placeholder

Master thesis Computer Science

Dennis Cardinaels

Promotor: Professor Mieke Haesen

Assistant: Jens Brulmans

2018 – 2019

Preface

TODO

Abstract

TODO

Contents

1 Introduction

2 The Electronic Patient Record

fill

2.1 Moving away from paper

For modern medicine, traditional paper-based medical records are not suited for today's world filled with technology. The drawbacks of information on paper are obvious when compared to digitally stored information.

Functionality Digital records allow systems to aggregate, process and create statistics of the data it contains. For example, a system can generate heart rate graphs with statistics, summarizing many values. If the user hovers over a data point in the graph, the exact value is shown. Printed tables and graphs showing the same data lack this interaction. Also, paper records demand more effort from clinicians as data is often spread across multiple dossiers.

A paper-based medical dossier can store for example medical images, such as x-rays. Compared to a digital image, paper loses a lot of detail. As such, most multimedia types can only be stored digitally and not on paper. Tools to interact with these file types are provided by an EHRS.

In terms of data input, an EHRS can detect and prevent false data input. The system ensures that all necessary data fields are filled in and in the correct format. This results in more complete and accurate data gathering with fewer errors, increasing the information quality.

Information quality A summarizing paper noted that the use of EHRS leads to more complete, accurate, comprehensive, and reliable data compared to paper-based records[1]. As mentioned in the previous paragraph, a digital system can impose rules on data fields to avoid missing or entering the wrong data. In terms of comprehensibility, poor handwriting leads to wrong or loss of information, which digital systems avoid. Medical instruments can save measurements immediately into system, avoiding copying by hand.

Accessibility Paper records are difficult to access because most of the time only one copy exists. Therefore, transferring these records to other branches or institutions is difficult as the record needs to be located in the often large medical dossier. Also, misfiling, flooding or fires lead to irrecoverable loss of the data. Creating backups of the digital records avoids the last issue, but difficulties surrounding data transfer are still present. Most institutions have their own database structures which hinders the transfer of raw medical data. To remedy this, interfaces are created by the IT staff to interpret and store the data. Sending for example PDF-reports via email is a convenient alternative, although limited in functionality comparable to an actual paper record. Therefore, integration capabilities and the use of standards (section 2.5) are important for an EHRS.

Moving data from paper to a digital format requires a lot of manual work. While automation is possible, such as scanning and processing the paper forms with software, verification is still necessary. All the data fields from the documents need to find a place in the system, which is difficult to achieve. Extra diagnostic information scribbled all over the the document, will be lost during this process. Also, what do we do with unreadable data due to poor handwriting? What happens with two forms which contain partly the same information? Do we save the information twice or do we add extra checks to prevent duplication? Because of these reasons, adopting an EHRS is a significant undertaking for an institution. As such, the benefits are not immediately apparent.

Because digital storage increases accessibility significantly, security measurements have to be taken. If not, data can be stolen, deleted or even altered, which means a breach of privacy (section 2.5). This also adds to the complexity of developing EHRS.

Efficiency An important task of an EHRS is to facilitate effective care. An early study saw a 6% increase in productivity when EHRS were deployed in health care institutions[2]. However, other factors are at play, such as the adoption rate. The transition from paper-based documenting to digital is accompanied by a learning curve, which can be steep. After the switch, the productivity will most likely be lower at the start. As the users become more experienced with the software, it will increase. Today most institutions have already made the transition to an EHRS, so this has become less of an issue.

2.2 Functionality

2.3 Extensibility

2.4 Customization

2.5 Privacy

2.6 Standards

3 Design

fdsafdsa

3.1 Proposal

The proposal features four core concepts which will be combined to form one cohesive solution: a dashboard, telemonitoring of chronic diseases, customization, and integration. As mentioned in the introduction, multiple applications and integrating them pose several difficulties: data is spread and stored multiple times. Interaction between these applications is initially not present and integration with existing systems requires a significant amount of work from the IT staff of the institution. As more applications are added, this becomes increasingly difficult. To remedy this problem, each application becomes a module which can be plugged into the dashboard system.

A module-based approach towards the dashboard allows easy customization and integration. Each module will serve its own purpose by showing data for one particular health parameter. As mentioned in section FILL, parameters such as heart rate and blood pressure are often measured to monitor multiple chronic diseases. Integration is handled by the fact that each module is independent and does not require other modules to work properly. This also helps with customization as a clinician can choose which modules are relevant for the patient in question.

The modules that will be developed are for use in chronic disease monitoring. However, modules can be developed for purposes more fit for in-patient care. Examples are real-time monitoring, documenting diagnoses, and viewing of lab results. Which modules will be implemented and what each one of them tries to achieve is described in section FILl. The implementation section will provide more insight on how this independence is achieved and how future modules can be easily integrated.

This combination allows the clinician to customize the dashboard according to the needs of each patient, facilitating effective care. Data is found rapidly, albeit summarized or detailed, and not hidden away behind multiple screens. Developers should be able to create modules that will fit into the dashboard, negating the use of multiple applications. The following section describes how the system will work and how the users interact with it.

3.2 Dashboard design

Dashboards are used in many contexts. The most common examples include a car speedometer and stock analysis. In computer networking, dashboards are used to quickly interpret network traffic. In our case, we want to gather all telemonitoring data for a specific patient. To create an effective dashboard, multiple design principles should be kept in mind.

While some dashboards provide fancy graphics, many miss the key point of what they are supposed to accomplish. The primary goal of a dashboard is clear

communication, which is achieved through effective design. A formal definition is as follows: "A dashboard is a visual display of the most important information need to achieve one or more objectives; consolidated and arranged on a single screen so information can be monitored at a glance."[3]. This definition contains four key characteristics:

Dashboards are visual displays. It is important that data is represented visually. Displaying charts and other graphics allows more efficient communication compared to textual information. For example, trends and outliers are easier to spot in a line chart compared to a table with the same values. Instead of using a different visualization for the sake of variety, one should always use the visualization which is best suited to effectively communicate the data[4].

Dashboards display information needed to achieve specific objectives. The data that is shown, must be of use for the job that needs to be done. It is possible that the data needs to be gathered from multiple sources and tailored to the specific context so an efficient visualization can be created.

A dashboard fits on a single computer screen. In order to see all information at a glance, scrolling must be prevented. If multiple screens are present, then it is no longer a single dashboard. This leads to another question: what type of display is the information shown on? Due to the wide variety of screen sizes and aspect ratios, a responsive dashboard would be most beneficial.

Dashboards are used to monitor information at a glance. Important data should be immediately noticed, whereas very specific details should be hidden. This means data must be summarized or aggregated in order to be effectively shown. However, if the user wishes to view the detailed data, the dashboard should provide means to do so.

To create an effective dashboard, the user-base must be well known and understood. What type of users are we dealing with? What are their characteristics? These are important questions to ask, as one user may not comprehend one visualization while another one can. This means that the focus should be put on the user[5]. We can ask the following question to better understand the user's needs:

- What metrics does the user need to see?
- What context does each metric require to make it meaningful? Do we need to visualize the variance, target to reach, trend...?
- What visualization best communicates the metric?

On that end, sketches and mockups can significantly help the design process. Multiple iterations, each incorporating feedback from the users, ensure that the end result is satisfactory.

3.3 Personas & scenarios

Based on the proposal mentioned in the previous section, personas and scenarios have been created to get a better understanding of its users and how the system will work. Each of these tries to highlight the problems the users face and how the new system tries to solve them.

3.3.1 Jake

Persona Jake is a 25-year-old male who currently works as a nurse at the hospital in his city. He has been working there for four years and lives alone in his apartment. Still being a young adult, Jake grew up with technology. As a result, he experiences no difficulties when using new software on his computer or smartphone. His hobbies include music, playing the guitar, and video gaming.

Currently, the workflow at the hospital is dated. A new health information system was introduced to summarize and gather all medical data in a singular space. Because this is an all-in-one system, a lot of features that Jake does not need still clutter the screen. Navigating the system is difficult and customization is not present. Jake wishes to only see the features he uses most while hiding the features he does not need.

Scenario Jake starts his first day using the new system by reading the manual that is accompanied with it. He starts the application and for each patient he is presented with a list of modules that can be added to the dashboard. These modules can help with the following: cardiovascular disease, respiratory disease, diabetes, and others.

After selecting a few modules, Jake is presented with a dashboard. The dashboard contains all the wanted functionality of the system, where each "block" represents a module that was added. Jake moves a few blocks around until he is satisfied with the layout. He noticed a module he does not need anymore and promptly deletes it by selecting the option on the module. After a while, Jake got transferred to another department and wants to add a new module to support his new workflow. He opens the module list and selects the he wants, which is then added to the dashboard.

By examining the new module Jake quickly notices that on the dashboard a summary is given. But when Jake clicks on the enlarge button, the module expands to fill a larger area where detailed information is shown. Jake feels he is in control of the system and that it will improve his productivity.

This scenario highlights the benefits of customization. Hiding unwanted components, while adding the useful ones allow for a clear dashboard to be displayed. The user is in control and is able to quickly view data of importance. It is still possible to view detailed data.

3.3.2 Dan

Persona Dan is a general practitioner since he graduated from university. He is on the job for 21 years and he is the preferred doctor in his town. Throughout the years Dan has used a multitude of systems and he always tries new ones to improve his workflow. Because he has been a general practitioner for such a long time, he has 500 patients that visit him at least twice a year. Some patients, especially the elderly, visit as much as once a month.

Most of Dan's patients visit for illnesses such as fever and a cold. To diagnose these illnesses no data is necessarily needed, just a description of the patient's symptoms will suffice most of the time. If Dan performs such a diagnosis, he promptly adds it to the information system and to the electronic health record of the patient.

However, if a patient visits that has a lot of problems regarding his blood pressure, then Dan has to perform a more complex diagnosis using historical data. In the current system that Dan uses, it is very difficult to search for this data. But what bugs Dan the most is that he has to do it every time this patient visits.

Scenario Because of Dan's ongoing curiosity, he tries a new health information system which allows customization for each specific patient. Because the diagnoses of most patients can be very different from time to time, Dan creates a default module group which is displayed for each patient, unless that patient has a specific module configuration. The default module group includes past diagnoses, known allergies, patient information (blood type, last weight, height...), and a current medication list.

The first patient of the day describes what sounds like a fever. Dan confirms this and hands the patient a prescription. The diagnosis is added to the 'past diagnoses' module and the prescribed medication to the 'current medication list' module. Dan did not need other health data to perform the diagnosis. Therefore, Dan does not change the configuration of this patient.

The next patient, an elderly woman, came for her second visit of this month. Dan knows from the past that it will probably be a heart related problem. Dan searches for a 'heart' module and adds it to the configuration of the elderly woman. Now both the default module group and a heart rate module are present, which is unnecessary according to Dan. He removes some modules of the default module group. Now, the next time the elderly woman visits, that configuration will be automatically loaded.

One specific patient had broken his arm three times in less than a year. When the patient came for a routine visit, Dan immediately added a module to easily view x-ray photos and view them in a timeline.

Again, the benefits of system customization for each patient are obvious. Initially, it will take some time to configure each dashboard for all the patients, but the system will try to make this process quick to complete. Once the configuration is done, the time spent during a consultation will decrease.

3.3.3 Emily

Persona Just like Dan, Emily has been a successful general practitioner for quite some time. However, she has different needs of an EHRS. Between each patient visit, there is a period of time in which Emily does not know what happens to patients regarding certain parameters. For example, if a person has to regularly measure his heart rate and blood pressure because of cardiovascular disease, it is imperative that the doctor is made aware of these values. If Emily sees that these values are not looking well, she can contact the patient to come in for a checkup.

If a patient has sleep issues, Emily wishes to not see detailed measurement values, but regular descriptions of the nights sleep. This includes the hours of sleep, amount of wake-ups, subjective feeling of tiredness. Currently, Emily has no way of regularly receiving this information without having the patient visit.

Scenario Emily recently received notice of a new platform that includes telemonitoring support. Several mobile applications are developed that can send data using an API to the platform, which in turn processes the values and can notify Emily of any anomalies. It includes customization for certain parameters in which Emily can individually assign thresholds for each patient.

A patient who recently had a cardiac arrest is continuing rehabilitation at home. However, the patient has chest pains and pays Emily a visit. She tells the patient to regularly measure his blood pressure and heart rate, and to take note of these values in a mobile application. This application also allows taking general notes, such as feeling pain or having caught a cold. After they have scheduled the next visit, the patient is sent home.

As the next few weeks pass by, Emily is notified that this patient has crossed a threshold regarding his blood pressure. Immediately Emily checks the measured data and sees a graph of all measured values. This dataset delivers insight into the history of the patient and Emily sees that there is currently no need to panic. She decides to not take action and configures a weekly reminder to keep monitoring the blood pressure.

The next week, Emily receives a notification that the patient has made a note. It reads that he experienced chest pains. Again, Emily takes a look at the blood pressure data and sees a worsening trend leading to hypertension. Emily decides to call the patient to schedule an early visit. The system helped Emily to intervene as soon as the situation seemed to worsen while avoiding having the patient visit too early which in turn saves Emily time.

Another patient has sleep issues. Emily encourages the patient to use a sleep monitor, which is a wristband. This device is connected to a smartphone which communicates the quantitative data to the new platform. The application on the smartphone allows the patient to take note of qualitative data such as a general description of the night or what food he ate. One night the patient slept only three hours and took note that he went out and drank a lot of alcohol. This could explain for example the bad sleeping rhythm for the next few days. Emily wishes to keep track of this patient on a weekly basis and configures the

platform to notify her.

Here, the effects of telemonitoring are highlighted. By generating reports and alerts, the caregiver can intervene when necessary. Combine this with the aforementioned dashboard platform, greater efficiency of care can be accomplished.

3.3.4 Anna

Persona Anna is a software engineer working at the local hospital. From the early 2000's, she was responsible for the integration of health software systems. As such, she has experienced the continuous change associated with health software. Each unit of the hospital uses different software, while they all share a general system to view patient records. The interaction between these applications is one of Anna's responsibilities.

As time goes on, new software and medical instruments that improve workflow are purchased. Sometimes these replace old systems, otherwise they are added to work alongside them. Both are becoming increasingly difficult to do, as the codebase of the EHRS grows. When an older system gets replaced, Anna has to weed through the code and delete little pieces, hoping other parts wont break. It is a frustrating task as often band-aid solutions are applied.

Scenario A new EHRS was installed that should improve integration of other applications. Anna skims through the documentation and starts the transition process. She quickly notices that for each device or application she needs to create a back end data structure and a module that represents it in the EHRS. The new EHRS is designed that these modules can serve as complete standalone applications or be part of a larger set of modules.

After the initial transition period, everything is put into place. Not soon after, new bedside monitors with accompanying software were purchased. These replace the old bedside monitors and consequently, the software. Thanks to the new EHRS, Anna deletes the old module without much hassle. Also, she reworked the back end data structure to work with the readings from the new device. The data of the old device is ported to the new data structure, so no data is lost. Anna designed the new module to be similar in appearance and functionality. After implementing the module, the staff barely notices any change and workflow resumes with minor disruption.

This persona and scenario highlight the importance of integration. Significant time and effort can be saved if this process is simplified. Also, maintenance is easier due to looser coupling of the modules.

4 Implementation

This section describes the implementation process. The design highlighted the usage of the dashboard through its personas and scenarios, while the low fidelity mockups gave an early glance at its appearance. However, at the heart of this application lies the ability to customize and easily integrate new components. To provide these features, a lot of thought went into choosing the best libraries fit for the job.

First, an overview of the global scope of the application is given, after which we go into more detail. The dashboard consists of two major parts common to web development. The back end is described first. This section covers topics such as the chosen frameworks, the REST API, and data storage. For the front end part, the framework which supports our module-based design and its underlying principles are explained. The contribution of both parts towards a customizable dashboard which allows simple integration, is explained further in their respective sections. It is without question that during the implementation small changes were made to the design. These changes are also documented for both parts. During the entire development process, no effort was put towards privacy safeguards and standard support as they are out of the scope of this thesis.

4.1 Overview

The first major choice that was made, was the type of application to develop: a native application or a web application. It was a relative simple choice, but nonetheless an important one. Native applications are developed, in most cases, for one type of operating system. If a developer wants to support both Android and iOS devices, separate applications need to be developed for each of them. Maintaining multiple applications requires a lot more development effort. Java, for example, has the benefit that it is cross-platform thanks to its virtual machine. However, regardless of platform, platform specific changes need to be made to tackle bugs. One of the largest drawbacks of native applications is updating them. This process often requires the reinstallation of the application. As a result, rolling out software updates in hospitals is a difficult task.

Web applications saw a recent surge in popularity. Web applications today are sometimes very similar in user interface and functionality. An example is Office 365 Web Apps, which shares a lot of functionality from its native counterpart. While these web applications also require updates, these happen on the server side. When the user refreshes the web page, the latest version of the application is automatically loaded. This simplifies the roll-out process significantly. Also, all operating systems that support web browsers, are capable of running this type of applications. As a result, developers only have to develop one version. Mobile devices, therefore, are capable of viewing these applications. However, to fit their small screen, developers need to support a flexible layout.

With this in mind, the choice to create a web application was evident. Users can view the dashboard regardless of the device they use, without sacrificing

functionality. It should be noted that complex image processing applications will not fit easily into the dashboard, due to their computational complexity. Web applications are limited in this regard. Transferring the computational burden to the server side also has the drawback of latency, which the user can experience as slow.

A typical web application consists of two parts: the back end and the front end. In software engineering principles, these refer to the separation of the data access layer and the presentation layer of the web application. For example, the back end fetches data requested by the front end. In turn, the front end presents the retrieved data towards the user. Due to the modular design of the dashboard, each module needs to have its own back end (data retrieval and database models) and front end functionality as summarized in figure. As seen in the figure, there is no interaction between the modules. Furthermore, the front end has a general presentation in which the modules reside. The next sections describe the back end and front end in detail. Also, before the implementation began, I only had basic knowledge of HTML, CSS, and JavaScript. As a result, the learning curve was personally quite steep.

4.2 Back end

The back end consists of a REST API which handles the requests sent by the front end. Each request that the back end supports performs an operation on the database. Examples of such requests are fetching, posting, and deleting data. In this section we describe the REST API, its structure, and the operations it allows. Hereafter, we mention how the database program was chosen, its models and their relations.

4.2.1 REST API

Node.js was used to create the back end server. This server environment was chosen due its large user base and available libraries. The REST API itself was created with the Express framework and tested with Postman. The documentation of the REST API present in the prototype can be found in appendix. We now describe the general structure of the API and the operations it supports.

General structure The API structure starts at the user. Each user has a unique identifier which is passed on to the subroutes. To get the information of the user, we send the following request:

```
GET /user/123456
```

If there is a user present with the id '123456', the API sends a response containing the user information. Modules that want to use data from a certain user, need to prepend their request with `/user/id_value`, as this is the main route. For example, both a heart rate and a blood pressure module want to retrieve their respective values of a certain user whose id is '1a2b3c'. The modules each send the following request:

```
GET /user/1a2b3c/heart
```

```
GET /user/1a2b3c/bp
```

Each module is responsible for its own subroutes and operations it supports. As an example, the threshold values can be retrieved differently for each module, depending on the developer's choice:

```
GET /user/1a2b3c/heart/thresholds
GET /user/1a2b3c/bp/misc_values/thresholds
```

The subroutes are simple to integrate thanks to the Express framework. All the requests a subroute handles are stored in its own file. The main route imports this file so it can forward the subroute towards that module. Suppose we have a `route/heart.js` file that we want to integrate in the main route `user.js`. This is done as follows:

```
const heartRoutes = require('.routes/heart'); //import routes
router.use('/:userId/heart', heartRoutes); //forward routes
```

Only two lines are needed to import all the necessary requests from a module. The main route, and all subroutes of all modules present in the prototype are documented in appendix.

Operations The modules can create new, retrieve, delete, and update data entries. Respectively, these are the following request types: POST, GET, DELETE, and PATCH. In the prototype however, data deletion is not possible because it concerns valuable medical data. For threshold values, deletion is also not possible as every patient is required to have these. Creation of the thresholds for a patient is only possible there is currently no entry present for that patient. Again, appendix documents all implemented operations for all modules present in the prototype.

4.2.2 Database

At the very beginning, SQL was used in the back end, but it was quickly switched out for MongoDB. MongoDB uses JSON-like documents to store data. Groups of these documents are stored in collections. A feature of MongoDB is that the documents inside a collection do not need the same data fields. Suppose we update our patient information page to also show their phone number. In SQL, tables would need to be redefined and the existing rows updated to have a default value for this attribute. In MongoDB this is not necessary as patients with this new attribute can be added without a problem to the existing collection. These flexible documents facilitate integration as no schema rules need to be defined. This allows developers to create their own document schemas and pair them to existing users by solely referring to their id.

The prototype runs a local MongoDB database server where all dashboard data is stored. Each module is responsible for its own documents. As such, every module has a value collection, which contains all values of all users, and a threshold collection, where one threshold entry exists for each user in the dashboard system. No module accesses data from another module. The database is managed via the Mongoose modeling library for Node.js. By defining models,

Mongoose helps us by managing relationships between data. Also, it translates source code objects to their representation in the database.

The models present in this prototype are straightforward, as shown in figure Fig. It is possible that one blood pressure entry can have an additional field, such as whether it was seen or not by the clinician. However, the current implementation of each module, ignores extra data fields. If in the future data fields are added, the front end part of the module must adapted in order to present these to the clinician.

4.3 Front end

Ultimately, the front end is what the user sees. From the onset of the implementation process, the search for a library which supports a modular approach began. It was clear that Web Components would be used to create the individual modules, which will be explained in the next section. There are several libraries that support the creation of Web Components. Google's Polymer library was eventually chosen, due to its extensive documentation. The following sections describe the Web Component principles and the structure of the front end. For the latter, the customization aspect and the library used to achieve this are explained.

4.3.1 Web Components

HTML provides us with standard elements such as `h1` and `img`. Web Components allows the developer to create custom elements which can be used in the same manner as the ones HTML provides. Suppose a calendar Web Component was created, a developer could add the element to any web page as follows: `<calendar></calendar>`. The actual definition of the calendar module resides in another file which is imported on the page where it will be used. This approach supports reusability, and above all, easy integration which is a primary goal of our dashboard application. At the heart of Web Components is the shadow DOM.

Shadow DOM We assume the reader is familiar with the DOM concept. The shadow DOM can be seen as a scoped subtree inside a custom element. The root of this subtree is called the shadow root. The structure, styling, and behavior of the children within the shadow DOM do not affect any elements outside of it. Therefore, the appearance and functionality of the custom element is encapsulated and hidden at the document level. Only events fired by elements in the shadow DOM can be pickup outside the shadow DOM boundary. Figure Fig visualizes this concept. This idea of encapsulation is important for our approach towards the dashboard, as each module is responsible for its own appearance and functionality. Developers of custom modules should not need to worry about what happens outside of the shadow DOM.

4.3.2 Structure

The dashboard features two panels. On the left, only small modules can be pinned, while both the small and large modules can be placed on the right panel. The left panel is thin and is automatically hidden in case of small screen estate. Should too many modules be present on the right panel, a scroll bar appears. The left panel will stay in place if scrolling happens in the right panel to show the information of the small modules that reside there. Figure Fig shows the empty dashboard.

The right panel allows the ordering of the modules via drag-and-dropping them into place. Several libraries were tried out to achieve this functionality, but many of them did not support dragging Web Components. As a result, the front end was rebuilt several times. Eventually, Packery.js was used. This library allows free placement of the modules and tries to arrange them to maximize space efficiency. In case of resizing, the modules are rearranged automatically. Modules are added via a dialog which can be accessed by pressing the plus-sign button in the top-right corner. The same applies for the smaller panel on the left side. However, the modules in this panel can only be realigned vertically.

4.4 Result

Throughout this section multiple screenshots show the final prototype in action. Extra screenshots can be found in appendix Ref, as some modules are very similar in design and functionality. The threshold settings menu is found at the start of that appendix. Changes visible in a screenshot compared its low fidelity counterpart, are explained in the paragraph where it is referred to. First, the small and large modules are shown. Hereafter, multiple complete dashboards are presented to show the different possibilities in terms of layout.

4.4.1 Small modules

Figure Fig shows four small modules, with each showing data of different time periods. For easy arrangement, all small modules have the same width. Note that there is no heart rate data for the time period selected in that module. The only changes present in the small modules are also applied to the large modules. One of these changes was the addition of a small handle bar at the top of each module. This allows safe dragging and dropping. Previously, the entire module was draggable, which resulted in unexpected button behavior, but could also lead to accidental dragging.

The second and last change, was the replacement of the menu button in the top-right corner. This menu was replaced by buttons for each action it would contain: resize, update thresholds, and delete. As a result, a click is saved. The unit of measurement, which was originally in the title, had to be moved to the tables below to make room for the buttons.

4.4.2 Large modules

The larger modules saw additional changes on top of the two mentioned in the previous section. When the large modules were completed, the right-hand side had a lot of whitespace. To remedy this, the statistics were placed beneath the chart. This had the added bonus that the modules became narrower. All large modules have a fixed width of 716 pixels, which is the minimum screen resolution width without needing to scroll horizontally.

The charts were generated using the C3.js chart library. Not mentioned in the design, most charts feature a line showing the average of the values for the selected time period. Indicating the lowest and highest values is difficult to do in C3 and was left out of the prototype. Hovering over the plotted data points shows the exact values, as shown in figure Fig for medication. Labelling the axes and indicating the measurement units was overlooked and added later for all charts. All charts are simple to understand and no "chart junk" is present.

4.4.3 Dashboard

As mentioned in section 4.3.2, different screen sizes are supported, and the left panel is hidden if the available screen space is too narrow. This section shows several examples of dashboards on smaller and larger screens. The empty dashboard was already displayed in figure Fig. The plus-signs on top open the menu seen in figure Fig. Only small modules can be added to the left panel. Figure Fig shows a very wide dashboard, which was automatically resized on a smaller screen to the one shown in figure Fig. If the screen is even smaller, the left panel is hidden as seen in figure Fig. Pressing the button in the top left corner shows the left panel on top the other modules (figure Fig). Extra dashboard layouts are found in appendix Fig.

5 Usability test

6 Discussion

7 Conclusion

A Appendix

appendix

References

[1] Kristiina Häyrinen, Kaija Saranto, and Pirkko Nykänen. Definition, structure, content, use and impacts of electronic health records: a review of the research literature. *International journal of medical informatics*, 77(5):291–304, 2008.

[2] Dwight C Evans, W Paul Nichol, and Jonathan B Perlin. Effect of the implementation of an enterprise-wide electronic health record on productivity in the veterans health administration. *Health Economics, Policy and Law*, 1(2):163–169, 2006.

[3] Stephen Few. Information dashboard design. 2006.

[4] Stephen Few. Intelligent dashboard design. *Information Management*, 15(9):12, 2005.

[5] Richard Brath and Michael Peters. Dashboard design: Why design is important. *DM Direct*, pages 1011285–1, 2004.