
DA dan.norstrom@hotmail.com dan.norstrom@hotmail.com	Username-Password-...	30	3 hours ago

Figure 34: User info

Auth0' dashboard allows application administrators to create, block and remove users of the platform. This could be used to add logins suitable to each hospital's standard. They may also send verification emails, change passwords, and add roles.



Roles

Create and manage Roles for your applications. Roles contain collections of Permissions and can be assigned to Users.

Name	Description
Management	m
Matron	matron
Nurse	General Nurse

Figure 35: User Roles

Roles can be created and added to each user depending on their access rights. Each role can assign permissions to use certain resources, such as API endpoints.

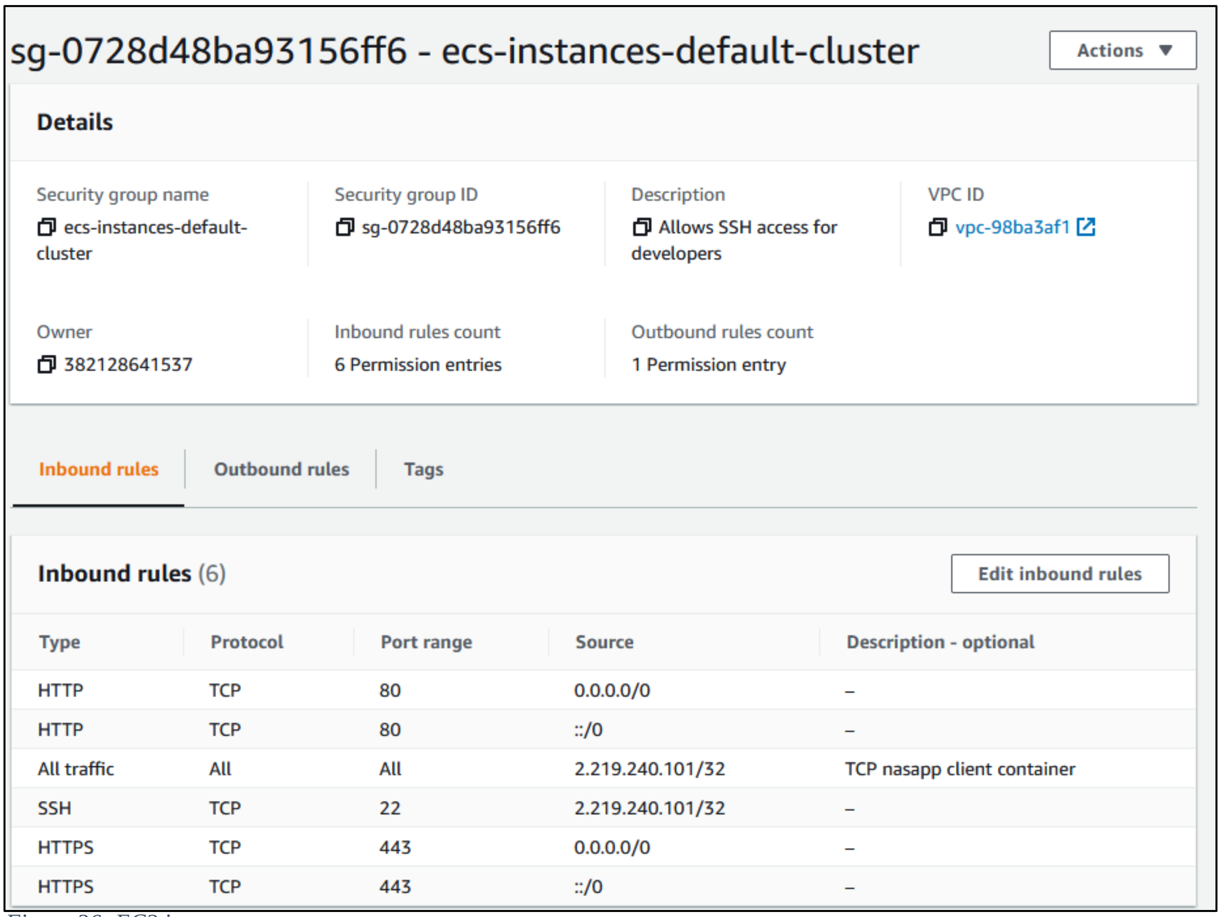
There are limitations to the number of users and number of queries that can be made using the free-tier of this service. 1000 queries / connections are plenty for showcase, but the tier would have to be improved to use the current login forms at an organisation, that would come at a diverse monthly fee, but as mentioned earlier, in most cases any hospital interested in this software would have their own security and authentication.

Amazon Elastic Compute Cloud (EC2)

Amazon offers a myriad of different services to host web applications and services. In this project I chose to use their EC2 service. Ec2 is essentially a VM running on the cloud with dynamic resource allocation. It allows me to only pay for the resources required to run the software when its online.

To begin with I created an account and a root user.

Then a security group must be made and managed to grant the correct permissions to modify any applications.



sg-0728d48ba93156ff6 - ecs-instances-default-cluster

Details

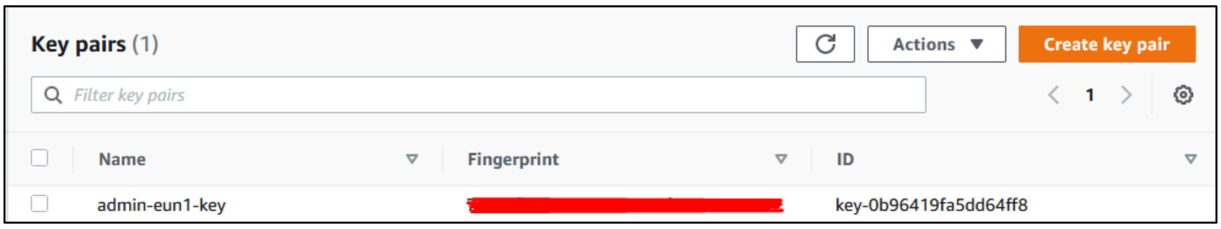
Security group name	Security group ID	Description	VPC ID
ecs-instances-default-cluster	sg-0728d48ba93156ff6	Allows SSH access for developers	vpc-98ba3af1
Owner	Inbound rules count	Outbound rules count	
382128641537	6 Permission entries	1 Permission entry	

Inbound rules (6)

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	–
HTTP	TCP	80	::/0	–
All traffic	All	All	2.219.240.101/32	TCP nasapp client container
SSH	TCP	22	2.219.240.101/32	–
HTTPS	TCP	443	0.0.0.0/0	–
HTTPS	TCP	443	::/0	–

Figure 36: EC2 instance

This security group allows the new user to SSH connect to the EC2 instances using key pairs generated by amazon.



Key pairs (1)

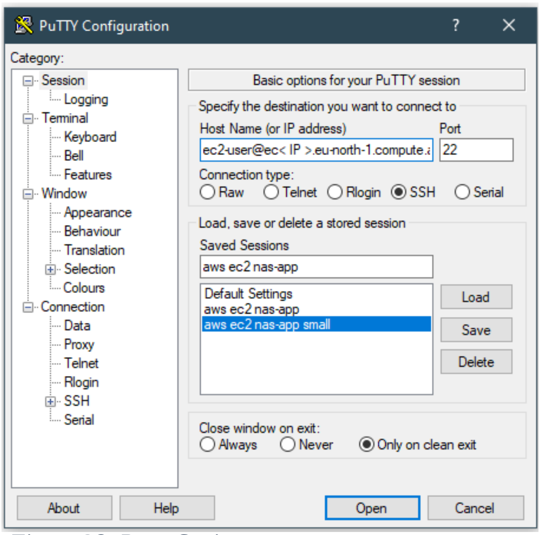
Name	Fingerprint	ID
admin-eun1-key	[redacted]	key-0b96419fa5dd64ff8

Figure 37: AWS Key Pairs

Now I needed to create an EC2 instance to host my application. There are a myriad of different operative systems, software, and hardware to choose from. I selected the Amazon Linux/Unix operative system using the t3.small service.

Linux is about two times cheaper to run and host compared to windows servers, allowing a healthier milage while hosting this application.

T3 is one of the many computer resource models available at AWS, it is a blueprint for how much resources should be allocated to the cloud instance. T3.micro is cheaper and allows some free-tier operations compared to T3.small, but it cannot be modified for extra disk space. The current software needs an instance with about 10-12gb of storage capabilities to run and build its docker components successfully. Hence, I am using a modified version of the t3.small service at a cost of about five pence an hour during no-stress.



Specify the destination you want to connect to

Host Name (or IP address): ec2-user@ec< IP >.eu-north-1.compute.a

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

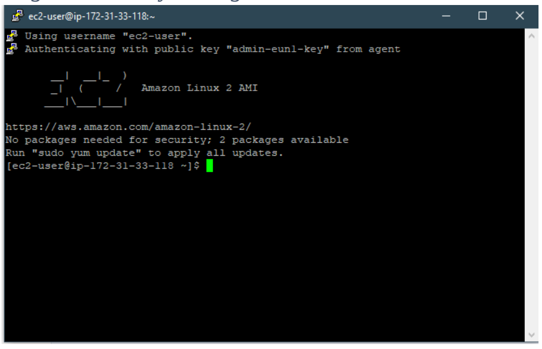
Load, save or delete a stored session

Saved Sessions: aws ec2 nas-app

Default Settings
aws ec2 nas-app
aws ec2 nas-app small

Close window on exit: Always Never [✓ Only on clean exit]

Figure 38: Putty Settings



```
ec2-user@ip-172-31-33-118:~
Using username "ec2-user".
Authenticating with public key "admin-eunl-key" from agent

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
No packages needed for security; 2 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-33-118 ~]$
```

Figure 39: Putty Terminal

Once the instance has been launched I can SSH into it using Putty with the details from the security user and the admin key I built earlier.

Inside the Ec2 Instance I have installed Docker and set it to start automatically with the instance. I have also limited the amount of RAM it may claim from the instance as Docker happily occupies whatever it can find with standard settings.

To start the application I clone it from git using:

git clone https://github.com/DanNorstrom/nas_software.git

--config core.autocrlf=input

The configuration part changes any whitespaces from windows to Linux, allowing me to develop on a windows device and push it to a Linux device unhindered.

Now when all the files are cloned, the containers can be built and deployed using the command:

```
docker-compose up
```

Before Auth0 was implemented this was live and could be accessed from the accepted domains in the security group. During development these are set to my personal pc's public ipv4, but during testing its set to either public or the testers ipv4, allowing users to test the software from anywhere. After the security update this instance ran into some issues that are described in the future works section.

COMPONENTS

❖ *This chapter explains the intricacies about each JavaScript React component used in frontend*

Name: NAS_STAGE_1.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 28, 30, 40, 42, 44, 45 ,48

User-Persona: " As a Nurse, I would like to swiftly report my patients NAS to spend more time caring for patients".

Description: This component renders the NAS form which is for reporting each basic activity taken by a nurse during their shift for each patient. The form is implemented using React style, which means that I am not using the innate form submitting event used in HTML. Instead, there is an onChange event on each part of the form inputs:

```
<input
    type="text"
    name="PATIENT_ID"
    value={state.PATIENT_ID}
    onChange={handleChange}
    placeholder="Patient ID"
    required
/>
```

This calls the handleChange method that changes the components state based on the input.

For the forms checkboxes it extracts the inputs checked status from evt.target.checked and assigns it to the state with the variable name equal to evt.target.name in the components state. This allows us to update the states for all boolean fields correctly.

For the forms text, date, and time fields it extracts the inputs evt.target.value and assigns it to the state with the variable name equal to evt.target.name in the components state.

For the few radio inputs in the form, the basic activities with letters such as A,B and C, more work is required. Because the standard HTML radio inputs modify a singular field, and our API requires all JSON fields to exist to process the data properly, I needed to either implement a new version of radio buttons or write individual code to handle each radio button.

```
function handleChange(evt) {

    const letterList = ['A','B','C']
    var resList = {}

    if (letterList.includes((evt.target.name).slice(-1))) {
      for (var s in letterList){
        var target_var = (evt.target.name).substring(0, (evt.target.name).length-1)
        target_var  += letterList[s]

        if (target_var != [evt.target.name]){
          try{
            document.querySelectorAll("input[name="+target_var+"]")[0].checked = false;
            resList[target_var] = false
          }
          catch(err) { }
        }
        else{
          console.log('yes')
          const value = evt.target.type === "checkbox" ? evt.target.checked : evt.target.value;
          resList[evt.target.name] = value
        }
      }
      setState({
            ...state,...resList
          });
    }
```

I added a code segment to the handleChange method that detects checkboxes ending with A, B or C. Once detected it uses a for loop to check for that inputs alternatives, the other letter endings with the same numbering. It then uses a document.querySelectorAll to find each related checkbox and unchecks it so that only one of the custom radio inputs are checked at one time and updates the components state accordingly. This allows the application to use these custom radio inputs to control multiple fields instead of singular fields, as required in our JSON specifications.

This method only takes this form into account, if expansion is required, care needs to be taken to avoid extra fields getting picked up in the custom radio input detection. Currently anything ending with A,B, or C will be detected.

One the form has been filled out, its data is present in the state of the component and can be sent to the server using a call to the formSubmit() method from the submit button in the form.

```
<form onSubmit={formSubmit}>
…
<button type="submit" >Submit</button>
```

formSubmit() prevents the default actions of HTML forms and instead creates a fetch request using standard request options for JSON while adding our data in JSON format to its body:

```
const requestOptions = {
      method: 'POST',
      headers: { 'Content-Type': 'application/json','Access-Control-Allow-Origin' : '*' },
      body: JSON.stringify(state)
    };
```

Then a global variable located in globals.js dictates if this application is running on the development or live platform and provides the correct IP address to query our API.

The fetch is sent to the API endpoint that populated the stage 1 collection in the database, A response is returned to the API that forwards it to the client and displays an alert with dynamic messages from both the API and the DB.

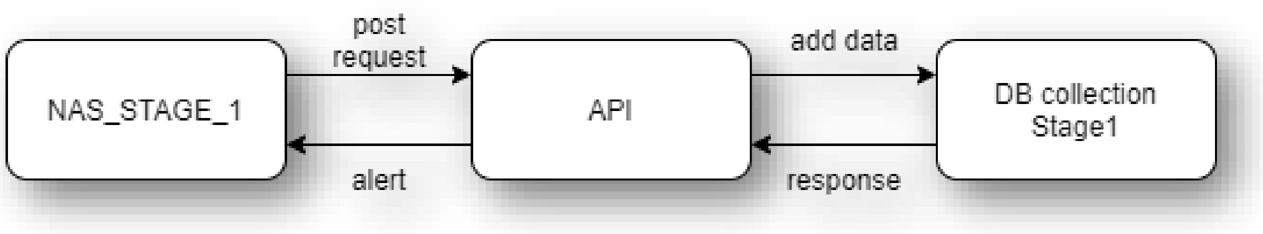


Figure 40: NAS_STAGE_1 Communication

Like most of the components, its wrapped in an authentication protection from Auth0 to only allow authenticated users to render the component, else they are redirected to the front page of the application.

```
export default withAuthenticationRequired(NAS_STAGE_1, {
  onRedirecting: () => <Loading />
});
```

Name: NAS_STAGE_2.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 29, 31, 34, 36, 54

User-Persona: " As a Matron, I would like to swiftly report the daily personnel count of each shift to save time".

Description: Much like the NAS_STAGE_1 component, this component uses the inputs onChange events to trigger the handleChange method that changes the state of the component to match the form. A response is returned to the API that forwards it to the client and displays an alert with dynamic messages from both the API and the DB.

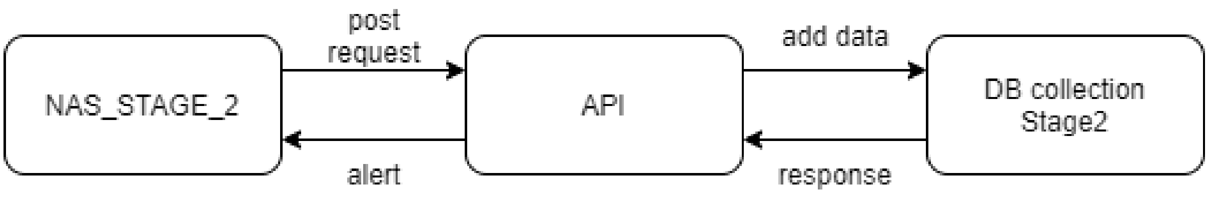


Figure 41: NAS_STAGE_2 Communication

Name: PeerView.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 149

User-Persona: "As a Matron I would like to view and edit both patient and personnel data in one page. ".

Description: This Component acts as a parent component for the peer-view controllers. It uses the react-flexbox-grid react library to render each component inside dynamic flexboxes. These can scale well with all viewports, contributing to a better user experience.

```
<Grid fluid>
    <Row>
        <Col xs><PeerViewStage1/></Col>
    </Row>
    <Row>
        <Col xs><PeerViewStage2/></Col>
    </Row>
</Grid>
```

The columns basic xs mode allows the library to automatically decide on the sizing, fitting each row to cover the entire view-width.

Name: PeerViewStage1.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 150, 151, 152, 153, 154,156

User-Persona: " As a Matron, I wish to view and edit Patient data so that I can remedy errors".

Description: This Component uses the react-bootstrap-table-next library to render and edit existing patient data in the database.

```
import BootstrapTable from 'react-bootstrap-table-next';
```

Each field in the patient (stage 1) JSON has been linked to different column objects in the table. Allowing me to modify how they are shown, as well as adding search fields, styles, and data formatters.

```
...  {
    filter: textFilter(),
    dataField: 'DATE',
    text: 'DATE',
    sort: true,
    headerStyle: () => {
      return { minWidth: '200px' };},
    formatter: (cell) => {
      let dateObj = cell;
      if (typeof cell !== 'object') {
        dateObj = new Date(cell);
      }
      return `${('0' + dateObj.getUTCDate()).slice(-
2)}/${('0' + (dateObj.getUTCMonth() + 1)).slice(-
2)}/${dateObj.getUTCFullYear()}`;
    },
...
```

Filters allows the users to sort data lexically based on the input. The most common lexical filter can be imported from the react-bootstrap-table2-filter library using

```
import filterFactory, { textFilter }
from 'react-bootstrap-table2-filter';
```

dataField refers to the name of the field in the JSON object used as data. In this case it is our patient data objects from the stage1 collection. This variable links the JSON source data fields to this column, allowing the objects fields to appear in its own column, effectively transposing the data.

Text refers to the column label shown to the Matron.

headerStyle can modify the looks of columns, its most commonly used to change colours of columns or change its width.

The formatter is more complicated, it allows us to modify the data in each cell in a specific column to fit our needs. In the example above its used to format the ISO date object to the DD/MM/YYYY format, providing easier to read data to the Matron. This brings a bit more complexity to the usage of this library as we also need to return the date to ISO format before sending it to our API. We can achieve this by importing the cellEditFactory.

```
import cellEditFactory from 'react-bootstrap-table2-editor';
```

This component allows us to modify the rules of how the table modifies data. I added rules to return the data to ISO format after a new value has been modified by the user using the afterSaveCell option.

```
afterSaveCell: (oldValue, newValue, row, column) => {
                if (column.dataField == "DATE"){
                   row.DATE += "T00:00:00.000Z"
                   this.setState({data: this.state.data});
                }
```

This is also where we can save our modified data to the components state, saving our changes until the Matron decides to submit them.

For the Matrons convenience I have also added some alerts that triggers when incorrect data format has been modified in the time fields. All other fields uses dropdown menus to change the data, absolving the chance for incorrect inputs.

The BoostrapTable provides many other utilities that I have made us off:

```
 <BootstrapTable
            keyField="_id"
            data={ this.state.data }
            columns={ this.columns }
            filter={ filterFactory() }
            pagination={paginationFactory()}
            striped={true}
            wrapperClasses="table-responsive"
            filterPosition="top"
            cellEdit={ cellEditFactory({
              mode: 'click',
              blurToSave: true,
              afterSaveCell: ...
 >
```

Except from the fields discussed above, the react-bootstrap-table2-paginator allows us to add pages to the peer-view, making it more accessible by managing how many rows to show per page.

```
import paginationFactory from 'react-bootstrap-table2-paginator'
```

The striped boolean allow us to access a .css class to modify the rows colour scheme using the nth-child.

```
.table-striped > tbody > tr:nth-child(2n+1) > td, .table-striped > tbody > tr:nth-child(2n+1) > th {
  background-color: var(--stripeOdd);
}
.table-striped > tbody > tr:nth-child(2n) > td, .table-striped > tbody > tr:nth-child(2n+1) > th {
  background-color: var(--stripeEven);
}
```

For this table to become responsive it requires a certain wrapper class for one-page applications, hence its surrounded in an empty div that will be populated by the “table-responsive” class.

```
wrapperClasses="table-responsive"
```

cellEdit’s settings allows the user to modify fields by clicking them once, it also accepts blur input, which means that the field changes even if the user does not press enter upon changing it. This promotes swift workflow by minimizing the amount of button presses required by the Matron.

Once the Matron has modified the data and pressed the submit button, the data is being sent to the API that in turn requests a bulk modification of the DB based on the changes. A response is returned to the API that forwards it to the client and displays an alert with dynamic messages from both the API and the DB.

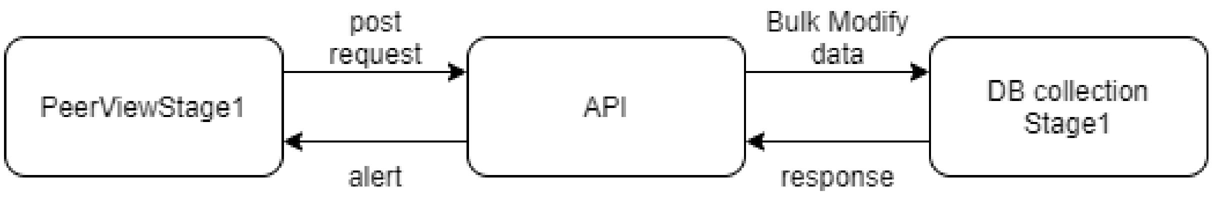


Figure 42: PeerViewStage1 Communication

Name: PeerViewStage2.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 150, 151, 152, 153, 154

User-Persona: “ As a Matron, I wish to view and edit Personnel data so that I can remedy errors”.

Description: The inner workings are identical to PeerViewStage1. But this one acts on Personnel data in the Stage2 DB collection.

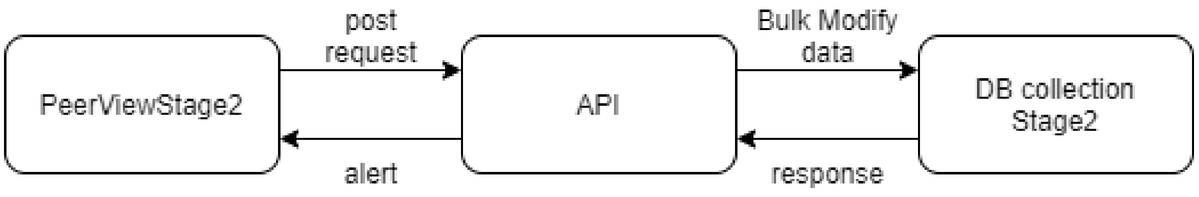


Figure 43: PeerViewStage2 Communication

Name: ReportDashboard.js (GitHub Link)

User-Persona: "As a manager I would like a responsive dashboard so that I don't need to select hospitals for each graph manually.

Stories & tasks: My issues in A301057 numbering: -61 -62 -63 -64 -173 -174

Description: This is the parent of the dashboard components. Its task is to render the dashboard components and pass props and methods to them, allowing them to share data and variables to change their states accordingly. This is done by passing the:

```
updateHospital(hospital) {  this.setState(  {  Hospital: hospital   }  );  }
```

method to the menu component (DashboardMenu.js) and a reference to the variable to all dashboard children components using the props system:

```
<DashboardMenu
    hospital={this.state.Hospital} updateHospital={this.updateHospital}
/>

< < some_child_component > hospital={this.state.Hospital} >
```

This allows the Menu component to communicate its selection to the parent using a reference to the parent's method. Calling said method from inside the menu component changes the state (Hospital variable) inside ReportDashboard.js, which in turn is linked to each child using the same props system. This allows the child components to notice changes in the state of their parent, as their props changes with the parent's state.

Once a child's component prop gets updated it triggers a hook inside the useEffect that reacts to these changes, effectively calling the formSubmit() method to request updated data from the API to re-render itself.

```
useEffect(() => {
        formSubmit()
    }, [props.hospital]);
```

This creates a responsive design pattern where the data in each component updates once ReportDashboard.js receives an updated props.Hospital from the DashboardMenu component.

Name: DashboardMenu.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 161, 172, 177

User-Persona: " As a manager I would like to swiftly change between different hospitals so that I can oversee their status".

Description: This components purpose is to forward the users hospital selection to the ReportDashboard component. A menu button allows the user to select between two different nested functional components that may be used to change hospital. Its onclick method calls toggleMenu() which in turn toggles a react hook boolean called showmenu

```
const [showMenu, setShowMenu] = React.useState(true)

    const toggleMenu = () => {
    if (showMenu) setShowMenu(false)
    else setShowMenu(true)
    }
```

The showMenu boolean is used to decide which menu to render using react ternary logic which returns the correct component based on true or false.

```

{showMenu ? <Dropdown />  : <SelectMenu />}
```

The Dropdown component displays a button grid using the react-flexbox-grid library. The buttons are build using the following pattern:

```
<Col xs={12} sm={5} md={4} lg={3}>
     <button
         className="btn btn-xs btn-outline-secondary btn-block"
         style={{marginTop: '.5rem'}}
         data-toggle="buttonM"
         onClick= {() => {
             updateHospital("Voss sjukehus")
         }}
     >
           <h>Voss sjukehus</h>
     </button>
</Col>
```

React-flexbox-grid allows us to define auto-shrinking on column numbers using the xs (extra small), sm (small), md (medium) and lg(large) viewport types. The current settings allow a lg classified viewport to show 4 columns, md 3, dm 2 and xs 1. xs turns into a long dropdown menu on the smallest viewports. The viewport numbers for this library depends on the innate size of its components and div, hence it's a trial and error approach during development to create the optimal settings.

Bootstrap is used to style these buttons, while the style has been modified using react style. Data-toggle is a Bootstrap JavaScript method, these ended up not blending well with react, only highlighting buttons if they are clicked, the mouse removed and then released, instead of the expected permanent highlighting on a single click. Currently it is a placeholder for further work.

The onClick method calls the updateHospital() method with the current hospital variable. Which in turn passes it to its parent, ReportDashboard, trough props (discussed in detail in the ReportDashboard component)

```
function updateHospital(h) {
        props.updateHospital(h)
    }
```

The other optional menu is the SelectMenu component which provides a simple dropdown menu using a onChange function call to handleChange() which updates the hospital in the same way as the Dropdown component.

Name: ReportNAS.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 33, 62, 133, 173, 174, **135**

User-Persona: "As a Manager I would like to see each hospitals NAS so that I can compare them and notice trends and issues as time passes".

Description: This component uses the react-chartjs-2 library to render NAS data over time in a line graph on the dashboard.

Once the dashboard renders, the hospital is updated or the user submits a new date, this component re-renders itself using useEffect.

```
useEffect(() => {
        formSubmit()
    }, [props.hospital]);
```

The formSubmit() function calls the API to update the graphs data. It firsts gets the active API IP (development or live) from globals.js using the get_ip() function.

```
get_ip().then((IP) => {
        fetch(" < endpoint address with variables >", requestOptions)
        .then(res => res.json())
        .then(json => {
```

It proceeds to chain into a fetch statement to the APIs NAS endpoint using variables for IP, hospital, and dates. Then it converts the API response to JSON so that we can treat it like an object and perform operations on it.

This response holds both patient data and personnel data, we iterate over each object and push its data into three different arrays, holding patient, personnel, and date data separately.

Then I calculate the NAS and push it into a new array:

```
for (let i = 0; i<apiPatient.length; i++) {
    apiNAS.push( (apiPatient[i]/apiPersonnel[i])*100 )
}
```

Finally, I use the setChartData react hook to mount these data structures on the graph, NAS is mounted as the Y-axis opposite to the Date on the X-axis.

```
setChartData({
    labels: apiDate,
    datasets:
    [{
        barPercentage: 0.8,
        label: "NAS per day",
        backgroundColor: […],
        borderColor: […],
        hoverBackgroundColor: […],
        hoverBorderColor: […],
        data: apiNAS
    }]
})
```

As we can see in this snippet the X-axis is called labels and the Y-axis is called data, there are various design aspects contributing to colouring, hovering effects as well as bar width.

While our chartData react hook allows us to customize data, the Line component from react-chartjs2-table also takes an option object that allows more in detail customization of the graphs.

```
<div className="dashboard-item-graph">
    <Line data={chartData} options={chartOptions}/>
</div>
```

Certain chartOptions are important regarding responsiveness in one-page applications.

```
const [chartOptions,setChartOptions] = useState(
    {
     maintainAspectRatio: false,
     responsive: true,
     animation: {
         duration: 2000,
     }, ... }
```

MaintainAspectRatio set to false allows me to stretch graphs in width while retaining the same hight, this is required as dashboards seldom stretch in y-direction.

Responsive set to true allows the graph to react on view port size changes and other resizing interactions.

Animation set to a 2000ms delay allows for smoother transitions between data exchanges, in this wat I do not need to implement a loading screen for the graphs.

Name: ReportPatientNAS.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 68, 69,72, 79, 123

User-Persona: “As a manager I would like to see each days patient NAS to further study the basis of the daily workload”.

Description: The inner logic of this component is as described in ReportNAS with minor changes to design and data.

Name: ReportPersonnel.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 33, 62, 133, 173, 174

User-Persona: “As a manager I would like to see the daily number of personnel so that I can verify the workload resistance ”.

Description: The inner logic of this component is as described in ReportNAS with minor changes to design and data.

Name: ReportPatientWeights.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 33, 62, 133, 173, 174

User-Persona: “As a manager I would like to study the patient workload categories to get a better understanding of the average patient severity”.

Description: The inner logic of this component is as described in ReportNAS with minor changes to design and data.

Name: ReportPatientPersonnelAvgPerShift.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 33, 62, 133, 173, 174

User-Persona: “As a manager I would like to study the workload and workload resistance that make up NAS separately, to draw conclusions regarding the relationship between patients and personnel”.

Description: The inner logic of this component is as described in ReportNAS with minor changes to design and data.

Name: NorwayHeatMap.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 179, 180, 181, 182, 183, 184

User-Persona: "As a manager I would like to observe the average NAS across all regions of my country so that I may notice geographical trends".

Description: This component is only available on the global dashboard, built using the am4chart-geodata library. it shows an overview of the average NAS for each region in Norway. It takes multiple hospitals in each region into account during calculations but requires manual setup for each hospital as the .svg map data provided by the library does not have any coordinate system incorporated. Hence hospitals must be added to their correct regions manually by inspecting the hospitals real location and the regions id in the .svg files. The Norwegian .svg data can be found here: Link

To begin with I need to instantiate a chart object from the am4maps (one of the am4charts assistive libraries) once the component is rendered. componentDidMount() is a react class component method that is being called when the component is loaded.

```
componentDidMount() {
        let chart = am4core.create("chartdiv", am4maps.MapChart);
        this.chart = chart;
        this.formSubmit()
    }
```

Once created, I need the data from the API before I can build other components in the chart object. I will proceed to discuss the data fetching before returning to the chart.

NorwayHeatMap uses a class component as recommended by its documentation (Link). To use the same passing of methods using props that have been used in the earlier functional components I need change how the communication works as react hooks are only usable in functional components. I got around this by using functional binding instead.

```
constructor() {
        super();
        this.handleChangeD1 = this.handleChangeD1.bind(this)
        this.formSubmit = this.formSubmit.bind(this)
        ...
    }
```

This allows me to bind the "this" command of the current NorwayHeatMap component to a certain method, allowing that method to refer to NorwayHeatMap instead of its own scope using "this". Hence, we can use a slightly different version of handleChange(), using the "this" keyword to provide the same functionality as its counterpart using react hooks.

```
handleChangeD1(evt){
        this.state.DATE1 = evt.target.value;
    }
```

At this stage, this component updates in the same way as the patientNAS component.

Because a different library is being used, the data returned by the API is handled differently, here the data must be categorized to all the different regions. Because this specific endpoint sorts it data lexically and always include the same hospitals, we can ascertain that the JSON will be formatted the same, hence we can use it as a regular data structure and access elements by index (this is discussed in future work) and dividing multiple regions by their count to create an average NAS for each region.

```
this.state.data = [
    { id: "NO-01", title: "Østfold", value: json.data[25]["NAS"] },
    ...
    { id: "NO-19", title: "Troms", value: (json.data[6]["NAS"]+json.data[27]["NAS"])/2 },
    { id: "NO-20", title: "Finnmark", value:json.data[3]["NAS"] }
]
```

At this point the component receives data when its rendered and when the user submits a new date. I can now loop back to the creation of the chart.

I define the SVG map to use with the chart:

```
import norwayHigh from "@amcharts/amcharts4-geodata/norwayHigh";
this.chart.geodata = norwayHigh
```

And proceed to create a polygonSeries that holds multiple objects specialised in handing map related data from the am4maps library.

```
var polygonSeries =this.chart.series.push(new am4maps.MapPolygonSeries());
```

The heatRules object defines settings for colouring regions based on their values. It is added into the chart using

```
polygonSeries.heatRules.push({...});
```

After setting the useGeodata boolean, the SVG maps ids are loaded into the chart, allowing us to bind our data that is labelled the same way pre-emptively.

```
polygonSeries.useGeodata = true;
polygonSeries.data = this.state.data
```

To track each zone on a scale, a heatLegend object is instantiated from am4maps. Multiple settings allow customization of positioning, colouring, min, and max.

```
let heatLegend = this.chart.createChild(am4maps.HeatLegend);
```

A zone mouseover tooltip is added using the polygonTemplate, allowing the user to receive the name and NAS of the region

```
var polygonTemplate = polygonSeries.mapPolygons.template;
polygonTemplate.tooltipText = "{name}: \n {value}";
```

To add some flair to the mouseover a colour difference is added, making the current region stand out more

```
var hs = polygonTemplate.states.create("hover");
hs.properties.fill = am4core.color("#ffffff");
```

This hover effect is triggered using both click (hit) and hover (over), calling a standard function to showcase the tooltip, once the region is not hovered on anymore the “out” method hides the tooltip again

```
polygonSeries.mapPolygons.template.events.on("over", function (event) {
handleHover(event.target);
})

polygonSeries.mapPolygons.template.events.on("hit", function (event) {
handleHover(event.target);
})

function handleHover(column) {
if (!isNaN(column.dataItem.value)) {
    heatLegend.valueAxis.showTooltipAt(column.dataItem.value)
}
else {
    heatLegend.valueAxis.hideTooltip();
}
}

polygonSeries.mapPolygons.template.events.on("out", function (event) {
heatLegend.valueAxis.hideTooltip();
})
```

Name: navigationBar.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering:

Description: This navigation decides what main component to render in the software based on its current route using history from react-router-dom, the standard react library for handling addresses.

```
import { NavLink, withRouter, useHistory } from "react-router-dom";
```

clicking a menu button triggers the onClick event:

```
onClick= {() => nextPath("/")}
```

that in turn uses an anonymous react function to call the nextPath() method:

```
function nextPath(path) {
    history.push(path);
  }
```

This method uses history to modify the current address, in a regular webpage this would change which page is being rendered. React is a one-page library, but we can still use these paths to do ternary rendering of components based on the address field.

As discussed briefly in the technical section for react, the App.js component decides which components to render, its main body is decided based on the current address space, or route.

```
<Route exact path="/">
    <LoginPage />
</Route>

<Route exact path="/NasStage1">
    <NAS_STAGE_1 />
</Route>

<Route path="/NasStage2">
    <NAS_STAGE_2 />
</Route>
```

This snipped shows how App.js renders these components based on the current route.

Name: LoginButton.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 162

Description: This is a standard login component from auth0's react setup guide Link.

Name: LogoutButton.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 162

Description: This is a standard logout component from auth0's react setup guide Link

Name: AuthenticationButton.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 162

Description: This is a standard component from auth0's react setup guide (Link). It uses ternary rendering of the logout and login button.

Name: auth0-provider-with-history.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 162

Description: This is a standard wrapper component from auth0's react authentication guide (Link) to secure the app using domain and client id. It uses auth0's onRedirectCallback to supervise how connection are to be made to the application, allowing users to be redirected to auth0's universal login page for authentication and then returned to the page they got redirected from.

Name: profile.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 162

Description: This is a standard component from auth0's react authentication guide (Link) that has been slightly modified. It simply extracts user information and displays it.

Name: LoginPage.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 170

Description: This is a placeholder component for a hospital landing page.

Name: Footer.js (GitHub Link)

Stories & tasks: My issues in A301057- numbering: 165

Description: This is a placeholder footer component for development. It will be replaced with any footer required when deployed at a hospital.

API

❖ This chapter explains the inner workings of the Server and API

Calculate Patient NAS

The result_date_to_daily_patient_nas_collection function in data.controller.js is used to calculate patient NAS for the API, its output is specifically used in the PatientNAS_on_date export. More specifically it is a function capable of converting patient form JSON into patient NAS.

I needed a data structure to hold the NAS for each individual basic activity that is being scored by the system. These units of measurements are the NAS tools guidelines for how much work is required by an ICU nurse to care for the patient based on the basic activities performed. For example, basic activity 1A (“BA1A”) is worth 4.5% of a nurses workday in workload.

```
nas_map = new Map([
    ["BA1A", 4.5],
    ["BA1B", 12.1],
    ...
]);
```

As well as data structure to hold time weights, these are multiplied with the basic activities based on how long the patient has spent in care.

```
hour_map = new Map([
    [0, 0.05],
    [1, 0.05],
    ...
]);
```

The result_date_to_daily_patient_nas_collection function, henceforth identified as ‘F’, may take an array of patient forms and return the NAS per patient per date.

The F does this by instantiating a time_range_map, holding the min in-time and the max out-time of each registered patient per date.

```
time_range_map = new Map();
    for (var i = 0; i < result.length; i++){
        var p_id = result[i].PATIENT_ID
        var p_in = null
        var p_out = null
        if (typeof time_range_map.get(p_id) =='undefined'){
            time_range_map.set(p_id, {TIME_IN: result[i].TIME_IN, TIME_OUT: result[i].TIME_OUT})
        }
```

If the patient already exists in the map, as a patient may appear in all three different shifts, an else statement handles this by calculating time differences.

```
else {
    // add earliest if earliest
    var t_in_old = new Date("01/01/1970 "+time_range_map.get(p_id).TIME_IN)
    var t_in_new = new Date("01/01/1970 "+result[i].TIME_IN)
    var p_date_in = (t_in_new < t_in_old) ? t_in_new : t_in_old;

    // add latest if latest
    var t_out_old = new Date("01/01/1970 "+time_range_map.get(p_id).TIME_OUT)
    var t_out_new = new Date("01/01/1970 "+result[i].TIME_OUT)
    var p_date_out = (t_out_new > t_out_old) ? t_out_new : t_out_old;

    // convert back to HH:MM format
    p_in = p_date_in.toTimeString().split(' ')[0].slice(0,5)
    p_out = p_date_out.toTimeString().split(' ')[0].slice(0,5)

    time_range_map.set(p_id, {TIME_IN: p_in, TIME_OUT: p_out})
}
```

Once the time ranges have been obtained, F creates a time_map that holds the number of hours the patients spent at the ICU based on the latest out-time and earliest in-time of each patient for each date. This map will act as F’s time variable.

The F then proceeds to calculate the basic activities for each patient in a map called ba_tot_map. The F does this by summing the highest workload basic activities among the three shifts for each patient, obtaining each patients raw workload.

```
ba_tot_map = new Map();
    patient_ba_map.forEach((value,key) => {
        ba_tot = 0;
        if (value.BA1C) ba_tot += nas_map.get("BA1C");
        else if (value.BA1B) ba_tot += nas_map.get("BA1B");
        else if (value.BA1A) ba_tot += nas_map.get("BA1A");
...
}
```

One the F has obtained both the total time the patient stayed as well as the sum of their basic activities, it proceeds to calculate NAS for each patient and returns the payload.

```
patient_nas_payload = [];
ba_tot_map.forEach((value,key) => {
    var time_weight = hour_map.get(time_map.get(key))
    patient_nas = value[0] * time_weight;
    patient_nas_payload.push({
        "PATIENT_ID":key, "NAS":patient_nas, "DATE": value[1]
    })
})
return patient_nas_payload
```

Personnel count

The count of personnel’s are returned using a simple additive operation per date. This is mainly used for the Personnel_count export in data.controller.js, allowing the software to receive the count of personnel over multiple dates.

```
function get_personnel(result){
    var personnel_count_count = {
        "Personnel_D":0,
        "Personnel_A":0,
        "Personnel_N":0
    }

    for (const dataObj of result) {
        personnel_count_count.Personnel_D += dataObj.Personnel_D
        personnel_count_count.Personnel_A += dataObj.Personnel_A
        personnel_count_count.Personnel_N += dataObj.Personnel_N
    }
    var personnel_count_payload = []
    personnel_count_payload.push({
        "Personnel_D":personnel_count_count.Personnel_D,
        "Personnel_A":personnel_count_count.Personnel_A,
        "Personnel_N":personnel_count_count.Personnel_N
    })

    return personnel_count_payload
}
```

Patient Categories

The PatientWeights_on_date export in data.controller.js is used summarize the workload categories of patients during a specific time.

It uses the result_date_to_daily_patient_nas_collection function to create patient NAS data and proceeds to classify each patient into different categories.

```
weight_map = new Map([
    ["lost", 0],
    ["<50", 0],
    ["50-80", 0],
    ["81-120", 0],
    ["121-140", 0],
    [">140", 0]
]);

result_data.forEach((patientNas,key) => {
    if (patientNas.NAS > 140) weight_map.set(">140", weight_map.get(">140")+1);
    else if (patientNas.NAS > 120) weight_map.set("121-140", weight_map.get("121-140")+1);
    else if (patientNas.NAS > 80) weight_map.set("81-120", weight_map.get("81-120")+1);
    else if (patientNas.NAS > 50) weight_map.set("50-80", weight_map.get("50-80")+1);
    else if (patientNas.NAS > 0) weight_map.set("<50", weight_map.get("<50")+1);
    else if (patientNas.NAS < 0) weight_map.set("lost", weight_map.get("lost")+1);
    });
```

It returns a JSON with range classified data

```
{"RANGE":key, "NAS_WEIGHT":value}
```

Calculate Patient & Personnel NAS

The ReportPatientPersonnelAvgPerShift export in data.controller.js returns both the patient and personnel NAS (workload and resistance) sorted so that they may be iterated upon based on dates. This method required Queries to two different collections to collect all the required data, this proved challenging with how promises work in JavaScript.

The patient query uses the result_date_to_daily_patient_nas_collection method to calculate its patient NAS and then summarizes the total nas per date and returns it sorted after dates.

```
for (const dataObj of patient_nas_payload) {
    var date_int = +dataObj.DATE
    patient_map.set(date_int, (
        (patient_map.get(date_int)+dataObj.NAS) || dataObj.NAS)  )
}
```

Because JavaScript Map objects cannot use Date objects as keys, I had to create a reducer method to make a token in its place while performing calculations.

```
const groups = result.reduce((groups, data) => {
    const dateShort = data.DATE.toJSON().split('T')[0];
    if (!groups[dateShort]) {
    groups[dateShort] = [];
    }
    groups[dateShort].push({"Pe_NAS":(
        (data.Personnel_D +
         data.Personnel_A +
         data.Personnel_N)),
         "DATE": data.DATE
    });
    return groups;
}, {});
```

Finally the Personnel NAS needs to be divided with the amount of shifts, which in this case is 3, hence:

```
personnel_date_nas.push({"Pe_NAS":(penas*100/3), "DATE": currentDate})
```

With the implementation of a global dashboard, this function suffered the most. Requiring some in depth rebuilding to handle both a singular hospital as well as multiple hospitals.

To solve the issue where sometimes only one of the queries are finished when the data is sent, resulting in half the data being lost, I implemented a Promise.all() field that awaits multiple promises before resolving itself using a small timer delay before summarising and sending the data.

```
await new Promise(r => setTimeout(r, 200));
```

Calculate Geodata

Much like the ReportPatientPersonnelAvgPerShift export, the NAS_mapData export requires dual queries from both DB collections. The biggest difference is that NAS_geoData sorts the average NAS on a per hospital basis.

During the Patient Query, one patient at a time is converted using the result_date_to_daily_patient_nas_collection function. It then adds the patient to a map using the hospital as key, iterating until all patients have been assigned to a hospital.

```
for (const doc of result) {
    patientArr = result_date_to_daily_patient_nas_collection([doc])
    for (const patient of patientArr) {
    patient_map.set(doc.HOSPITAL, (
           (patient_map.get(doc.HOSPITAL)+patient.NAS) || patient.NAS) )
    }
}
```

Then we format our output and sort it

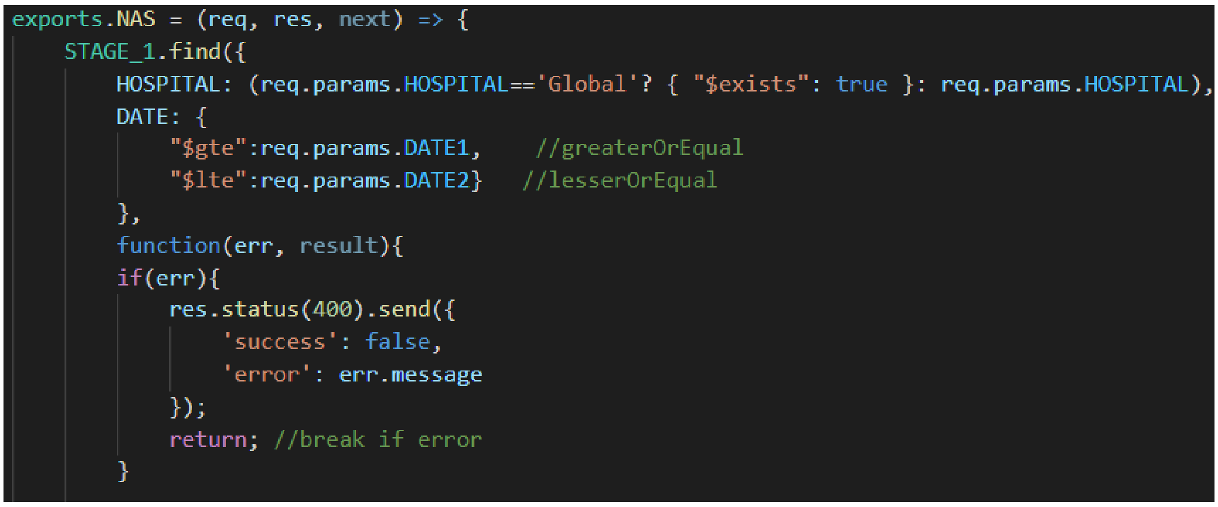
```
patient_map.forEach((value,key) => {
     patient_date_nas.push({"hospital":key, "PAnas": value})
});

patient_date_nas.sort((a, b) =>  {
     var aa = a["hospital"].toUpperCase().trim()
     var bb = b["hospital"].toUpperCase().trim()
     var sum = (aa > bb)?1:(aa < bb)?-1:0
     return sum
});
```

During the personnel query we simply add each patient shift category into one, times it by 100 and divide it by three, creating our personnel NAS on a hospital basis. We format and sort it much in the same way as the patient query.

```
for (const day of result) {
    personnel_map.set(day.HOSPITAL,
    (day.Personnel_A + day.Personnel_D + day.Personnel_N  )*100/3 )
}
```

Mongoose

One of the bigger hurdles with mongoose schemas was when I implemented the global dashboard. As it requires details from all the current endpoints, from all hospitals instead of a singular one between two dynamic dates. This was solved with some of MongoDB's smart tags using the $-sign and ternary operator.



```
exports.NAS = (req, res, next) => {
    STAGE_1.find({
        HOSPITAL: (req.params.HOSPITAL=='Global'? { "$exists": true }: req.params.HOSPITAL),
        DATE: {
            "$gte":req.params.DATE1,     //greaterOrEqual
            "$lte":req.params.DATE2}   //lesserOrEqual
        },
        function(err, result){
        if(err){
            res.status(400).send({
                'success': false,
                'error': err.message
            });
            return; //break if error
        }
```

This code allows the client to receive all the hospital id's if the hospital name is set as "Global" else it returns singular hospitals. The dates are using the greater-then-equal ($gte) and Lesser-then-equal ($lte) tags to create a dynamic date range. As MongoDB takes multiple types of date formats, it took some tinkering to find the correct combination.

TESTING

In the early stages of the application some simple user-testing was done using TeamViewer (Link), a software that allowed my testers to connect to my laptop and control my mouse and keyboard to test the application first-hand. I used a laptop that had the same screen size as the testers so that the software rendered without any scaling from TeamViewer. The tester was a Nurse I had earlier worked with at Kirkenes hospital, who has knowledge about NAS.

The early feedback was regarding the patient form and its usability. The tester mentioned that:

1. The Checkboxes were small and hard to use
2. It was possible to select multiple options, where only one should be allowed.
3. The text font was hard to read
4. After submitting the form, no verification showed up. The tester proceeded to click the button multiple times, expecting some type of interaction with the webpage.

To solve these issues, the following changes were made:

1. Custom big, animated checkboxes were implemented
2. Custom radio buttons were implemented
3. Implementing Bootstrap added a better and more readable text font
4. An alert system was implemented, carrying result from both the DB and the API.

In the later stages a lot of the testing required a big amount of data, hence I carried out tests using different database population scripts to verify correct behaviour of the peer-view controller and the dashboard. These population scripts can be found at Link.

To verify correct output of statistics based on the inputted forms I did the following tests multiple times during development:

1. cleaned the database
2. entered the same data in the forms as I entered my statistical excel model
3. compared the output

During the extend of the project, API tests have been carried out and replaced as required during development in Link.

The last user-test was carried out by hosting the application on EC2, pre-security update. Allowing the testers public ipv4 to connect to the service. This test was regarding future work and improvements of the software.

The feedback included:

1. The main menu was none-descriptive and took a while for the tester to locate.
2. Due to some css changes the patient and personnel form stopped rendering nicely on mobile.
3. The form does not reset data after submitting, so as the tester attempted to add two patients, they had to reset the form manually or by hitting F5 to proceed.
4. The tester was confused if the none-descriptive field “ID” in the peer-view controller was the nurse id or the patient id.
5. Adding “BA” for basic activity in front of each number in the peer-view controller was confusing and made it harder to read for the tester.
6. The tester found it hard to find hospitals as they are not sorted lexically.
7. The initial dark text on dark background in the geo maps scale on the dashboard was hard to view for the tester until the tester realised that hovering a region highlights the numbers in bright colours.

The result of this last user test was interesting, while most of these issues only require minor changes I did not manage to pick up on them beforehand (Except for the menu). This proves to me how important user testing is in all stages of development, especially while working with software that is supposed to alleviate work, not add to it.

There are currently a couple of tools on the global market that provides NAS workload statistic supportive software. Norway’s government uses MetaVision, a fully integrated health management tool that can calculate NAS from their patients journal systems. Compared to my software it also skips the step of reporting the data, automating the entire process. This is only possible when the system is integrated with the healthcare systems.

In this chapter I will talk about a mobile and web application developed in Brazil [11] with the same purpose as my application, to allow nurses to report their NAS.

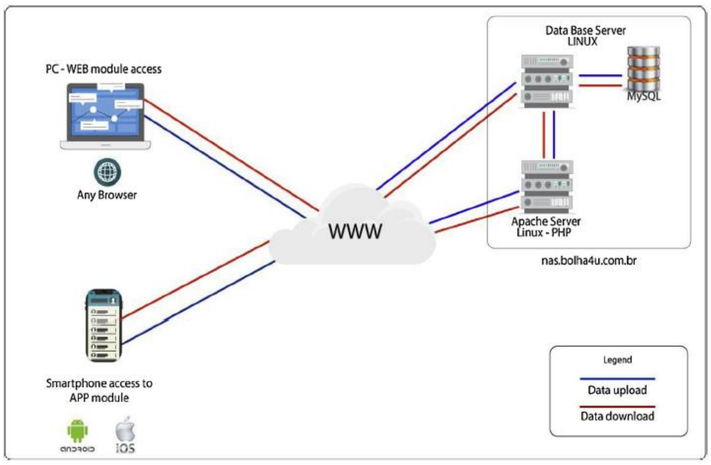
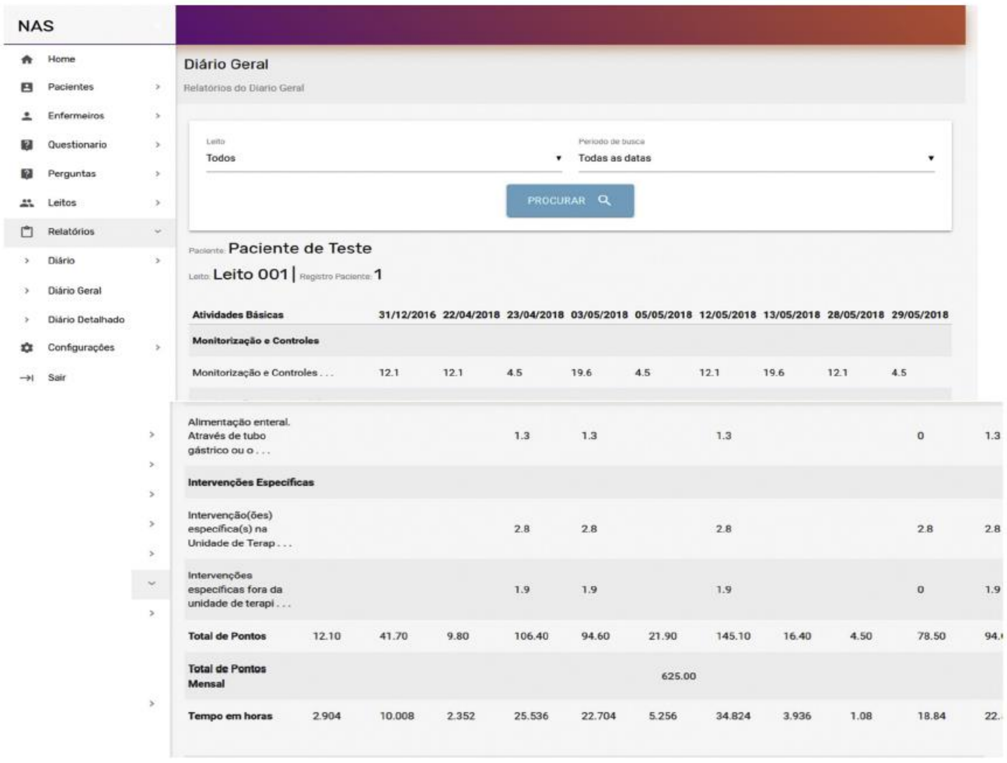


Figure 44: App Architecture

Their architecture is using a PHP backend on a Linux apache server, a MySQL database and a separated web and mobile application. A classic governmental structure, in comparison to mine that uses the same software across all devices.

Their reports are numeric on a day per day basis, compared to mine that uses visualized data across dynamic dates.



Diário Geral

Relatórios do Diario Geral

Leito: Todos

Período de busca: Todas as datas

Paciente: Paciente de Teste

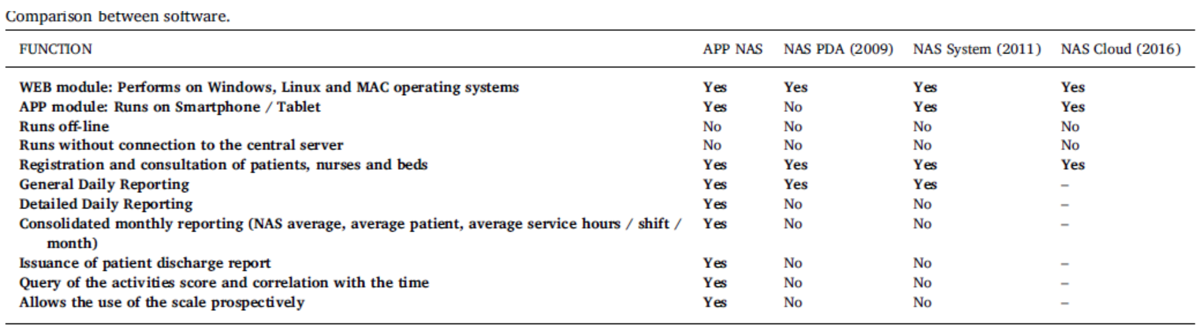
Leito: Leito 001 | Registro Paciente: 1

Atividades Básicas	31/12/2016	22/04/2018	23/04/2018	03/05/2018	05/05/2018	12/05/2018	13/05/2018	28/05/2018	29/05/2018
Monitorização e Controles									
Monitorização e Controles . . .	12.1	12.1	4.5	19.6	4.5	12.1	19.6	12.1	4.5

Alimentação enteral. Através de tubo gástrico ou o . . .				1.3	1.3		1.3			0	1.3
Intervenções Específicas											
Intervenção(ões) específica(s) na Unidade de Terap . . .				2.8	2.8		2.8			2.8	2.8
Intervenções específicas fora da unidade de terapi . . .				1.9	1.9		1.9			0	1.9
Total de Pontos	12.10	41.70	9.80	106.40	94.60	21.90	145.10	16.40	4.50	78.50	94.
Total de Pontos Mensal						625.00					
Tempo em horas	2.904	10.008	2.352	25.536	22.704	5.256	34.824	3.936	1.08	18.84	22.

Figure 45: App Report

And going by their own comparrison between software table my software checks most of the boxes, except that it cannot run in offline-mode because it’s a web application.

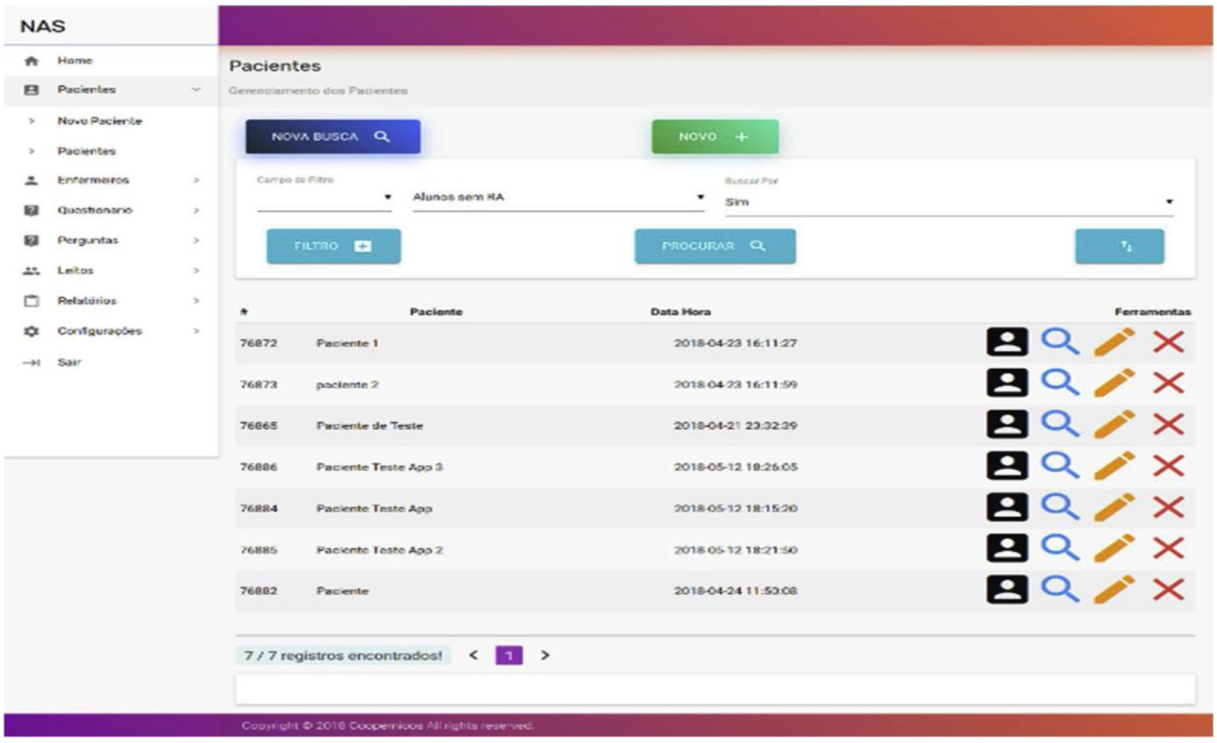


Comparison between software.

FUNCTION	APP NAS	NAS PDA (2009)	NAS System (2011)	NAS Cloud (2016)
WEB module: Performs on Windows, Linux and MAC operating systems	Yes	Yes	Yes	Yes
APP module: Runs on Smartphone / Tablet	Yes	No	Yes	Yes
Runs off-line	No	No	No	No
Runs without connection to the central server	No	No	No	No
Registration and consultation of patients, nurses and beds	Yes	Yes	Yes	Yes
General Daily Reporting	Yes	Yes	Yes	–
Detailed Daily Reporting	Yes	No	No	–
Consolidated monthly reporting (NAS average, average patient, average service hours / shift / month)	Yes	No	No	–
Issuance of patient discharge report	Yes	No	No	–
Query of the activities score and correlation with the time	Yes	No	No	–
Allows the use of the scale prospectively	Yes	No	No	–

Figure 46: App Comparisons

Just like mine they provide a peer-view control, but they’ve designed their system around a questionnaire layout, compared to my full-detail form layout, they only allows searches on three fields compared to mine that uses all fields. Where in mine you may edit forms with a single click, modifying multiple forms at the same time, this software requires a separate modification window that only edits a single form/questionnaire at a time.



*	Paciente	Data Hora	Ferramentas
76872	Paciente 1	2018-04-23 16:11:27	
76873	paciente 2	2018-04-23 16:11:59	
76865	Paciente de Teste	2018-04-21 23:32:39	
76886	Paciente Teste App 3	2018-05-12 18:26:05	
76884	Paciente Teste App	2018-05-12 18:15:20	
76885	Paciente Teste App 2	2018-05-12 18:21:50	
76882	Paciente	2018-04-24 11:50:08	

7 / 7 registros encontrados! < 1 >

Copyright © 2018 Coopernicos All rights reserved.

Figure 47: App Form Layout

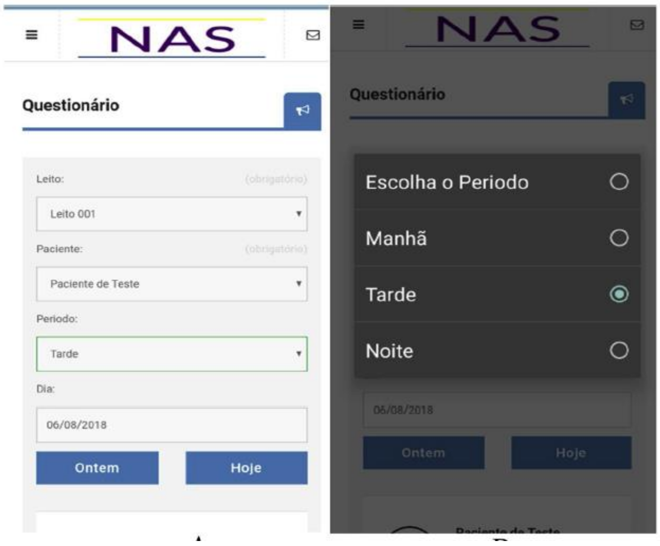


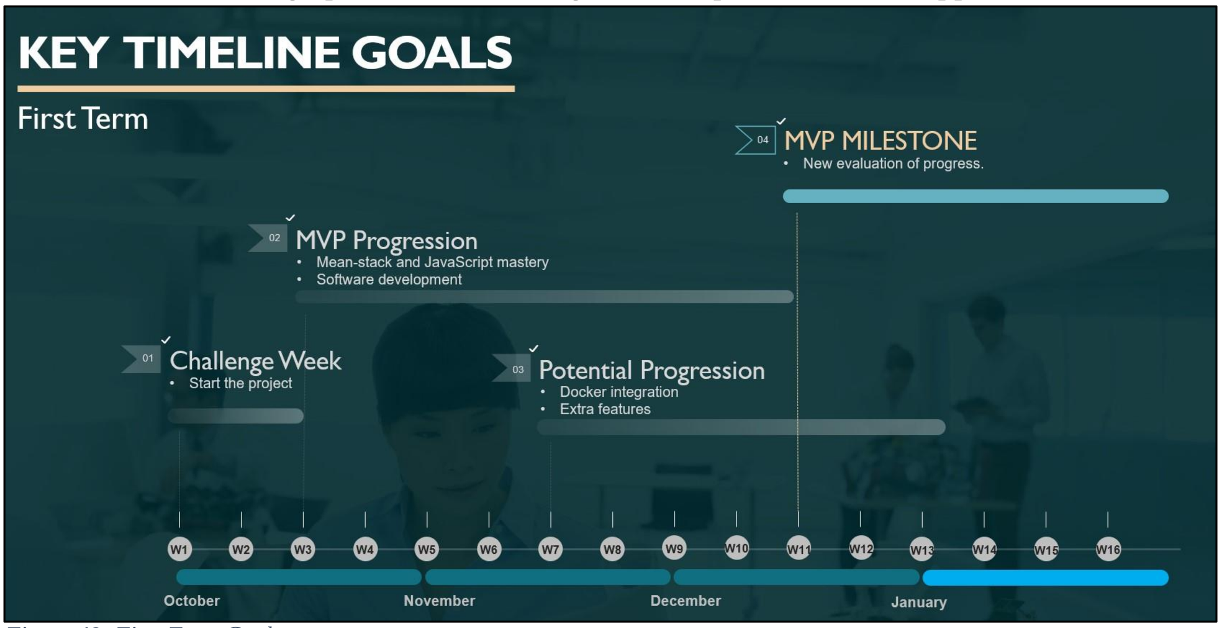
Figure 48: App Login

Their mobile application works much like my patient and personnel forms, where the data is filled in based on the three shifts, patient id, data and so on.

Conclusion:

1. My dashboard has a better visual representation of statistics compared to their numeric reports.
2. My software does not create numerical reports, while their software produces csv formatted reports.
3. As their application is divided between mobile and web, it allows app users to store forms offline before submitting, this is a smart function that my software does not provide.

In the early stages of development, I had to evaluate the risk of using technology I had never used before, such as the MERN-stack and Docker. Without prior experience working with these tools, it was challenging to evaluate how much time would be required to get to the starting point of the project, let alone the end. With the summer reading in mind and my previous minor experience with web development and JavaScript I estimated that my first term would be building up and understanding the basic platform for the application.


KEY TIMELINE GOALS

First Term

04 MVP MILESTONE
- New evaluation of progress.

02 MVP Progression
- Mean-stack and JavaScript mastery
- Software development

01 Challenge Week
- Start the project

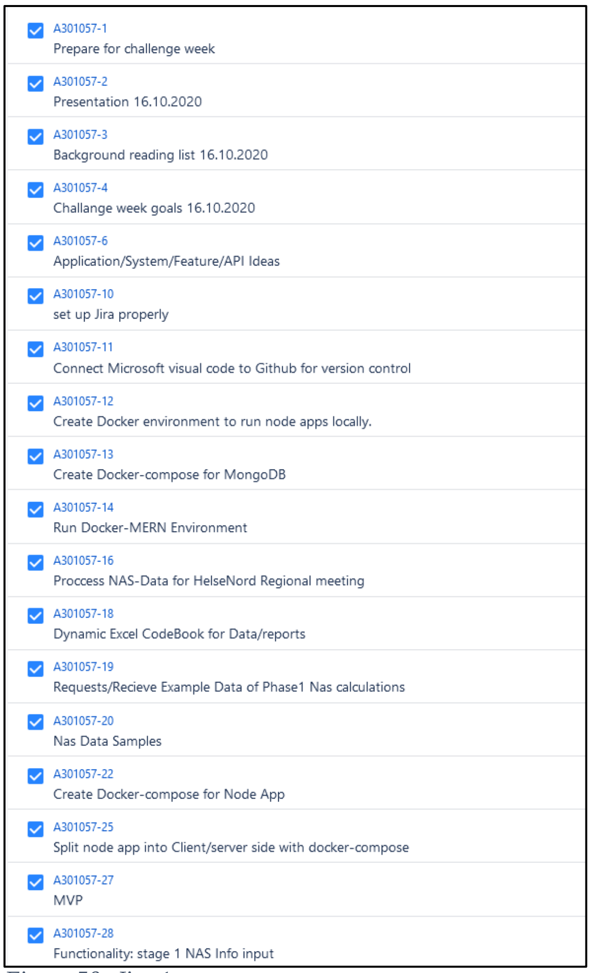
03 Potential Progression
- Docker integration
- Extra features

Figure 49: First Term Goals

During challenge week I assessed that:

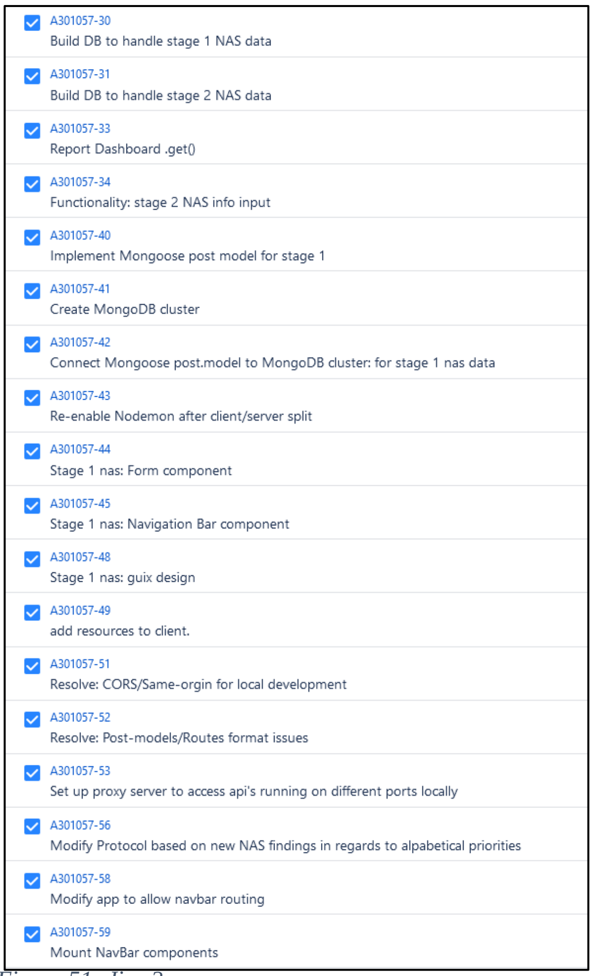
1. I should have learned how to use JavaScript, Docker, MongoDB, JSON, Node.JS, Express.JS and React.JS well enough to develop my software.
2. The basic environment with MERN and Docker would be initialized and working at the end of first term.
3. The project would be built on Essex Gitlab and my own GitHub simultaneously.
4. Some extra features would be implemented in some form.

During the first term, 36 Jira tasks were completed, Including:



- ✓ A301057-1
  Prepare for challenge week
- ✓ A301057-2
  Presentation 16.10.2020
- ✓ A301057-3
  Background reading list 16.10.2020
- ✓ A301057-4
  Challange week goals 16.10.2020
- ✓ A301057-6
  Application/System/Feature/API Ideas
- ✓ A301057-10
  set up Jira properly
- ✓ A301057-11
  Connect Microsoft visual code to Github for version control
- ✓ A301057-12
  Create Docker environment to run node apps locally.
- ✓ A301057-13
  Create Docker-compose for MongoDB
- ✓ A301057-14
  Run Docker-MERN Environment
- ✓ A301057-16
  Proccess NAS-Data for HelseNord Regional meeting
- ✓ A301057-18
  Dynamic Excel CodeBook for Data/reports
- ✓ A301057-19
  Requests/Recieve Example Data of Phase1 Nas calculations
- ✓ A301057-20
  Nas Data Samples
- ✓ A301057-22
  Create Docker-compose for Node App
- ✓ A301057-25
  Split node app into Client/server side with docker-compose
- ✓ A301057-27
  MVP
- ✓ A301057-28
  Functionality: stage 1 NAS Info input

Figure 50: Jira 1



- ✓ A301057-30
  Build DB to handle stage 1 NAS data
- ✓ A301057-31
  Build DB to handle stage 2 NAS data
- ✓ A301057-33
  Report Dashboard .get()
- ✓ A301057-34
  Functionality: stage 2 NAS info input
- ✓ A301057-40
  Implement Mongoose post model for stage 1
- ✓ A301057-41
  Create MongoDB cluster
- ✓ A301057-42
  Connect Mongoose post.model to MongoDB cluster: for stage 1 nas data
- ✓ A301057-43
  Re-enable Nodemon after client/server split
- ✓ A301057-44
  Stage 1 nas: Form component
- ✓ A301057-45
  Stage 1 nas: Navigation Bar component
- ✓ A301057-48
  Stage 1 nas: guix design
- ✓ A301057-49
  add resources to client.
- ✓ A301057-51
  Resolve: CORS/Same-orgin for local development
- ✓ A301057-52
  Resolve: Post-models/Routes format issues
- ✓ A301057-53
  Set up proxy server to access api's running on different ports locally
- ✓ A301057-56
  Modify Protocol based on new NAS findings in regards to alpabetical priorities
- ✓ A301057-58
  Modify app to allow navbar routing
- ✓ A301057-59
  Mount NavBar components

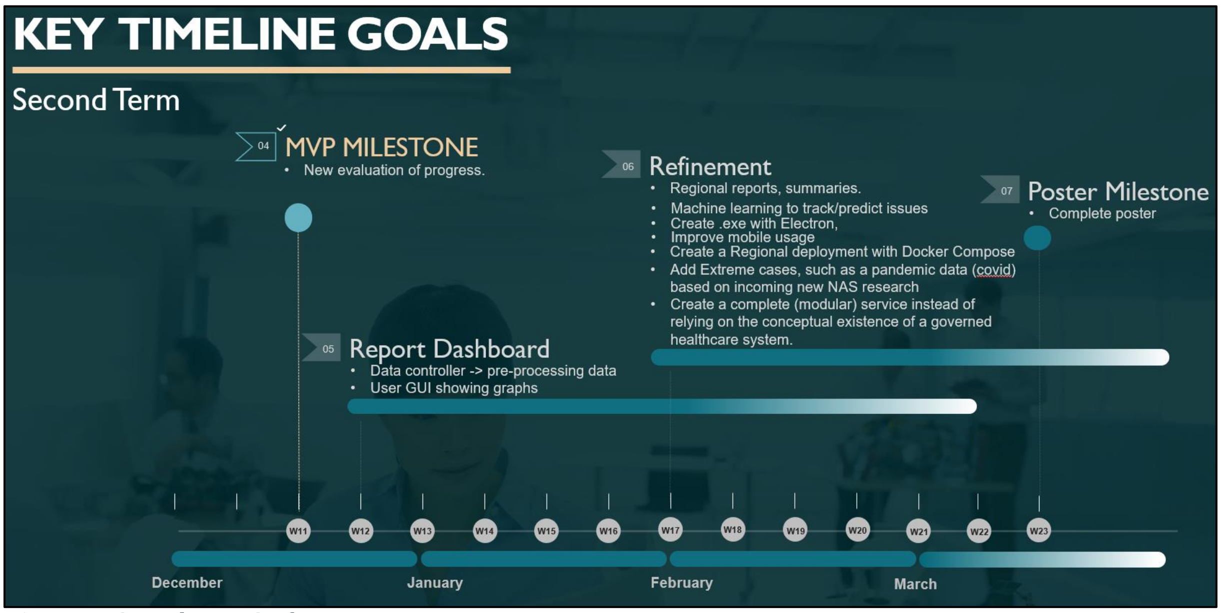
Figure 51: Jira 2

At this stage I managed to finish the basic environment of the project, the software was build using the MERN stack and was deployed by docker containers using docker compose. This fulfilled my two first goals concerning learning the technologies and building the software. I used Visual Code to connect this project to both my GitLab and GitHub, fulfilling my third goal. For extra features I had already:

1. Begun developing the backend and frontend for the forms
2. Added Nodemon to increase development velocity.
3. Built the first version of the navigation bar.

I had to adapt my development environment regarding communication with CORS locally. I had not expected to run into these issues as I was using docker, but as I bound the docker containers to my pc's port it did indeed trigger CORS as it should. I worked around this by adding a proxy server in my docker containers, allowing the client to port forward the communication to the server.

During the second term I estimated to fully implement the application. The Report Dashboard would need to be fully developed and polished.


KEY TIMELINE GOALS

Second Term

04 MVP MILESTONE
- New evaluation of progress.

05 Report Dashboard
- Data controller -> pre-processing data
- User GUI showing graphs

06 Refinement
- Regional reports, summaries.
- Machine learning to track/predict issues
- Create .exe with Electron,
- Improve mobile usage
- Create a Regional deployment with Docker Compose
- Add Extreme cases, such as a pandemic data (covid) based on incoming new NAS research
- Create a complete (modular) service instead of relying on the conceptual existence of a governed healthcare system.

07 Poster Milestone
- Complete poster

Figure 52: Second Term Goals

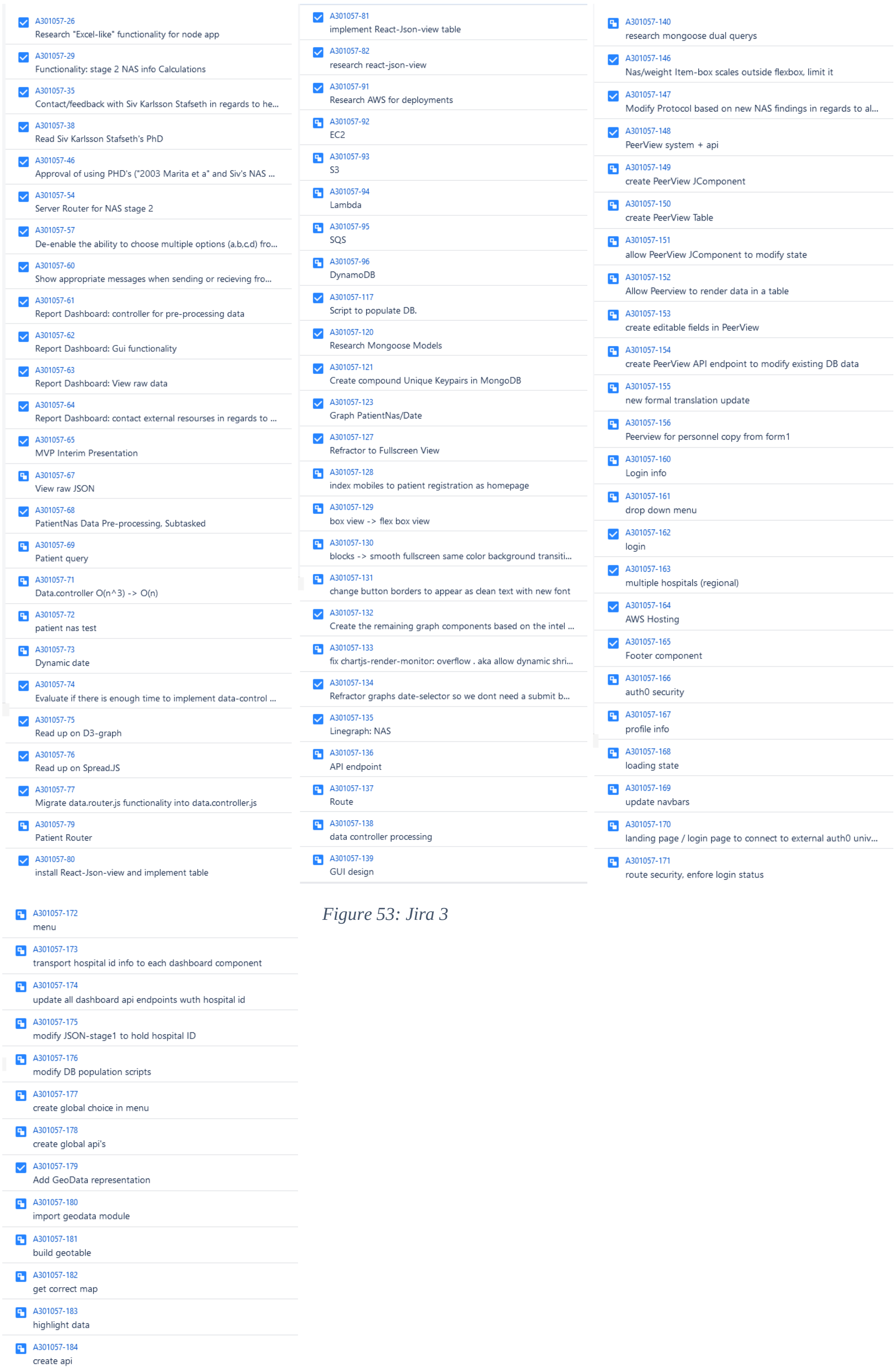
During this time, I was communicating with Siv Stafseth and received some feedback on the project, and I realised that there were some missing elements that should be considered core components. Hence, I had to adapt most of my refinement goals:

1. Machine learning was a stretch goal and was marked as "won't do"
2. Using Electron was an alternative idea and was marked as "won't do" in favour of using this software as a web application. Making it easier to hook onto existing services such as hospital systems.
3. Regional deployment was marked as "won't do" in favour of the web-hosting stretch goal
4. Covid19 research was being carried out on the NAS-tool [2] but it was hard to relate to at this stage. I added the geographical data stretch goal instead, as it can act as an early identifier on how a pandemic spread in a country. This stretch-goal was later completed as a section in the Report Dashboard.

As well as adding a new one:

1. Add a peer-view controller for Matrons to verify and edit data sent in by the nurses. This was later completed as a component.

During the second term, 87 Jira tasks were completed, Including:



A301057-26
Research "Excel-like" functionality for node app

A301057-29
Functionality: stage 2 NAS info Calculations

A301057-35
Contact/feedback with Siv Karlsson Stafseth in regards to he...

A301057-38
Read Siv Karlsson Stafseth's PhD

A301057-46
Approval of using PHD's ("2003 Marita et a" and Siv's NAS ...

A301057-54
Server Router for NAS stage 2

A301057-57
De-enable the ability to choose multiple options (a,b,c,d) fro...

A301057-60
Show appropriate messages when sending or recieving fro...

A301057-61
Report Dashboard: controller for pre-processing data

A301057-62
Report Dashboard: Gui functionality

A301057-63
Report Dashboard: View raw data

A301057-64
Report Dashboard: contact external resourses in regards to ...

A301057-65
MVP Interim Presentation

A301057-67
View raw JSON

A301057-68
PatientNas Data Pre-processing, Subtasked

A301057-69
Patient query

A301057-71
Data.controller O(n^3) -> O(n)

A301057-72
patient nas test

A301057-73
Dynamic date

A301057-74
Evaluate if there is enough time to implement data-control ...

A301057-75
Read up on D3-graph

A301057-76
Read up on Spread.JS

A301057-77
Migrate data.router.js functionality into data.controller.js

A301057-79
Patient Router

A301057-80
install React-Json-view and implement table

A301057-81
implement React-Json-view table

A301057-82
research react-json-view

A301057-91
Research AWS for deployments

A301057-92
EC2

A301057-93
S3

A301057-94
Lambda

A301057-95
SQS

A301057-96
DynamoDB

A301057-117
Script to populate DB.

A301057-120
Research Mongoose Models

A301057-121
Create compound Unique Keypairs in MongoDB

A301057-123
Graph PatientNas/Date

A301057-127
Refractor to Fullscreen View

A301057-128
index mobiles to patient registration as homepage

A301057-129
box view -> flex box view

A301057-130
blocks -> smooth fullscreen same color background transiti...

A301057-131
change button borders to appear as clean text with new font

A301057-132
Create the remaining graph components based on the intel ...

A301057-133
fix chartjs-render-monitor: overflow . aka allow dynamic shri...

A301057-134
Refractor graphs date-selector so we dont need a submit b...

A301057-135
Linegraph: NAS

A301057-136
API endpoint

A301057-137
Route

A301057-138
data controller processing

A301057-139
GUI design

A301057-140
research mongoose dual querys

A301057-146
Nas/weight Item-box scales outside flexbox, limit it

A301057-147
Modify Protocol based on new NAS findings in regards to al...

A301057-148
PeerView system + api

A301057-149
create PeerView JComponent

A301057-150
create PeerView Table

A301057-151
allow PeerView JComponent to modify state

A301057-152
Allow Peerview to render data in a table

A301057-153
create editable fields in PeerView

A301057-154
create PeerView API endpoint to modify existing DB data

A301057-155
new formal translation update

A301057-156
Peerview for personnel copy from form1

A301057-160
Login info

A301057-161
drop down menu

A301057-162
login

A301057-163
multiple hospitals (regional)

A301057-164
AWS Hosting

A301057-165
Footer component

A301057-166
auth0 security

A301057-167
profile info

A301057-168
loading state

A301057-169
update navbars

A301057-170
landing page / login page to connect to external auth0 univ...

A301057-171
route security, enfore login status

A301057-172
menu

A301057-173
transport hospital id info to each dashboard component

A301057-174
update all dashboard api endpoints wuth hospital id

A301057-175
modify JSON-stage1 to hold hospital ID

A301057-176
modify DB population scripts

A301057-177
create global choice in menu

A301057-178
create global api's

A301057-179
Add GeoData representation

A301057-180
import geodata module

A301057-181
build geotable

A301057-182
get correct map

A301057-183
highlight data

A301057-184
create api

Figure 53: Jira 3

All the major goals were completed including:

1. The Report Dashboard was completed.
2. Regional reports were added on a per hospital basis
3. A login system to authenticate users (Atuh0) was implemented to create a more modular service, allowing the system to be accessed in a secure manner without the aid of any conceptional hospital security.
4. A peer-view controller was built

Some minor additions were added to the application as they seemed fitting:

1. Built a footer component

During the security development I realised that AWS EC2 is hosted on a HTTP domain, but Auth0 required a secure origin like HTTPS. This was a complication that was unaccounted for in the scope of the project and while it may seem small, it requires extensive work. This means that the software cannot be showcased on AWS in its final form as of writing this. As this was the first time I developed a web application with a secured login, I was unaware of this risk.

At this point I had to decide if I should either reverse the security updates or add it to future works. I decided to keep it and wrote a small section in related work how this can be resolved in the future.

Due to issues with University of Essex GitLab I used my personal GitHub account for most of the first term. I can see that the repositories contribution has a pattern that is very close to the Cumulative Flow Diagram on Jira (see the next page), therefore I can assuredly say that the commits have been frequent enough.

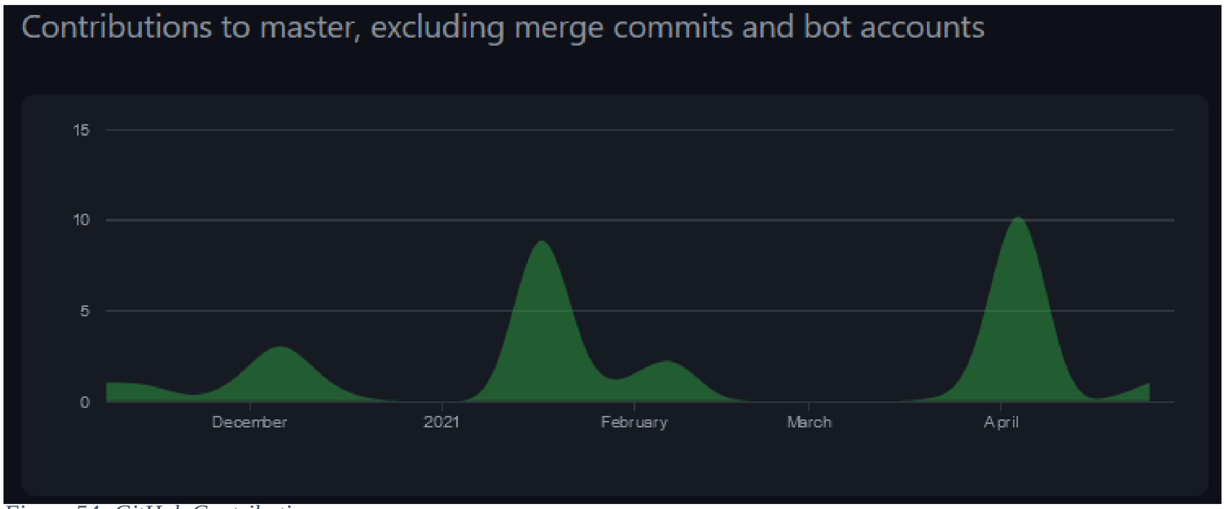
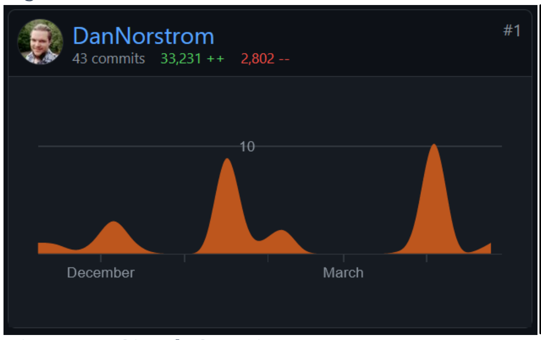


Figure 54: GitHub Contributions



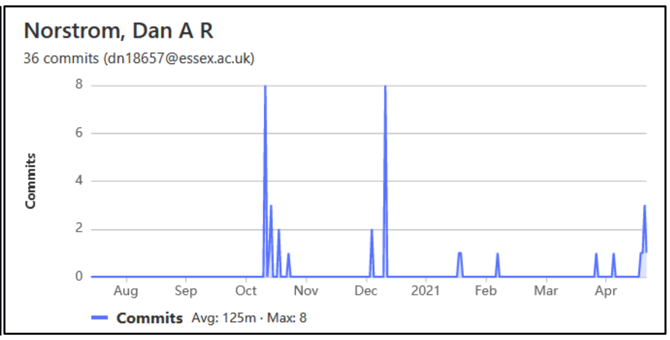


Figure 55: GitHub Commits

Figure 56: GitLab Commits

I began using The University of Essex Gitlab the 7th of December (Link), cloning the repository from my GitLab. I have only pushed to GitLab after big modifications, while I updated GitHub more frequently. There are earlier manual creation and uploads from 13th of October.

Halfway through the project I began adding more details to my commit message. GitHub does not keep more than a month worth of commits, but some of them can be found here (Link). I realised that adding more details will be worthwhile while doing version control. If I want to roll-back the repository to an earlier stage, it is great to be able to establish the state of the software based on the commits (if no releases has been made, as in this case).

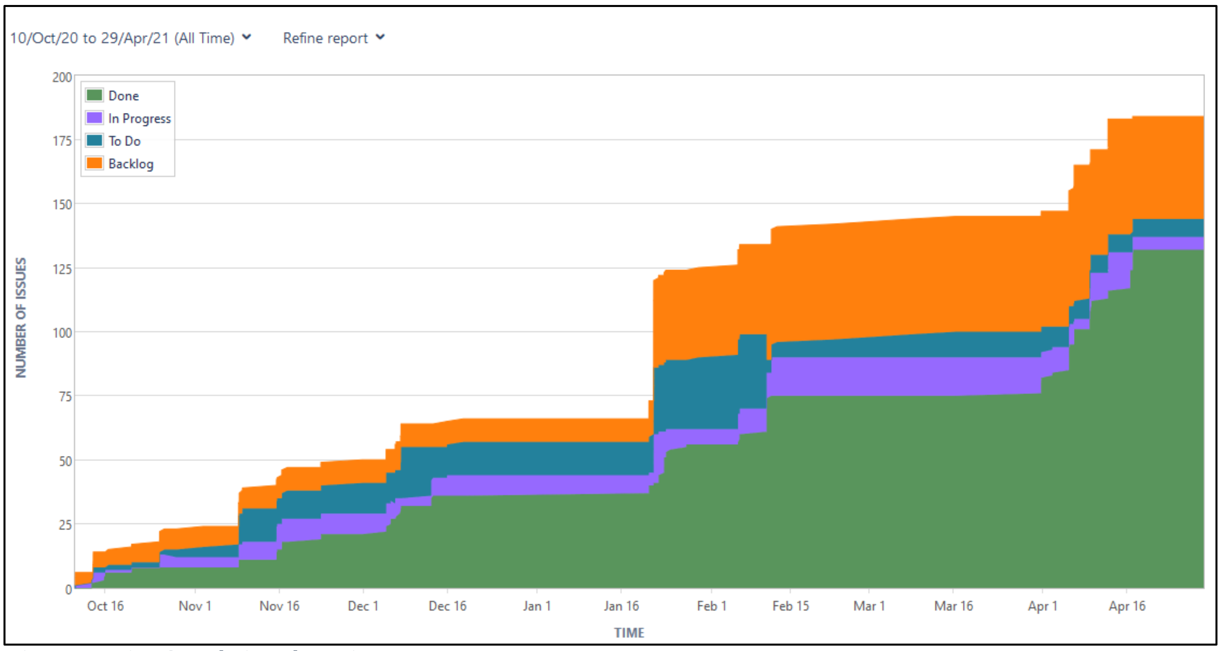


Figure 57: Jira Cumulative Flow Diagram

Looking back on how I managed the project I can see that I had a higher momentum in the second term compared to the first, this was as expected. As I mentioned I had to research a lot of technical documentation and develop the basis of the platform using trial and error. This was a time-consuming process; hence it took long to complete each individual task.

There are two major zero momentum zones, the first one is the winter break (dec16-jan16), while the second one (feb15-apr1) is the break I took to sort out other courses. They were both expected and accounted for.

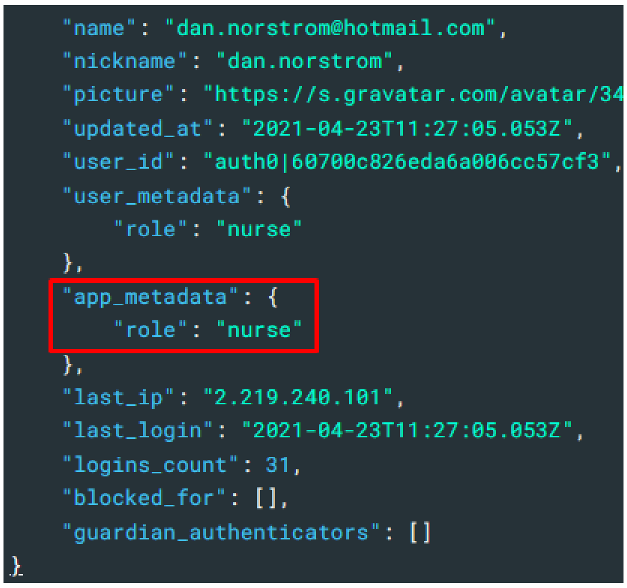
During second term there is a doubled increase in momentum, which is expected as the base application was finished, and it was time to implement functionality.

The backlog generally holds some number of issues, at the time of writing it holds user-personas and research lists. Because Essex University does not allow me to delete issues, some excess backlog is expected at the end of the project.

I can see a clear improvement in my project planning skills compared to the earlier projects I have completed at University of Essex. My time estimation skill for tasks is more robust than before, which allowed me to keep a good momentum through the entire project.

FUTURE WORK

Rendering different application components depending on users roles would allow the application to only render the patient form for nurses, the personnel and peer-view controller for Matrons and the dashboard for managers. This can be done using Auth0's metadata.



```
    "name": "dan.norstrom@hotmail.com",
    "nickname": "dan.norstrom",
    "picture": "https://s.gravatar.com/avatar/34
    "updated_at": "2021-04-23T11:27:05.053Z",
    "user_id": "auth0|60700c826eda6a006cc57cf3",
    "user_metadata": {
        "role": "nurse"
    },
    "app_metadata": {
        "role": "nurse"
    },
    "last_ip": "2.219.240.101",
    "last_login": "2021-04-23T11:27:05.053Z",
    "logins_count": 31,
    "blocked_for": [],
    "guardian_authenticators": []
}
```

Figure 58: User app_metadata

Each user has their own dedicated metadata that is used for identification and authentication. Here is an example where an administrator has granted a user the role of "nurse".

If a hospital would like to implement their own role system and render the applications components based on their own permission systems, this can be set up here.

To finalize alternative rendering the application would need to make an API call to Auth0s User endpoint to extract role information to render the components using ternary operators.

The basic work has been done to implement rendering based on roles in Auth0, but it has not been fully implemented due to a lack of time.

While I secured the application using Auth0, I noticed that it requires a secure domain to function (Link). AWS EC2 is not using a secure origin, causing the application to stop running on AWS after the security updates. To remedy this, extensive work is required to either create a load balancer that can accept SSL/TLS protocols or to write a comparable service by hand. The extent of work required for both parts are way out of scope for this project and will hence be left as future work.

Adding a light and dark mode button would be great and has been a stretch goal in this application for a long time, but it did fall in between more important futures.

In the forms the time in and time out ranges should be limited based on the current shift, As the shift is divided into three section, 00-08, 08-16, 16-24, the time inputs should only allow these ranges, preferably trough a dropdown menu.

The Nurses name, email or other identifier should be added to the JSON as forms are submitted, this will allow Matrons to see who sent in each form in the peer-view component. This in turn allows Matrons to notice nurses that struggle with reporting correct forms, allowing them to do more individual training where its required to increase the correctness of the data being reported.

There should be way to create a report based on the dashboard, preferably a pdf, excel or word formatted file with a combination of csv fields and dynamic graphs. One possibility is to connect the excel file used in the early stages to calculate NAS to the application, allowing it to be written and printed based on the data in the software.

Zooming into the webpage shows unwarranted scaling of components such as the footer and navigation bar.

The Peer-view controller does not allot the deletion of forms, this should be added as it is an expected functionality.

Most of the changes mentioned in the last user-testing will have to be fixed to improve usability in the future.

CRITICAL EVALUATION

Volatile handling of JSON

Because the NAS_mapData endpoint sorts it data lexically and always include the same hospitals, I am expecting that the JSON will be formatted the same when its received by the NorwayHeatMap component, hence I use it as a regular data structure and access elements by index.

This way of handling JSON formatted data might prove dubious if not handled correctly, and if the application gets expanded, this should be improved to validate the data instead.

```
this.state.data = [
    { id: "NO-01", title: "Østfold", value: json.data[25]["NAS"] },
    ...
    { id: "NO-20", title: "Finnmark", value:json.data[3]["NAS"] }
]
```

Security flaw exposes Auth0's domain and ClientID used for authentication.

These constants should have been hidden in environmental files, but it has yet to be changed because of how environmental variables must be integrated in docker containers when using docker compose. This is a security flaw that must be addressed before live deployment.

```
const Auth0ProviderWithHistory = ({ children }) => {
    const domain = 'dev-jekvb0py.eu.auth0.com'
    const clientId = '5NhbgERGXLKv83STgXNBbnVNh0FWZGF2'
```

Numeric Precision

The JavaScript backend data controllers have shown many volatile behaviours regarding mathematical precision. It should be replaced by a more precise backend specialised in data, such as pythons Django or flask.

CONCLUSION

The requirements for this project was to create a NAS software capable of automating all calculations required by nurses at an ICU and convert them to reports for management. The stretch goals included geographical data reports, peer-view components, EC2 hosting, Login, and security. All these requirements and stretch goals have been met.

Using JavaScript and react to create frontend GUI was an enlightening experience compared to my earlier works with AJAX and XML. The way the MERN-stack handles communication and JSON feels a lot smoother compared to my old WAMP-stack. In WAMP I had to create HTML objects in the backend based on queries from the DB with variables from the frontend forms. While this simply updates info, the bigger issue is where I had to render each area in the application, which required excess code. While in React all I had to do was pass some props variables and methods to update each children component. In short, pairing the relevant code and its HTML into components using React allowed me as a developer to keep the code tidy and minimalistic while giving me an easy way to render it effectively compared to the cumbersome methodology in WAMP.

Hosting this application on one of the myriads of different amazon web services showed more challenging than expected. Using docker and docker compose is supposed to alleviate the deployment to cloud applications using the co-operative services between the AWS CLI and Docker, Surprisingly, neither Fargate (Link) nor Lightsail (Link) had any innate functionality to ease deployment for this project. I do believe that I could have made good use of them if I had more experience with Docker, this will be something I will improve on in the future.

I realised that Docker easily claims a lot of RAM and does not let go of it readily. Both my EC2 instance and my local environment ran into issues where simple containers occupied more than 9GB of RAM. Over the course of the project I found multiple ways to handle this, such as limiting the available RAM Hyper-V can use or pruning docker images often. It was a great journey to enter the VM field and manage container applications like Docker.

In my earlier studies I have never had to read as much technical documentation as I have during this project. I picked up some great routines that allows me to find and read documentation quicker.

While building out the APIs, Postman turned out to be an asset to have. The ability to check API responses without using the frontend was a great way to track some errors that was hard to spot from the frontend alone. Giving me as a developer a clear indication if the issue was with the frontend code, the connection to the API, or the APIs treatment of the requests. Using tools like these instead of print() commands was a great experience in debugging.

During the planning phase of this project I assisted Kirkenes ICU in Norway with their report to the Northern health department (Helse Nord), This report was discussed and evaluated to great result, leading to actions that are currently alleviating the nurses at said ICU. It was a great experience to work closely with a hospital and see the real result and improvement that the data science summary of the NAS tool brought about.

While planning this project I had very little knowledge about the technologies I was planning to incorporate, aware that this was a big risk while planning my scope, I expected that my first term would be mostly learning and reading documentation. It turned out that I was right, a major amount of time was required to understand how to apply Docker together with MERN, how to build components and handle communication.

When the bare-bones version of this application was completed during the MVP stages I felt that I had digested some of the complexity of these tools and felt more comfortable using them. My work velocity quickly picked up and I managed to solve issues at a much faster rate. At this point I kept changing the basic components in both frontend and backend as I learnt more and detected flaws and issues, slowly refining my project and knowledge. Before my study break in February I felt comfortable with the technologies and all the expected requirements had been completed. I decided to use my last time well to take a closer look at the responses I got from user-testing as well as my earlier stretch goals, and it turned out that they did not seem to daunting anymore. With my current knowledge I swiftly implemented the last stretch goals and polished the application. The software became much more versatile and allowed more actions to be taken regarding data handling and NAS statistics.

The last low hanging fruits turned out to be the application rendering based on user roles and allowing Auth0 authentication on the EC2 instance, both extensions of stretch goals. While completing the project without these two minor extensions of the stretch goals does leave a bitter aftertaste as I cannot showcase the current software as a live instance hosted on AWS. I do believe that this will act as a valuable lesson regarding how swift scope changes close to the end a projects sprint can greatly affect the outcome. This risk could have been mitigated if I had tested or read about the interaction between Auth0 and EC2 earlier and in more detail.

As far as the result goes, I am pleased that I managed to add all the stretch goals to the project in time. The skills I learnt during the extent of this project will act as a great tool in my continued life as a developer and I look forward to where it will take me.

REFERENCES

[1] Miranda D R, Nap R, Rijk A, Schaufeli W & Iapichino G. Nursing activities score. Critical care Medicine. 2003;31(2):374-82. Epub 2003/02/11

[2] Padilha KG, Stafseth SK, Solm D, Hoogendoorn M, Monge FJC, Gomaa OH…Cudak E. Nursing activities Score: un updated guideline for its application in the Intensive Care Unit. Revista de Escola de Enfermagem da USP. 2015;49 (SPE):131-137.

[3] Esmaeli R, Moosazadeh M, Alizadeh M & Afshari M. A systematic review of the workload of nurses in intensive care units using NAS. Acta Medica Mediterranea. 2015;31:1455.

[4] Hoogendoorn ME, Margadant CC, Brinkman S, Haringman JJ, Spijkstra JJ, de Keizer NF. Workload scoring systems in the Intensive Care and their ability to quantify the need for nursing time: A systematic literature review. Int J Nurs Stud. 2020 Jan;101:103408. doi: 10.1016/j.ijnurstu.2019.103408. Epub 2019 Sep 17. PMID: 31670169.

[5] Lachance J, Douville F, Dallaire C, Padilha KG & Gallani MC. The use of the Nursing Activities Score in clinical settings: an integrative review. Revista da Escola de Enfermagem da USP, 49 (SPE):147-156.

[6] Greaves J, Goodall D, Berry A, Shrestha S, Richardson A & Pearson P. Nursing workloads and activity in critical care: A review of the evidence. Intensive & Critical Care Nursing 2018;48:10-20.

[7] Stafseth SK, Tønnessen TI & Fagerstrøm L. Association between classification systems and nurse staffing costs in intensive care units: an exploratory study. Intensive & Critical Care Nursing. 2018;45:78-84.

[8] Margadant, C C, de Keizer, N F. Hoogendoorn, M E, Bosman, R J, Spijkstra, J J, Brinkman, S.(2021) Nurse Operation Workload (NOW), a new nursing workload model for intensive care units based on time measurements: An observational study. International Journal of Nursing Studies. 021;113. doi.org/10.1016/j.ijnurstu.2020.103780

[9] Bruyneel A, Gallani MC, Tack J, d'Hondt A, Canipel S, Franck S, Reper P, Pirson M. Impact of COVID-19 on nursing time in intensive care units in Belgium. Intensive Crit Care Nurs. 2021 Feb;62:102967. doi: 10.1016/j.iccn.2020.102967. Epub 2020 Oct 28.

[10] K. Mongstad UNN, S. K Stafseth OUS, D. Solms OUS. Norwegian guide for Nursing Activities Score (NAS) Adapted to electronic curves (e.g., MetaVision). Feb 2020

[11] Rosilene DS, Baptista, R.L Serra, D.S.F Magalhãesa. Mobile application for the evaluation and planning of nursing workload in the intensive care unit. International Journal of Medical Informatics 137 (2020) 104120

APPENDICES

The report created from the early excel version of this program, using the same logic and graph types to provide accurate NAS statistics to the healthcare department of north of Norway to create an investigation on the status of Kirkenes hospital

The Helse Nord report I provided data for

Helse Nords investigation result of this report, labelled as control case 154-2020, my data is mentioned on page 16-17.

HelseNords NAS Investigation based on the report

This is a translation of the summary:

> "The working group has discussed a number of measures to increase the robustness of the intensive care unit in a level 2 category. These are presented in this report, but must be considered, processed, and decided by Finnmark Hospital. The professional communities in Kirkenes believe that the staffing of the unit should be strengthened by one nurse per shift at current level 1. With work every third weekend, this amounts to nine 100 percentage positions. It reportedly involves less in fixed expenses than the unit per I day spend on hiring and overtime beyond budget. The academic communities in Kirkenes' proposal for staffing for level 2 is a further increase eighteen employees in 100 percent positions. In periods with few patients / not respirator patient and on weekdays, the staffing will be quite high with such an increase - and it is necessary to discuss in more detail how and whether such staffing can and should be implemented."

And this is a translation of the recommended measures to be taken:

> "Intensive care nurses at the joint emergency department, emergency room and intensive care unit in Kirkenes take care of both nursing and mercantile tasks. It is recommended to add a position for mercantile personnel who can take care of the mercantile core tasks. Scope and the presence of mercantile personnel must be decided after a review of tasks and needs. However, such a resource in the department will be able to make a positive contribution and free up time for intensive care nurse for patient treatment and professional development. With training, nurses without special education will be able to do a good job both in the emergency room and in the emergency room."

Conclusion: A perfect example how the NAS report led to a better environment for the nurses.