A Modern Approach for Patient Waiting Time Systems

Bachelor's Thesis
of

Noor Alrabea
Student Number: 3103436

Faculty of Computer Science and Mathematics
Regensburg University of Applied Sciences
(OTH Regensburg)

Primary Examiner: Prof. Dr. Georgios Raptis
Secondary Examiner: Prof. Dr. Christoph Palm

Date of Submission: September 28, 2020

Eidesstattliche Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtlich und sinngemäße Zitate als solche gekennzeichnet habe.

Datum *Unterschrift*

Abstract

eHealth services enjoy widespread usage and improve the daily lives of practitioners and patients in today's medical sector. Additionally, the prevalence of smartphones enables companies and organisations to offer services in an innovative manner. Despite these novelties, a common nuisance a patient has to endure in a waiting room of a health institute is long waiting hours combined with no information about the remaining waiting time.

In this thesis, a new platform called mPat is introduced as a solution to this issue. It specializes in emergency departments and includes a patient management application, called the back-office, to administer patients waiting for new or further treatment in different departments. The back-office furthermore provides a patient scheduling algorithm based on patient categorization systems employed in current medical centers. Moreover, the mPat system ships with a mobile application targeting patients, which can be connected to an organisation by scanning a personal QR code, generated by the back-office. Through this connection, the mobile application can present further information to the patient, using the data provided by the back-office.

To achieve this goal, mPat is implemented using innovative technologies like ASP.NET Core and Node.js with React, as well as programming paradigms such as REST and single-page applications. The in-depth explanation of these frameworks, the features, the architecture and the inner workings of the patient scheduling algorithm is the main focus of this written thesis.

As a result, the mPat system offers a foundation for a modern patient waiting time system, which can be further extended to be deployable in real-world medical centers. Not only additional features can be added to mPat, but an elaborate empirical study on the effects of mPat on patient satisfaction regarding the waiting time in emergency departments, as well as the performance of the algorithm in different scenarios, are possibilities for future work with reference to this thesis and mPat.

Contents

1 Introduction

When observing daily life regarding healthcare, it is noticeable how indispensable eHealth services have become. From computer-assisted systems and remotely accessible health records to an effortless and straightforward manner accessing medical services or benefits, like booking an appointment online. Numerous studies have been conducted on the topic of eHealth: for instance, one in three individuals in Germany has at least one health-related application installed on their personal mobile device [1], as well as 60% of all internet users, seek answers with reference to health issues [1]. Furthermore, the World Health Organisation (WHO) states, that 87% of the state members reportedly have at worst a single mHealth program in their country [2].

It is apparent that eHealth has a huge impact on our lives, and it is also playing an enormous role in traditional health care centers. In the case of emergency departments, which is the main focus of this thesis, it is common for waiting facilities to be overcrowded and for patients to expect long delay hours. In total, a patient has to wait an average of 2.46 hours during the stay at the hospital [3]. While these hours could be spent more effectively, most patients are forced to wait without prior notice when to be chosen next. With only 28.6% of patient satisfaction level, "Waiting Time" is ranked as one of the lowest aspects related to the service in medical care centers [4].

Since smartphones are widely spread in this day and age with about 3.5 billion smartphone users around the world [5], they enable a novel way for companies and organisations to offer services. The question therefore arises, whether the patient's smartphone might be utilized to offer the possibility, to keep the patient informed about the waiting time at a medical center, without requiring further work of hospital employees.

This thesis tries to build upon this idea by providing a software solution, called mPat. It is divided into a back-office, which manages the patient selection, as well as a mobile application, that enables patients, according to their priority, to get a rough idea

when to be called next. Patients will be categorized within two main groups, new patients and work-in-process (WIP) patients. The former are patients, who recently entered the emergency department and are waiting to be treated for the first time. These patients are classified into four priorities, according to their clinical conditions, via an existing triage system, called the Canadian Emergency Department Triage & Acuity Scale (CTAS). Meanwhile, a WIP patient has already seen a specialist and is waiting for further treatment, professional evaluation, or to be dismissed. Through this categorization, a patient selection algorithm is able to calculate and offer the next suitable patient for the operator of the back-office. The patient can then be called directly via the system and will be aware of this information through the mobile application.

The thesis provides an overview of the frameworks and technologies employed in the system and illustrates the functional requirements, the data model, a listing of the system components, and an explanation of the patient selection algorithm. Lastly, the strengths and weaknesses alongside possible future modifications are discussed.

2 Methods

The following chapter describes how this bachelor's thesis was planned and executed. It details the course of actions taken from the beginning until the end of the project. Furthermore, it provides a thorough introduction of the selected technologies and frameworks, as well as development tools.

2.1 General Approach

At the beginning of this work, elaborate research was conducted to collect various information about the possible features as well as underlying theoretical principles, e.g. for selecting the subsequent patient. As a means of research, examining scientific papers on Google Scholar, scanning the OTH online library, as well as questioning practitioners and patients who visited the emergency department, was performed.

The OTH library was primarily used to find books related to technologies or development tools. The best results were acquired by looking up the name of the relevant technology or tool. Google Scholar was necessary to gather data supporting the motivation of this project, as well as real-world information on how patient scheduling is applied in emergency departments. For the former, search terms like *study eHealth* or *study emergency departments* yielded applicable results. For the latter, queries using keywords like *patient scheduling emergency department* were executed to collect material. The questionnaire of practitioners and patients consisted of unstructured interviews (according to the classification of [6]). It focused on the significance of certain features of mPat and whether the mobile application would provide a possible benefit, during their visit to the emergency department. The results of these interviews were either implemented directly into the platform or are outlined in further detail in

section 4.2, which lists potential further work in reference to mPat.

Based on this research, a patient selection algorithm was designed and technologies, frameworks, and developing tools for the implementation were selected.

The next phase of the project was the implementation. The backend and frontend were developed frontend driven, meaning whenever the frontend required new functionality of the backend side, it was implemented. After the frontend was fully implemented, the implementation of the mobile application began. Similar to the frontend, whenever new data was necessary for the app, the backend was extended.

Finally, the whole project was documented and explained in the thesis at hand.

2.2 Technologies and Frameworks

As already stated in the introduction, the software system is composed of a back-office and a mobile application. For each of these system components, different technologies and frameworks were chosen. In the following sections, these tools are explained in further detail.

2.2.1 MySQL

A database is the heart of nearly every system. To manage all available data, mPat uses MySQL at its core. MySQL is an open-source database system, originally developed by MySQL AB in Sweden, and has a client-server architecture, where the server provides data services to one or more clients [7]. Some of the reasons to make use of MySQL are [8]:

- MySQL is very portable and runs on all major operating systems, like Unix, Windows, and MacOS. Mark Matthews et al.[8] even claim, that MySQL can be executed on every existing operating system. MySQL is not limited to be

operated on various systems but also supports numerous programming languages.

- The database system is highly optimized with many efficient mechanisms and can therefore run most queries faster than other database systems. Hence, MySQL is widely used for web applications, where read operations are frequently conducted and exceptional performance is essential.

- MySQL is very easy to install, administer, and use. Compared to other database management systems, MySQL does not require a big administrative overhead. As a consequence, MySQL is popular among novices as well as professionals.

On account of MySQL's portability and ease of use, it was chosen as a fitting database system for mPat. MySQL is available in different flavours and editions like an enterprise edition or the independent MariaDB. In mPat, the standard community edition in version 8.0.19 is used at the time of writing.

2.2.2 ASP.NET Core

ASP.NET Core is a free and open-source web framework developed by Microsoft Corporation. It is a complete redesign from its predecessor ASP.NET, which was becoming hard to maintain for Microsoft as well as troublesome to use for developers [9].

ASP.NET Core possesses various key benefits, among others [9]:

- **MVC Architecture:** ASP.NET Core allows developers to create applications following the model-view-controller (MVC) pattern. MVC applications are structured in three layers: The persistence layer (models), the presentation layer (views), and an interface between the outside, models, and views (controllers). It is a very popular and widely used pattern for web applications. Systems adopting this pattern have a greatly enhanced separation of concerns and therefore improved code maintainability.

- **Cross-Platform:** While the predecessor ASP.NET was exclusively executable on Windows systems, ASP.NET Core can be developed and deployed freely on Linux and MacOS as well. Consequently, the backend of mPat is cross-platform

and can be deployed on various targets.

- **Modern API:** ASP.NET Core provides many modern features like lambda expression, asynchronous capabilities, and the powerful Language Integrated Query (LINQ). Thus, programs written with ASP.NET Core are cleaner and more expressive than was achievable with prior platforms.

Because of these reasons ASP.NET Core was chosen as a fitting candidate for the backend of mPat. In addition, ASP.NET Core applications can be programmed in three different languages specifically C#, F#, and Visual Basic. The mPat back-office was implemented in C#, being a modern, object-oriented, and type-safe language. The framework version used in this project is 3.1, which is the latest version at the time of writing.

2.2.3 React

React is a modern JavaScript framework developed and released by Facebook in 2013. React targets the inevitable challenges emerging from developing complex user interfaces along with data changing over the course of time [10]. The framework includes a variety of features that allows even novice programmers to construct a single-page application or user interfaces in general [10].

The following paragraphs describing some characteristics of React are based on the content of *Pro React* by *Cássio de Sousa Antonio* [11]:

React focuses on changing the stale page elements alone instead of reloading the whole web page, as was previously done in traditional websites. This concept is called *Reactive Rendering*. In React, the component's look and behavior are declared initially and re-rendered whenever its underlying data changes. To do this efficiently, React has an in-memory representation of the web page called *virtual DOM*. This representation enables React to compute the smallest subset of changes that has to occur in order to update the web page.

A React application consists of a group of independent and concern-specific components. Combined together they produce a rich and complex system. This infrastructure

is described as *Component-Oriented Development*. What adds up to that, is the fact that React uses plain JavaScript to build the website's user interface (UI) instead of being restricted to the shallow sum of HTML directives.

React has been chosen as the framework to implement the frontend of the mPat backoffice. The application was created and is managed by *create-react-app* [1], which sets up React applications with minimal configuration. At the time of writing, the implementation uses the version 16.13.1 of React.

2.2.4 React Native

Another interesting characteristic of React is its abstraction of the document model. More specifically, React is not reliant on the final visualization of the user interface but can be used independently to serve different platforms [11]. This feature comes in play with React Native, where React does not target web applications, but instead mobile operating systems like Android and iOS.

Since React is a JavaScript-based framework, React Native applications are implemented in JavaScript as well. According to Erik Behrends, this aspect leads to multiple advantages [12]:

- It is very easy for web developers to build a mobile application without prior experience because the development of React Native mobile applications is similar to developing web pages.

- Though React Native is not the first and only JavaScript app development framework, it differs in the way in which the mobile app is constructed and executed on a device. While other hybrid frameworks just place everything in a web view, React Native applications use native system elements and have therefore a superior performance.

While JavaScript is already sufficient for building React Native applications, it is also possible to include native code for iOS and Android, if necessary [12].

[1] https://github.com/facebook/create-react-app

React Native is an established technology and many sophisticated tools already exist. For the mPat mobile application, the Expo toolchain[2] was employed. Expo offers an effortless way to develop an app, while directly testing it on a smart device. Furthermore, many important UI elements and components are already built-in, like a QR code scanner, which was necessary for mPat.

During the time of writing, React Native 0.62.2 and Expo 38.0.8 have been used for the implementation of mPat.

2.3 Development Tools

As with any software project, development tools are necessary to be as productive as possible. The following paragraphs describe the primary tools used during the implementation of mPat.

Visual Studio Code (VS Code) VS Code[3] is a free, open-source code editor made by Microsoft. Alessandro Del Sole (see [13]) states that an important trait of VS Code is cross-platform support. Using VS Code, development is not restricted to only Windows, but Linux and MacOS users can benefit from VS Code's features, too. He furthermore lists other characteristics of VS Code like colored syntax, automatic indentation, a built-in debugger, and Git version control support. VS Code was a suitable choice for mPat because it provides excellent tooling for C# as well as JavaScript, the two languages in which mPat is implemented.

Postman When developing a web backend application, it is often necessary to verify whether all HTTP requests are working as expected. A common and widespread tool is cURL[4]. cURL is a powerful terminal based HTTP client which, however, can be cumbersome to use at times. A more modern alternative is Postman[5]. Postman

[2] https://expo.io/
[3] https://code.visualstudio.com/
[4] https://curl.haxx.se/
[5] https://www.postman.com/

provides an appealing visual interface to manage and send HTTP requests to an API. According to the official website, Postman is similar to VS Code, i.e. free and open source as well as available for every major operating system [14]. In the case of mPat, Postman was employed as the primary testing tool to check whether the mPat backend behaves as expected.

Sequel Pro Applications backed by a database require regular management and administration of the stored data. Most database management systems ship with a command-line interface (CLI) to execute commands or queries. Where these CLIs might be too complicated and confusing to use, graphical user interfaces (GUI) come at hand as a straightforward and compact way to visualize and manage the database. As for MySQL, Sequel Pro[6] is an excellent example that offers an effortless approach to interact with the database. It is a free and open-source database management client. Contrary to the other tools, Sequel Pro supports MacOS exclusively [15], which limits its value to a subset of the developer community. For the development of mPat, Sequel Pro served as a simple mechanism to get a quick overview of table structures, as well as to create, modify, and delete data records.

[6]https://www.sequelpro.com/

3 System Illustration

3.1 Functional Requirements

The principal value proposition of mPat is to give patients an easy way to stay informed regarding their waiting time during their visit to the emergency department. To achieve this goal, the mPat platform has to fulfill multiple requirements.

A health institute, in mPat referred to as *organisation*, requires a possibility to access the platform, which should be realized through a previously set *registration number* and an encrypted *password*. These credentials allow the organisation to successfully access the mPat back-office. For security reasons, a single login in mPat back-office should only be valid for a single session. Consequently, if another user logs in with the same credentials, the initial user will not be able to retrieve any data.

An organisation should have the ability to be divided into different *departments*. Therefore the mPat back-office has to offer an overview of each department, including the patients being served in each of them. Furthermore, the back-office should provide an interface to add, modify, and delete departments. Every department should organize patients in different priorities according to their medical condition. Obviously, a method to add new patients to a department is necessary. When adding a patient, the system operator must insert all essential information to ensure the correct execution of mPat. Besides creating new patients, the back-office users should be able to modify patients, in case a mistake has occurred during creation, or to delete patients, if they decide to leave the department without further treatment.

There should be two possibilities to call a patient. First, the mPat platform must periodically calculate which patient should be called next and present this information to the user. Whenever a next patient can be treated, the operator has the choice to

follow the recommendation of the system or not. In the case of the latter, the operator can use the second possibility to select the next patient: the system must offer a manual mechanism to call each patient. An additional benefit of this feature is the opportunity to manually intervene, in the event of a wrong patient selection by the system or an emergency.

On the mobile application side, every patient has to have mPat installed on their mobile device, in order to benefit from the additional waiting time information. To connect their instance of the mPat application with the back-office of the health organisation, a patient should have two options. After a patient has been initially added, the back-office should provide the ability to either show a QR code or a plain text id for the patient. Therefore, a patient can just scan the previously mentioned QR code and is immediately connected to the organisation's back-office afterward. In case the patient's camera is not working correctly, the second method acts as a fallback solution. The mobile application should offer an input box to the patient to enter the plain text id, so the usage of a camera is not necessary.

After the connection between the mobile client and the back-office has been established, the mPat application should provide as much information as possible regarding the waiting time. This may include an estimate of the remaining minutes that the patient has to wait or a position in the waiting queue. The mobile application must regularly contact the back-office to check whether the patient has been called by the operator or not. Whenever the mobile application detects the selection of the patient by the back-office, it must present this information immediately to the user.

3.2 Data Model

According to the functional requirements in section 3.1, a data model had to be designed to fulfill these specifications. Figure 1 shows the result of this design and will be explained in further detail in this section.

The mPat data diagram consists of three main tables that together build a hierarchy. The root of this hierarchy is the *organisation* table that represents the health institute. The next level of the hierarchy consists of the *department* table. The relationship be-

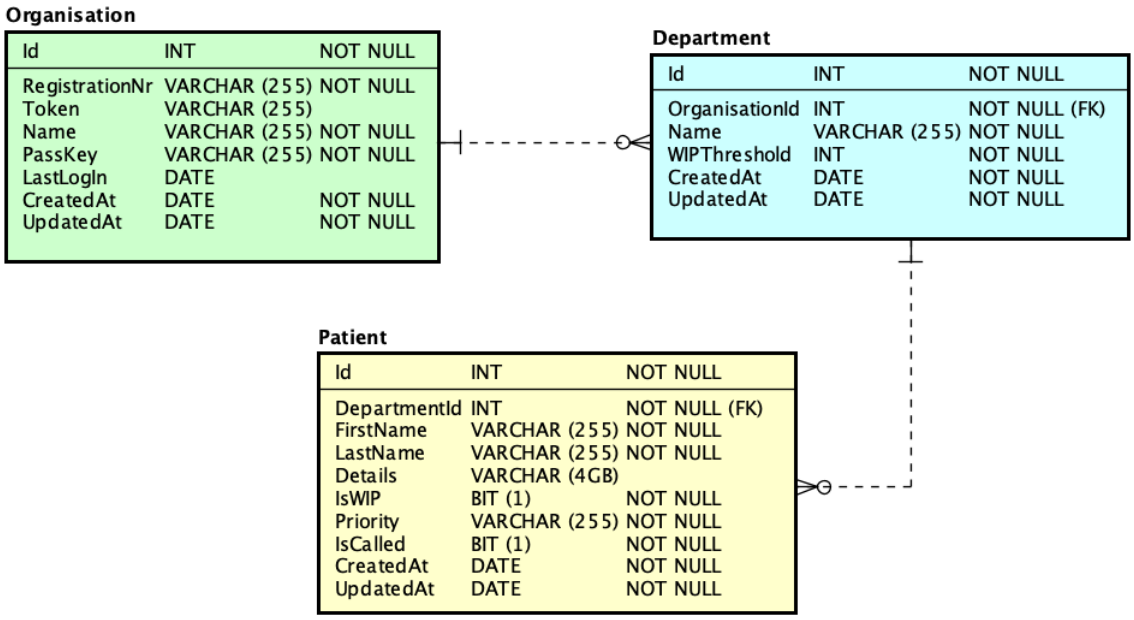

Figure 1: ER diagram of the mPat data model

tween these two tables is one to many, meaning each organisation has a capacity for multiple departments. A department, however, can only be related to a single organisation. At the bottom of the hierarchy is the *patient* table. The association between departments and patients is again equivalent to the association between organisations and departments. One patient can only be assigned to a single department, while a department consists of multiple patients. The information, which patient belongs to which organisation, is modeled in a transitive manner. The patient has no direct connection to an organisation, so this information must be obtained by first fetching the department and then the organisation.

As visualized in figure 1, all three tables repeatedly contain three similar attributes *Id*, *CreatedAt* and *UpdatedAt*. The first is a numeric type, while both of the last-mentioned ones are of type *DATE*. The *Id* grants each row in any table a unique identifier and acts as a primary key. The columns *CreatedAt* and *UpdatedAt* hold the date and time on which the object was stored for the first time, and the last time the object ran through any possible modification, respectively. The reason behind these two date and time columns is to be able to investigate or reproduce problems.

The organisation table holds all information about an organisation as well as credentials with which a user can authenticate. The *Token* holds the currently active, randomly

generated string, which the frontend sends on each request to indicate the organisation behind the request. The *RegistrationNr* and *PassKey* serves as the user's username and password combination. Naturally, the *PassKey* is not stored as plain text, but as a hashed string using the *SHA-256* hashing algorithm which, according to researchers, provides enough security for the coming years [16]. Lastly, the *LastLogIn* column contains the date and time of the last visit of the user to the platform.

Meanwhile, the department table contains an *OrganisationId* as a reference to the organisation table. It furthermore includes a *Name* as well as a *WIPThreshold*, which serves as a possible setting to configure the patient selection algorithm, that will be extensively explained in section 3.4.

Finally, the patient table has likewise a *DepartmentId* as a reference to the department table. Besides, it consists of *IsWIP*, *Priority* and *IsCalled*. The first field distinguishes the patient between a new patient and a WIP patient. The difference between the two and what consequence each type of patient might result in will be explained in-depth in section 3.4. The second classifies each patient into four different categories of urgency according to their injuries. At last, the third specifies whether the patient has been called yet or not.

3.3 System Components

The following subsections explain each component of mPat in further detail and highlight some interesting aspects of the implementation. The architecture is visualized in figure 2 and provides an outline of the components and their communication.

3.3.1 Backend

It is clearly visible in the architectural diagram, that the core of mPat is its RESTful backend. REST stands for Representational State Transfer and is a state of the art architectural style to build distributed systems [17]. This style was chosen for the mPat

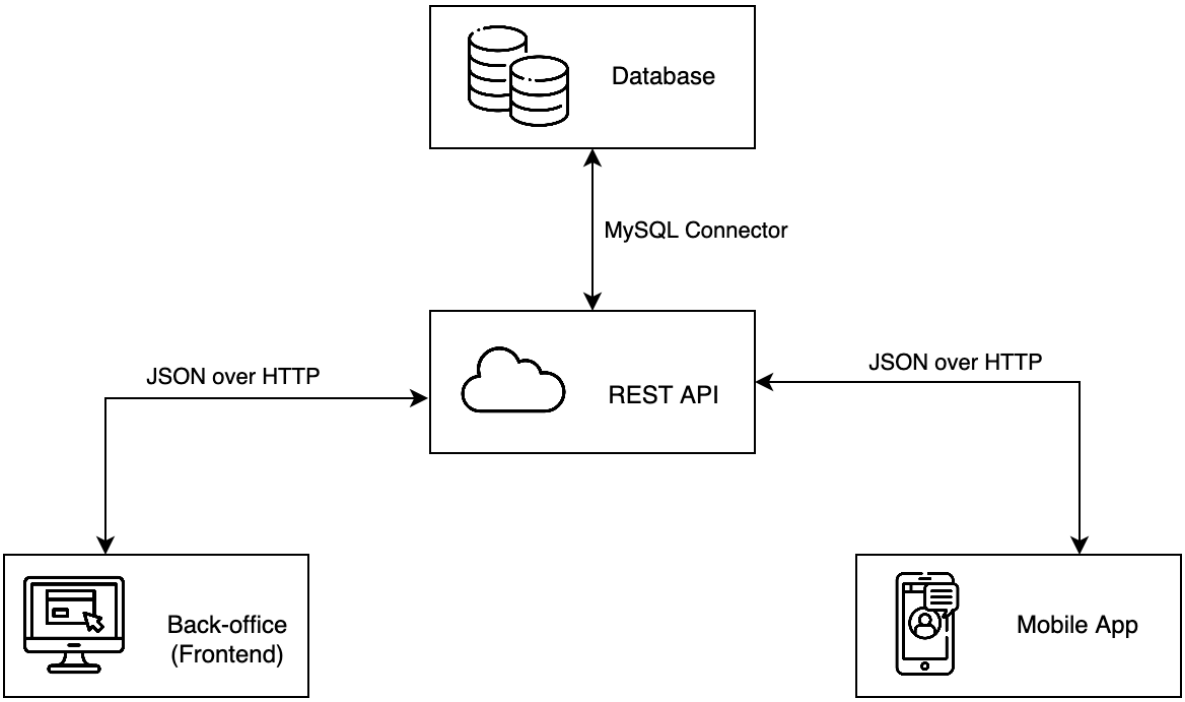

Figure 2: Architecture of mPat

backend due to its elemental feature: statelessness. In a stateless communication, the server does not store any information between subsequent requests of a client, instead, the client has to send all required data, in order for the server to be able to serve each request [17].

In general, REST applications are not limited to but often combined with JSON, as data representation, and HTTP as a transport protocol. JSON is a short term for JavaScript Object Notation and is the current standard choice to send data back and forth, between the server and the client, due to it's lightweight and human-readable syntax [17]. On the basis of these qualities plus the ability to be immediately parsable by JavaScript, JSON has been chosen as the main data trade format between the REST API and both frontend and mPat mobile application.

As mentioned in the methods (see 2.2.2) the mPat backend is based on ASP.NET Core and is similarly architectured as a model-view-controller application. The only difference between a strict MVC backend and mPat is the extraction of the view layer of mPat to a separate project. The backend offers controllers for departments and patients, that provide all create, read, update and delete (CRUD) operations for the back-office frontend to use. It furthermore provides a controller for authentication,

which will be explained in further detail shortly. Lastly, another controller exists, that serves the requests of the mobile application. Models in the backend are plain data containers that store the information returned by the database and are additionally used to serialize and deserialize data from and to JSON. An addition to the traditional MVC architecture is the *service* layer in mPat. Services provide all necessary functionalities to load and modify data in the database, as well as required business logic to serve the requests of the frontend and mobile client.

The statelessness of the mPat API results in interesting implications, concerning the security and authentication of the frontend. Since the REST API is stateless, it must not store any data about the currently logged in user, often referred to as *sessions*. To solve this problem, the mPat front- and backend exchange tokens. As explained in section 3.2, each organisation stores a registration number, a passkey, and a token. Whenever a user logs in, the frontend sends the registration number and the passkey to the backend. The backend compares the former and a hashed version of the latter with the stored credentials in the database. It then generates a random token using a cryptographically secure random generator provided by C# and stores it in the database. This token is then returned to the frontend and saved by React in a state variable. From now on, the frontend sends this token to the backend within an HTTP header for each request. This way, the backend can recognise the user by comparing the given token with the one currently saved in the database. Logging out is therefore as simple as deleting the token. Using a token versus sending the login credentials each time, has an important security benefit. If the token were somehow obtained, it would only be valid until the next login. However, the registration number and the passkey would be valid for future logins as well.

The REST API is designed to be hosted on a server, both cloud or in-house servers are possible. The MySQL database can be located on the same or on a different remote server and is connected to the REST API via a C# MySQL Connector library. Both the frontend and the mobile application need to have the correct URL of the REST API as a configuration to work properly. This results in flexible hosting for a variety of different situations and requirements.

3.3.2 Frontend

As mentioned during the explanation of the methods, the React framework is commonly used to create single-page applications (SPA). Implementing a frontend as an SPA is a modern alternative to the traditional way of building websites. In an SPA, all static resources are fully loaded on the initial request and the view is updated by replacing page components with other components according to the interaction of the user [18]. This results in a faster response of the application since the page does not have to be reloaded each time a user interacts with the frontend. To load data from a server, an SPA generally utilizes Asynchronous JavaScript and XML (AJAX) where requests can be dynamically issued whenever necessary [18].

The mPat frontend is architectured as such an SPA and is divided into React components. The components of the mPat frontend are structured in a tree-like composition, with one root component storing all the data of the frontend in its state and all other components, which build up the view, as direct or indirect children of the root.

On the one hand, this approach solves the token-based authentication elegantly. The token is stored in a global state variable in the root component and remains accessible throughout the existence of the application. On the other hand, the navigation through different screens has to be manually implemented, since the frontend consists physically of just a single page. In the mPat frontend, the currently active screen is set, similarly to the token, as a state variable. The root element of the application uses these variables to display the correct child screen to the user. Whenever the user navigates to a different screen, the state variable is updated and subsequently, the visible components are changed.

The frontend is compromised of three screens: the login page, the department settings page, and the main homepage. The main homepage, shown in figure 3, takes all the raw patient data and aggregates it to an informative view. On the top of the page, a large button is rendered, which contains the current recommendation of the algorithm as a label. The user can choose to click on this button, whenever a next patient can be taken. The patient is then notified on the mobile device and is not listed in the back-office anymore.

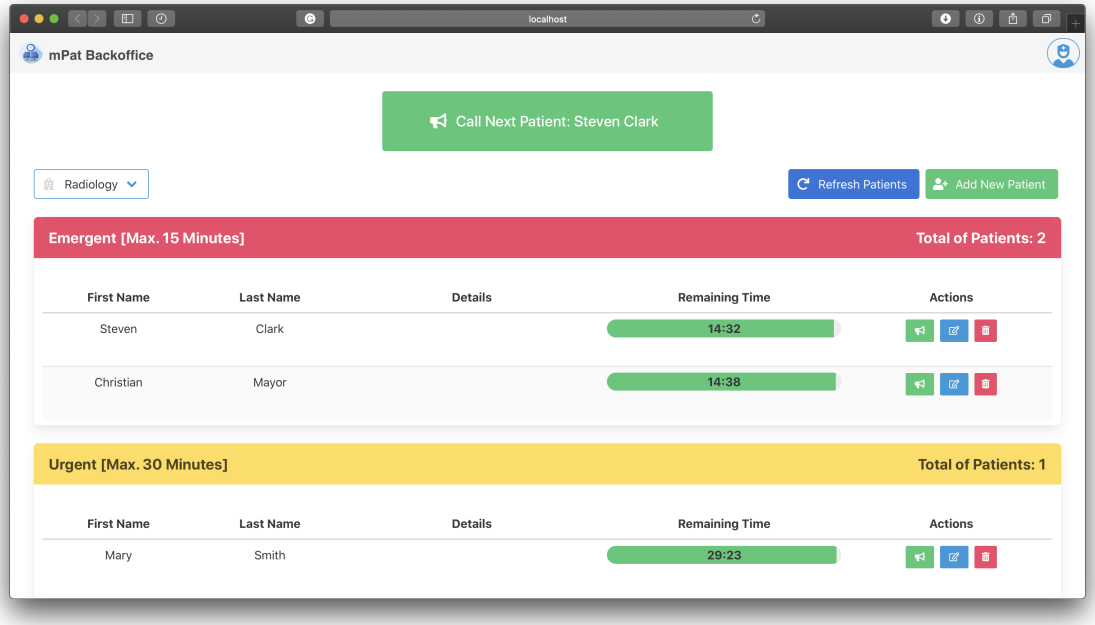

Figure 3: The homepage of mPat back-office

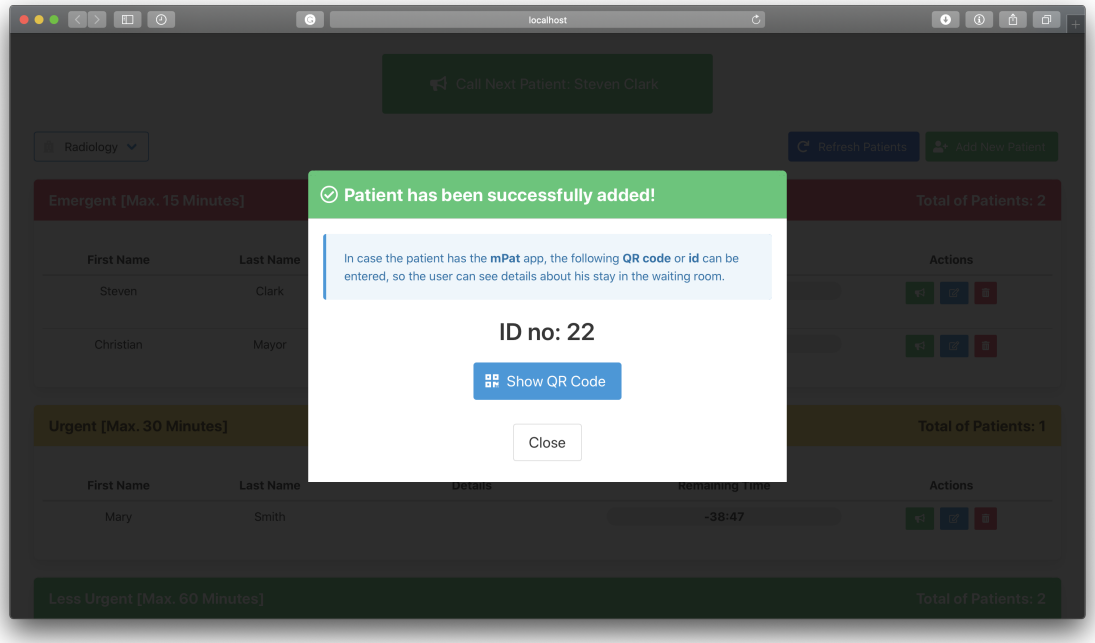

Figure 4: Dialog box after adding a new patient

Figure 5: QR code in a separate browser window

Every patient priority is depicted as a separate table, which contains all patients assigned to this priority. An additional table is shown at the end of the page that includes WIP patients. For each patient, three actions are offered: call, edit, or delete. While edit and delete require no further explanation, call offers the possibility to ignore the recommendation of the algorithm and triggers a manual call for the selected patient. Every row in a priority table displays the remaining time a patient can wait before the patient should be treated. This way the operator has a good overview of all the patients in the waiting room of a department. The user has furthermore the ability to switch between the different departments of the organisation. After each switch, the patient data of the department is immediately fetched from the REST API and the view is updated.

To add a new patient, the operator is first presented with a form, that requires the insertion of all necessary patient data. After submission, the back-office confirms the creation by providing the dialog box from figure 4 with the patient's ID and the option to generate a QR code representing this ID. The QR code will be opened by the frontend in a separate browser window as in figure 5, so it can be displayed on a separate monitor, which the patient can see and scan using the mobile device.

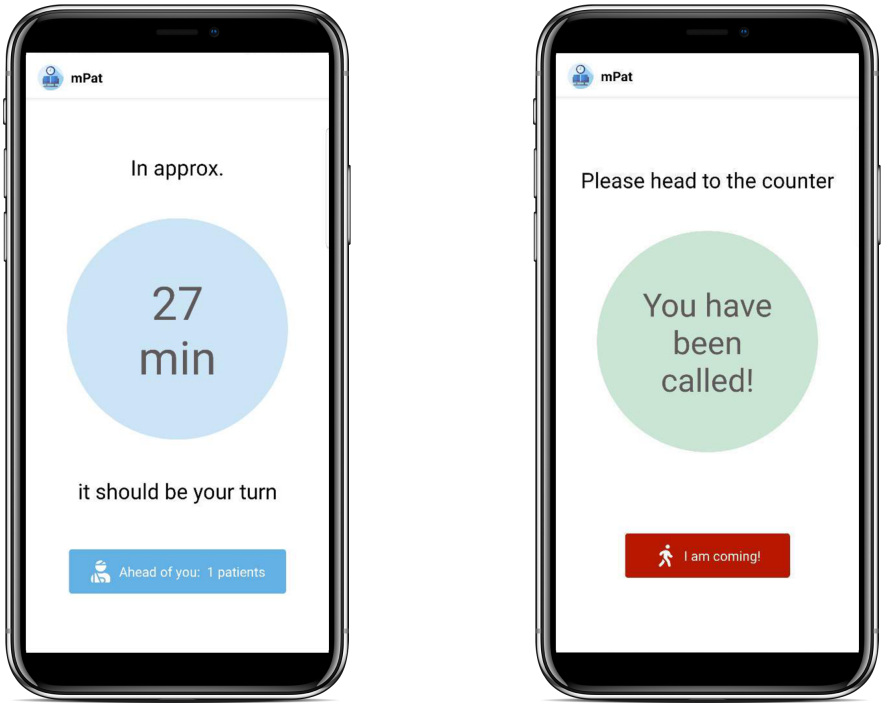

Figure 6: The mPat mobile application

3.3.3 Mobile Application

The mobile client is a React Native application written entirely in JavaScript. Compared to the frontend, the mPat mobile application does not consist of just a single physical screen, but actually utilizes native functionalities of the operating system as means of navigation. It uses a so-called *Stack Navigator* which manages the different screens of the app, by ordering them in a stack with the current view as the top element. Whenever a new screen is invoked, it is pushed on the top of the stack. By going back, the currently active screen is popped from the stack, leaving the top position again for the previous screen.

When initially opening the app, the patient is greeted with a message stating no connection to any organisation has been set up. The patient can then choose to set up a new connection whenever necessary. As stated in the functional requirements, the app offers two possibilities, either a manual registration by typing the given identification number or scanning the QR code displayed at the reception. By using the obtained

ID, the app can fetch the patient data from the REST API. Since no confidential information is being returned by the backend, there is no need for any authentication or other security mechanisms. The JSON payload contains only the estimated waiting time, the position in the patient queue, and whether the patient has been called or not, which after a successful fetch will be presented in the screen shown left in figure 6. The mobile client polls the backend on a regular interval to verify whether the patient has been called through the back-office or not. In case the calling of the patient has occurred, the application immediately switches to the screen shown on the right side of figure 6.

The mPat mobile application was built and tested on both Android and iOS and therefore supports the devices of most mobile users.

3.4 Patient Prioritizing Algorithm

The patient prioritizing algorithm has been already mentioned a few times throughout this thesis. In this section, the algorithm will be explained in further detail.

Patient scheduling algorithms categorize patients in an emergency department commonly in two groups: *new patients* or *triage patients*, which are yet to be examined as well as *work-in-process (WIP) patients*, which have been examined before and are waiting for further treatment [19]. For both groups, the aim is different [20]:

- New patients need to be treated within a given deadline, which depends on the severity of the patient's medical condition.

- WIP patients cause congestion costs since they overcrowd the waiting room and have to be treated to reduce these costs.

The principle aim of a patient scheduling algorithm should therefore be to find a balance between choosing WIP patients, to reduce the costs due to the accumulation of patients while meeting the deadline for the new patients.

Severity	Resuscitation	Emergent	Urgent	Less urgent	Non-urgent
Deadlines	Immediate	15 mins	30 mins	60 mins	120 mins

Figure 7: Deadlines according to CTAS [20]

To identify a deadline for a new patient, emergency departments often categorize medical conditions into priority classes [20]. The priority classification employed by the algorithm in this thesis is based on the Canadian Emergency Department Triage & Acuity Scale (CTAS). The different priorities of this scale are depicted in figure 7. A special priority is hereby the *Resuscitation* severity. Since these patients have to be treated immediately without any further delay [20], the *Resuscitation* priority will not be taken into account by the mPat system, since tracking the waiting time is not applicable here. However, the other severities as well as their corresponding deadlines are supported and implemented in the mPat system.

The basic data structures of the algorithm, in which patients are managed, are *queues*. Queues are generally employed for scheduling problems, for instance, in operating systems [21]: scheduling of processes can be managed in *priority queues*, where each process of the operating system is assigned to its corresponding queue based on the importance of the process. The priority queues are then processed in the order of their respective priority.

The mPat scheduling algorithm uses a similar approach. Patients are divided into three main queues as visualized in figure 8. New patients with a nearly expired deadline are assigned to the highest priority queue. The next queue contains all WIP patients, while the last queue consists of all remaining new patients, where the remaining waiting time is not critical yet. Both queues containing new patients are ordered by the remaining time until the deadline is reached in ascending order. As a consequence, the patient who is closest to a deadline expiration is at the first position in the queue. In the case of the WIP patients queue, patients are ordered by their time of registration in the mPat system.

The selection of the next patient is executed as follows: First, the highest priority queue is checked whether it contains any patients. If a patient exists in that queue, it is immediately selected and dequeued. If not, the algorithm will repeat the same procedure for the WIP patients queue and the new patients lower priority queue as well.

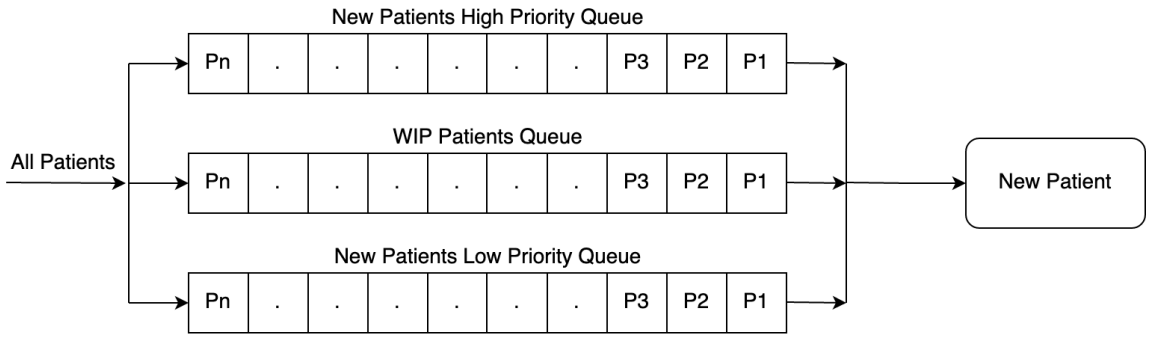

Figure 8: Priority Queues of the scheduling algorithm

The question now arises how new patients are divided into a higher and lower priority queue. For this mechanism, the mPat system offers a configuration for each department called the *WIP threshold*. It defines a numeric bound in minutes, which the algorithm uses to divide the new patients into the two queues. If the allowed remaining waiting time of a new patient is less than the configured WIP threshold then the patient will be assigned to the higher priority queue. If the opposite case is true, the patient is inserted into the lower priority queue.

Since the mPat frontend offers real-time information about the next patient, the algorithm has to be executed periodically. In case the remaining time of a new patient in the lower priority queue becomes less than the WIP threshold, the patient has to be moved to the higher priority queue and will be favoured over a WIP patient. Hence, the frontend executes the algorithm each second to immediately change the recommendation of the next patient, whenever a movement between the queue occurs.

By structuring the patients in queues, the patient position shown in the mobile application can be trivially computed. By concatenating the three queues together, the patient position is equal to the index of the patient inside the concatenation.

Through this approach, the mPat system tries to fulfill the principal aim of a patient scheduling algorithm. It tries to satisfy the deadlines for each new patient while balancing that with the selection of WIP patients, where the frequency can be configured as necessary.

4 Discussion

After the extensive elaboration of the implementation, this section provides a discussion of the mPat system. It highlights the strengths of the system as well as its weaknesses. Furthermore, it offers possibilities for future work and research.

4.1 Strengths

The primary strength of mPat is its fulfillment of the original motivation. mPat was created for patients to give them insight into their waiting time and thus improve the satisfaction of patients visiting health institutes. While definitely not complete, mPat already provides an improvement of the emergency department patient waiting process, through its management of data and informative visualization by the mobile application.

Another strength of mPat is its flexibility to host and its platform support. The back-office is operated through a web browser so any hospital registration with access to a browser will be able to use it. Furthermore, the mobile client supports both Android and iOS and can therefore be installed on the mobile devices of most patients. Lastly, the REST API can be flexibly hosted on all major operating systems as well as internally or externally depending on the requirements of the medical center.

An additional advantage of the mPat backend is its extensibility. Since the backend is a decoupled REST API, it is not limited for usage by the frontend and mobile client exclusively. Instead, any system could integrate the capabilities of the mPat REST API as necessary. The opposite is true as well: in case other systems offer an API to connect to, the mPat backend could be extended to communicate with these systems

and provide further functionality and information.

Lastly, the configuration of the algorithm through the WIP threshold provides the possibility to adapt to different scenarios. The relation of new patients to WIP patients will differ each day or hour. Through the WIP threshold, an emergency department will be able to adjust the algorithm to high as well as low levels of congestion inside the waiting room.

4.2 Weaknesses and Future Work

While every work offers a number of strengths, it suffers at the same time of a few weaknesses. In this section, these weaknesses of mPat are depicted, including future enhancements to the system to mitigate these respective weaknesses.

In the current form, it is only possible for a single user to access an mPat organisation at a point in time. In the case of another user logging into the system, the first will be automatically logged out. While this could provide some security benefits, it might not be applicable to all health institute organisations. Some organisations will require multiple people to be able to log in at the same time. The obvious solution for this problem would be to add a new table to the database of mPat containing all accounts with a reference to their corresponding organisation.

Furthermore, the current mPat system allows all login credentials to manage the departments of the whole organisation. This approach has two downsides: first of all, some organisations might want to restrict some employees to only see the relevant data of their department. Secondly, in the case of a leak of credentials, a criminal would have the possibility to get all the data of the whole organisation as well. Combined with the formerly mentioned feature of multiple accounts per organisation, a rudimentary rights management system could be implemented which allows an organisation to restrict accounts to only access certain departments.

Another future prospective concerning the mPat back-office could be configurable priorities and deadlines. Although the mPat system ships with the priorities and deadlines

of CTAS, not all medical centers will have the same system in use. The ability to configure priorities and deadlines per organisation or even department will enable mPat to be flexibly applied in additional scenarios which it currently does not support.

An additional improvement to the security of mPat would be the encryption of patient data in the database. Among the patient data are the first and last name, as well as arbitrary details which a department might store for further information concerning the treatment of the patient. In case an attacker accomplishes to access the database of the mPat REST API, all this information would currently be leaked. The problem that has to be solved when encrypting data is the secure storage of the encryption and decryption key. A common approach when employing encryption is to ask the user for the key [22]. The mPat system could therefore require the operator of the back-office to enter an additional passphrase after login, which serves as the key to encrypt and decrypt the organisation's data.

Similar to the back-office and backend, the mobile application offers some further prospective enhancements as well. Firstly, whenever a patient connects the app to the backend via the given ID of the registration, this ID is currently not stored on any persistent storage offered by the mobile operating system. As a consequence, the patient might have to reconnect to the backend after an accidental complete closing of the mPat application, where the application memory is erased or when restarting the mobile device. React Native offers the *AsyncStorage*-API to persistently store data on a mobile device [12]. The mobile application could be extended to store the ID of the patient using this API in order to be able to load the ID from the persistent storage, whenever necessary. If a patient is called and the mobile client receives this information, it could delete the ID from the persistent storage again to let the patient connect with a new ID on a next visit.

To improve the user experience of the mobile application, a notification system would be highly beneficial. There are endless situations where notification might provide the patient with even more information regarding the waiting time. Obviously, with a notification system, the user must be notified when being called to be treated next. Moreover, a notification could be sent to the user whenever a certain time threshold has been reached, for instance, five or three minutes are remaining. The same principle could be applied to the position of the patient in the queue. The patient could receive a notification if the position of the user has increased or decreased.

Another possible feature of the mobile client would be the ability of a patient to cancel the visit directly through the application. Currently, a patient has no way of canceling the visit other than getting in touch with the registration. In the case of a patient noticing that no treatment is necessary anymore, the app could offer the cancellation of the visit which would then directly be reflected in the back-office and for the algorithm.

A weak point of this work is the assumption that every patient will utilize a smart device. This will certainly be not the case for all patients. The mPat system must therefore provide functionalities to support patients without a mobile device at hand. This could include an integration of a traditional waiting time system, such as receiving the waiting number on a piece of paper along with displaying the next patient on a monitor in the waiting room.

Furthermore, during the course of this thesis, the mPat application was never tested in real-life emergency departments. It is therefore hard to judge how the patient selection algorithm will perform in stressful scenarios, as well as how different users will be able to interact with the software implemented in this thesis over longer periods of time. Moreover, a real-world test will result in revealing more essential weaknesses and important features, which could be added to the platform. A very important future experiment is therefore a trial of mPat in an emergency department.

5 Conclusion

The main goal of this thesis was to provide a new approach for keeping patients informed about their current status when waiting in an emergency department. Since mobile devices are commonly spread in today's world, they provide an obvious choice to act as a medium to receive and visualize this information. The mPat platform offers such a system that tries to be both modern as well as easy to use. During the course of this thesis, the general layout and functionality, as well as an algorithm for the selection of patients, was implemented which fulfills the general requirements of the motivation of this thesis.

Additionally, this work should also serve as an example for the application of modern and contemporary technologies in the medical section. All tools, technologies, and frameworks chosen for the implementation of mPat have been developed and released in recent years. Especially in hospitals, the usage of traditional and older technologies is widely spread [23] which can result in negative consequences (see [24]). mPat, therefore, serves as an example for a platform, built upon modern technologies and transfers frameworks or architectural styles to the clinical section, which are established in software systems of other fields of work.

Although the current implementation already offers the fundamental functionality of a patient waiting time system, it is still far from being ready for production usage. The momentary form of the application requires further research and development to mature. Especially, an extensive empirical study on the real-world usage of mPat and its effects on the satisfaction of patients must be conducted, and further features resulting from this study, as well as the ones that were mentioned previously, should be implemented. Moreover, the integration of the platform in existing health institute systems, instead of being a stand-alone software, would be highly beneficial to be more practicably suitable in medical centers.

Bibliography

[1] G. Gigerenzer, K. Schlegel-Matthies, and G. G. Wagner. *Digitale Welt und Gesundheit: eHealth und mHealth-Chancen und Risiken der Digitalisierung im Gesundheitsbereich*. SVRV, Sachverständigenrat für Verbraucherfragen, 2016.

[2] W. H. Organization. *Global diffusion of eHealth: making universal health coverage achievable: report of the third global survey on eHealth*. World Health Organization, 2017.

[3] P. Sreekala, D. Arpita, and E. M. Varghese. Patient waiting time in emergency department. *International Journal of Scientific Research Publications, 5 (5): 1* 3, 2015.

[4] A. M. Soares and M. Farhangmehr. Understanding patient satisfaction in a hospital emergency department. *International Review on Public and Nonprofit Marketing* 12(1), 1–15, 2015.

[5] D. Georgiev. 67+ Revealing Smartphone Statistics for 2020. URL: `https://techjury.net/blog/smartphone-usage-statistics/` (visited on 09/24/2020).

[6] P. Gill, K. Stewart, E. Treasure, and B. Chadwick. Methods of data collection in qualitative research: interviews and focus groups. *British dental journal* 204(6), 291–295, 2008.

[7] R. Adams. *SQL: eine Einführung mit vertiefenden Exkursen*. Carl Hanser Verlag GmbH Co KG, 2012.

[8] M. Matthews, J. Cole, and J. D. Gradecki. *MySQL and Java developer's guide*. John Wiley & Sons, 2003.

[9] A. Freeman. *Pro ASP.NET Core MVC*. Apress, 2016.

[10] C. Gackenheimer. *Introduction to React*. Apress, 2015.

[11] C. de Sousa Antonio. *Pro React*. Springer, 2015.

[12] E. Behrends. *React Native*. O'Reilly Verlag, 2018.

[13] A. Del Sole. *Visual Studio Code Distilled: Evolved Code Editing for Windows, MacOS, and Linux*. Apress, 2018.

[14] P. Inc. Postman official website. URL: `https://www.postman.com/` (visited on 09/14/2020).

[15] S. P. Developers. Sequel Pro official website. URL: `https://www.sequelpro.com/` (visited on 09/14/2020).

[16] R. Garcia, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido. A compact FPGA-based processor for the Secure Hash Algorithm SHA-256. *Computers & Electrical Engineering* 40(1), 194–202, 2014.

[17] F. Doglio. *Pro REST API Development with Node.js*. Apress, 2015.

[18] M. A. Jadhav, B. R. Sawant, and A. Deshmukh. Single page application using angularjs. *International Journal of Computer Science and Information Technologies* 6(3), 2876–2879, 2015.

[19] J. Huang, B. Carmeli, and A. Mandelbaum. Control of patient flow in emergency departments, or multiclass queues with deadlines and feedback. *Operations Research* 63(4), 892–908, 2015.

[20] J. Huang. "Patient flow management in emergency departments". PhD thesis. PhD thesis, National University of Singapore (NUS), 2013.

[21] C. Baun. *Operating Systems/Betriebssysteme: Bilingual Edition: English – German*. Morgan Kaufmann, 2020.

[22] C. Ellison, C. Hall, R. Milbert, and B. Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems* 16(4), 311–318, 2000.

[23] M. Korpela. From legacy systems via client/server to web browser technology in hospital informatics in Finland. *Studies in Health Technology and Informatics* (1), 222–226, 1998.

[24] P.-J. Maenhaut, H. Moens, V. Ongenae, and F. De Turck. Migrating legacy software to the cloud: approach and verification by means of two medical software use cases. *Software: Practice and Experience* 46(1), 31–54, 2016.