Daniil Smirnov

PHARMACY AGGREGATOR WEB APPLICATION

Bachelor's thesis
Information Technology

2019

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Daniil Smirnov	Bachelor of Engineering	December 2019
Thesis title Pharmacy Aggregator Web Application		57 pages
Commissioned by -		
Supervisor Timo Mynttinen		

Abstract

The goal of this thesis was to study the technology stack required for the web application development. This includes Java EE, React TypeScript and all the side frameworks. Another goal was to create a functional web application based on the knowledge gathered during the first part. The application had to utilize both the backend and frontend approaches. This approach of having two goals has defined the section structure of the thesis. There are two sections which describe both: the study made and the practical appliance of the knowledge.

The theory part has a detailed description of all the technologies used for the server-side application and the UI part. Furthermore, all additional frameworks and tools like Keycloak and Wildfly are covered by this section as well. On the other hand, the implementation part contains a step by step process of building a web application called Pharmacy Aggregator. It is a tool made for medicines search. Moreover, it has features like medicine ordering implemented.

The result of this work was a functional web application which became a logical outcome for the topics studied during the thesis. The Pharmacy Aggregator application followed the decisions made during the theory part and went through a development process in an implementation part.

Keywords

Java, React, Typescript, Web application

CONTENTS

1 INTRODUCTION 5

2 THEORY PART 6

2.1 Introduction to Java development 6

2.1.1 Maven 8

2.1.2 Wildfly 11

2.1.3 Keycloak 15

2.2 Frontend 17

2.2.1 React 19

2.2.2 TypeScript 22

2.3 Databases 23

2.4 Additional tools 25

3 IMPLEMENTATION PART 26

3.1 Product planning 27

3.1.1 Backend development environment 28

3.1.2 Frontend development environment 30

3.2 Server development 31

3.2.1 RESTful API 31

3.2.2 Adding database support 38

3.2.3 Implementing authorization 40

3.3 Web application development 42

3.3.1 UI Planning 43

3.3.2 UI Implementation 43

3.3.3 Styling 48

3.4 Finalized version demonstration 49

4 CONCLUSIONS 51

REFERENCES 53

1 INTRODUCTION

Software development has always provided flexible and scalable solutions for various businesses. These solutions may be websites, mobile applications, web applications and many other implementations. While mobile applications are available only on a limited number of devices and platforms, web applications allow for great device diversity. They may be accessed using any portable or stationary device that has a web-browser of any sort. Moreover, web applications provide the same full range of abilities as mobile applications do.

Most of the web services provided by businesses or non-commercial organisations nowadays are web applications. This approach allows clients to get access to services no matter what device they use for that purpose. The result is not only a significant productivity increase, but also the ease of service use for customers.

Nevertheless, many social organisations neither have enough resources to develop high-quality online services, nor understand their necessity. This may even affect the areas of paramount importance, such as the medical sector. Many people have to follow the complicated routine to get their medicines. This and many other related problems may be eliminated by implementing a simple online solution.

A web application I decided to create for that purpose is a secure, yet a scalable solution allowing its customers to significantly simplify the whole process of getting medicines they need. The system will recognise customers based on their personal approved credentials providing them with all the necessary information on the requested medicines. The application will present the list of pharmacies where medicines are currently available to simplify customers' life. One of the possible features is the ability to book certain medications straight in the web application's interface.

My theoretical part includes all the research outcomes on software development used for achieving the thesis goal. It also contains theory and explanations for the

server-side development using Java and the actual web application development using React. Furthermore, detailed information on all the side frameworks and theoretical knowledge that is required for task completion are also included there.

In order to accomplish my goals I use the following methods:

- Research, gathering skills and implementation for the backend application server with databases support written in Java: these databases must contain pharmacies' data by the initial plan.
- Research and implementation of a web application written in React TypeScript: this includes creating UI and implementing styles for it.
- Exploring possible authorization mechanisms and integrating them into the application.

The final part consists of the product functional demonstration and evaluation of the work done.

2 THEORY PART

The theory grounds of this work cover all the theoretical knowledge required for achieving the project's goal. A detailed review begins with Java backend development. In addition, the following chapters will describe the process of running and maintaining a backend Java application, side software and the configuration for each application's segment. Then, frontend programming principles are studied on an example of React development. The concepts of a programming language such as structure, lifecycle and fundamental features are explained. Furthermore, security features Provided by Keycloak are introduced in detail. And last but not least, there is a database structure review and studying the MariaDB database management system.

2.1 Introduction to Java development

Java brings its roots from the year 1991 when the development process began. James Gosling was working for Sun Microsystems company at the time. During the first development iterations, Java had a completely different name – Oak. The initial purpose of the language was an interactive television appliance. However,

it changed soon after the development process started. Java was designed with C-like languages which were highly popular at that time. (Liang 2010, 8.) That significantly helps developers nowadays, as the syntax similarity of languages makes switching from C language family to Java or vice versa as painless as possible. Nevertheless, all the similarities occur only at a superficial level. Although C/C++ languages may offer higher code processing speeds, Java wins the competition with high security and overall high balanced performance. (Niemeyer 2000, 7.) Sun Microsystems was acquired by Oracle. Therefore, Java has been supported and further developed under Oracle's jurisdiction.

Java is an object-oriented multiple purpose programming language (Liang 2010, 8). Developed almost three decades ago, Java has retained its key principles until this day. The most important one is WORA which stands for "Write Once Run Anywhere". That is true since java applications are supported across most of the modern platforms. (Langley 2002.) Nevertheless, Java significantly differs from other programming languages. Instead of translating its code straight to machine code Java uses a Java Virtual Machine (JVM) in the middle. It takes a bytecode received from a compiler as an input. Bytecode is processed by the JVM and translated into the machine code specific for the target system. That brings not only a great supporting platform variety, but also makes programs written in Java to perform the same way across all those platforms. Looking at Figure 1 below which is based on Niemeyer (*2000, 4-6)*, the platform-independent source code is easily compilable on any platform and operating system. JVM is responsible for detecting the platform and providing the right machine code. (Niemeyer 2000, 4.)

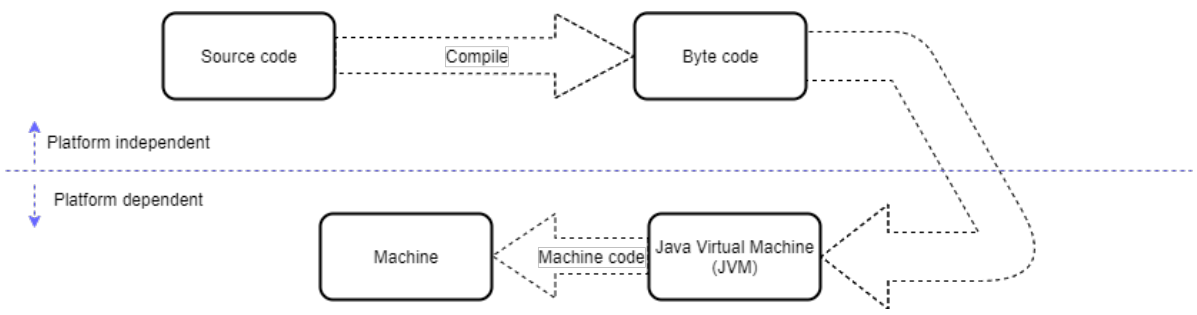


Figure 1. Java code execution steps.

The most common Java use cases often relate to mobile software development. Nevertheless, Java is also a great choice for building a backend skeleton for web applications. Therefore, it was chosen as the most efficient way of developing a web applications server for this thesis work.

2.1.1 Maven

Standard Java Development Kit (JDK) always contains a built-in compiler. This tool takes any .java file and compiles it to the executable .class file format. (Liang 2010, 13-14.) It has been a simple and reliable way of running Java applications for many years. Nevertheless, JDK compiler is only suitable for small Java projects containing only a few .java files. As the programs grow bigger and span many classes, the complexity of this compiling solution rises well above reasonable. That is why, there are multiple third-party open-source solutions to address this problem.

Maven is one of them. This framework’s name takes its roots from the Yiddish language where "meyvn" means to be an expert (Apache Software Foundation 2019a). Furthermore, this name precisely illustrates the actual capabilities of the framework. In other words, it is a tool that not only builds a Java project considering all its dependencies but also compiles it and prepares to be executed. The framework significantly simplifies and completely automates the program deployment process. (Apache Software Foundation 2019a.) Furthermore, the bigger the project, the more efficient compiling becomes in comparison with the Java Development Kit default compiler. Thus, Mavens capabilities get far beyond the ones offered by the standard JDK.

Besides, there are other noteworthy frameworks on the market that perform similarly to Maven. One of them is called Ant which is an acronym for Another Neat Tool (Apache Software Foundation 2019c). It has been developed under the Jakarta project that was initiated to support open-source software development. Therefore, Ant has become the first Java project management tool in Jakarta's bundle. (Conor 2005.) It is capable of automatically building java projects and getting all set up code from Extensible Markup Language (.xml) configuration

files. Setup code contained in these files is executed sequentially, based on imperative programming principles. This approach is found to be overcomplicated, as the developer has to include all the "Targets" for project building in the correct order. (JavaTpoint 2018.) As a result, Apache company has developed another solution called Maven after four years since Ant release.

Compared to its predecessor, the new framework has a different structure. It deploys a project based on declarative programming principles. They are completely opposite to imperative principles, as configuration files contain, not the hardcoded deployment instructions, but project specification and description. (JavaTpoint 2018.) In addition, all the file related tasks that Maven gets from the specification are processed using either internal or external plugins. To sum up, Maven is considered a more modern and easy to use alternative to Ant, mostly for improved lifecycle. Tasks can be easily added to any part of the specification and run using one of the plugins that are widely available nowadays. (Javin 2017.) Therefore, Maven has become the best suitable project building tool for this thesis work.

The plugin based architecture of Maven allows the highest possible flexibility and performance during Java application deployment. Moreover, plugins do not need to be installed locally. This has become possible due to Maven’s specific architecture providing direct access to plugins’ input and output. That is why, the plugin pool is so versatile and still keeps growing rapidly. (Apache Software Foundation 2019d.) Any developer may write his solution to an existing issue and share it with the entire community. Setup specifications and dependencies are written to the pom.xml file and managed based on multiple key principles including the following:

- Repositories are libraries of files Maven makes references to while scanning through the setup file. If a dependency could not be resolved locally, the framework will connect to the distant repository stated in the setup and copy it as the local one.
- Transitive dependency is a dependency chain obtained from a setup document and the closest dependency is used to achieve better efficiency.

- Dependency search is carried out using special coordinates: “groupId”, “artifactId” and version. Coordinates can be found from dedicated search engines like Maven Search Engine.
- Dependency exclusion refers to a project description file which contains a feature of excluding dependencies that are either not required in a current build or create certain problems for the project.
- Repository managers mean that repositories may be found and downloaded from special Maven Repository Manager which performs a search through all known archives to find a required repository.

This dependency type list is based on the Apache Software Foundation (2019e) article. Standard Maven directory layout is illustrated below in Figure 2.

`src/main/java`	Application/Library sources
`src/main/resources`	Application/Library resources
`src/main/filters`	Resource filter files
`src/main/webapp`	Web application sources
`src/test/java`	Test sources
`src/test/resources`	Test resources
`src/test/filters`	Test resource filter files
`src/it`	Integration Tests (primarily for plugins)
`src/assembly`	Assembly descriptors
`src/site`	Site
`LICENSE.txt`	Project's license
`NOTICE.txt`	Notices and attributions required by libraries that the project depends on
`README.txt`	Project's readme

Figure 2. Maven directory layout (Apache Software Foundation 2018)

It clearly shows the hierarchy of all the standard subdirectories that will be reviewed in detail during the implementation part in Chapter 3. Furthermore, Figure 3 provides a hierarchy tree visualization of a Maven project.

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Figure 3. Standard project structure (Apache Software Foundation 2019b)

The setup pom.xml file is located straight under the head directory. Besides, there are two subdirectories of a source folder. First, (src/main/java) contains the source code of a project. On the other hand, (src/test/java) has the test resources that are used for various testing appliances in the project. (Apache Software Foundation 2019b.)

2.1.2 Wildfly

The application server is an essential platform that makes all the software components work in conjunction. The program mounted on top of the server becomes completely functional. It gets all of its dependencies in one place. Furthermore, features like storage, network properties, plugins and security are configured all in one place. Therefore, not only an application gets a significant performance boost, but also developers save their time utilizing application servers. (Red Hat 2017.)

Wildfly is a powerful application server certified for Java Enterprise Edition 8. Developed by JBoss initially, it is now distributed by Red Hat Inc which bought JBoss in 2006. It is a free open-source software. However, clients will have to pay for the support. One of the key advantages of the software is its accordance with Jakarta EE principles. These are the industry-wide standards that define architecture suitable for large companies. (Red Hat 2019a.) Among other

functions, Wildfly allows a full out of the box support for java applications. It provides services that enhance development a lot such as:

- Java Database Connectivity allows effortlessly linking an existing database with the application. The relationship between two is described programmatically. (Red Hat 2017.)
- Undertow is a well-performing and lightweight web server. It is optimized for maintaining a significant number of connections at once. Outweighs it's competitors with incredibly high "throughput". (Red Hat 2017.)
- Job Scheduler is a tool allowing planned task execution. It is responsible for adding a task to the timeline and executing it according to the schedule. (Red Hat 2019b.)

Figure 4 illustrates Java application program interfaces.

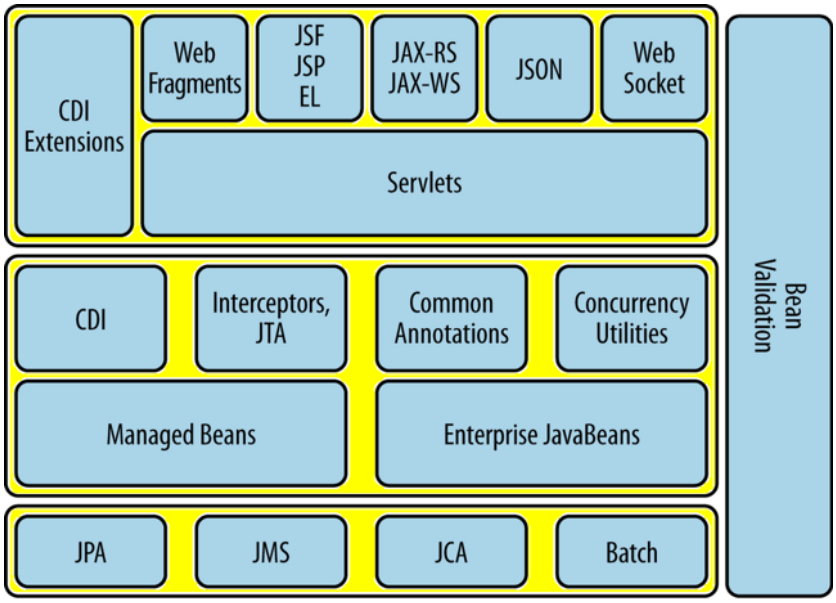


Figure 4. Java EE Api (WildFly 2019a)

These interfaces are available right after the software is deployed using a Wildfly application server.

According to the online documentation published by Red Hat (2019b), Wildfly has two main directory structures "domain" and "standalone". These two structures

provide the same server capabilities. However, the management is carried out in different formats.

The domain directory structure is a recent feature that allows running and managing multiple application server installations from one host. To control the lifecycle of Wildfly instances Host Controller interacts with Domain Controller. Besides, the configuration files are found all in one place. (WildFly 2019b.) The structure of folders is described in Table 1 that is taken from official Red Hat (2014) documentation.

Table 1. Domain directory structure (Red Hat 2014)

DIRECTORY	DESCRIPTION
configuration	Configuration files for the domain and the Host Controller and any servers running off of this installation. All configuration information for the servers managed within the domain is located here and is the single place for configuration information.
content	an internal working area for the Host Controller that controls this installation. This is where it internally stores deployment content. This directory is not meant to be manipulated by end-users. Note that "*domain*" mode does not support deploying a content based on scanning a file system.
lib/ext	Location for installed library jars referenced by applications using the Extension-List mechanism
log	A location where the Host Controller process writes its logs. The Process Controller, a small lightweight process that spawns the other Host Controller process and any Application Server processes also write a log here.
servers	Writable area used by each Application Server instance that runs from this installation. Each Application Server instance will have its own subdirectory, created when the server is first started. In each server's subdirectory there will be the following subdirectories: data -- information written by the server that needs to survive a restart of the server log -- the server's log files tmp -- location for temporary files written by the server
tmp	location for temporary files written by the server
tmp/auth	Special location used to exchange authentication tokens with local clients so they can confirm that they are local to the running AS process.

All the individual servers are found under the "servers" directory. They are combined and run under the Host controller guidance that is mainly located under the "content" directory.

The standalone directory structure has retained its functionality from the previous versions. It has been a simple, yet effective way of running an application server. The setup is simple enough for different developers' levels of expertise. In addition, the all-in-one-place approach provides not only high security, but also an impressive performance of the application. Directory structure represents a folder containing all the necessary configuration files. Furthermore, the application with its executable files is located in the same place. (WildFly 2019b.)

According to Table 2, the application files are put under the "deployments" directory for automatic initialization and execution.

Table 2. Standalone directory structure (Red Hat 2014)

DIRECTORY	DESCRIPTION
configuration	Configuration files for the standalone server that runs off of this installation. All configuration information for the running server is located here and is the single place for configuration modifications for the standalone server.
data	Persistent information was written by the server to survive a restart of the server
deployments	End-user deployment content can be placed in this directory for automatic detection and deployment of that content into the server's runtime. NOTE: The server's management API is recommended for installing deployment content. File system based deployment scanning capabilities remains for developer convenience.
lib/ext	Location for installed library jars referenced by applications using the Extension-List mechanism
log	standalone server log files
tmp	location for temporary files written by the server
tmp/auth	Special location used to exchange authentication tokens with local clients so they can confirm that they are local to the running AS process.

Other directories apart from “configuration” are used by Wildfly for temporary files storage. The standalone approach best suits for this project work. Therefore, it will be reviewed in detail later on in Chapter 3.

The configuration of Wildfly application servers is done inside standalone.xml or domain.xml – depending on a directory structure choice. However, in both cases, files have .xml extension and require the configuration code to be written with an eXtensible markup language. It is a data format specifically designed for distribution across the World Wide Web. The configuration file contains a full description of all extensions and plugins used in conjunction with the application. Links where these plugins are located must be included as well. (Red Hat 2019b.) Nevertheless, this type of configuration is easily modified and troubleshot in case of failure.

2.1.3 Keycloak

Keycloak is a free open source security solution for web and mobile platform applications. It is capable of managing user access, their privileges and tailors their access rights precisely up to the developers’ needs. The software provides a single sign-on principle. It means that the user has to sign in only once to obtain access to all the services and applications provided by the developer. (Keycloak 2019a.) This principle applies to the sign-out process as well, meaning that the user has to sign out only once to abort all of his active sessions. Keycloak allows high and simple configurability for developers. All the settings are applied using the web interface which might be accessed from the local server where the software runs. (Keycloak 2019b.) The standard configuration interface is presented in Figure 5 below.

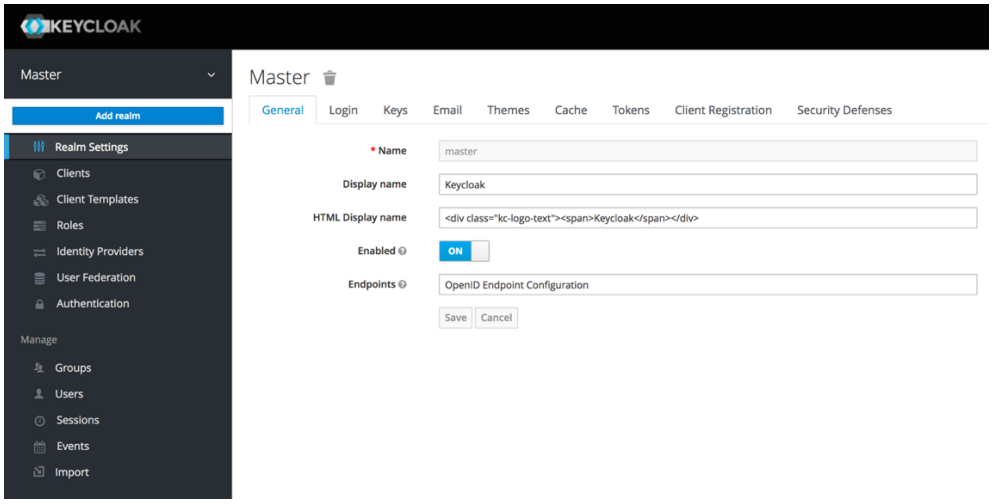


Figure 5. Keycloak web configuration interface (Harrington 2017)

The "Clients" tab is used to configure individual application settings, including different grades of user access. Not only specific users may be allowed to log in, but also their permissions could be changed very precisely.

Security protocols utilized in Keycloak are modern and highly effective. OpenID Connect protocol (or OIDC) is a significantly improved version of the OAuth 2.0 protocol. Its capabilities deliver complete authorization and authentication functionality. On the other hand, OAuth 2.0 simply provides specific authorization flow that is heavily utilized in OpenID Connect protocol. Furthermore, OIDC uses JSON Web Token standards to achieve high efficiency and supremacy over its predecessor. Therefore, tokens are distributed and processed in a JSON format. All operations are secure as the latest encryption methods are integrated into the protocol. In addition, token operations are designed to be lightweight and web-platform fully integrated. (OpenID 2019.)

According to Keycloak (2019c) web documentation, there are generally two situations when the OpenID Connect protocol is used. These situations are the following:

- The application initiates user authentication. Upon successful credentials, entry application receives two tokens. First, the identity token contains user data such as email, username and other personal credentials. At the same time, there is an access token. It provides a set of permissions

assigned to a user. Therefore, access is granted only to specific areas described in the access token.

- Clients initiate the authentication process pretending to be a user. It will receive an access token only in case a user gives his permission. A token may be used for gaining access to remote services. Nevertheless, each service request will be examined to meet all the specific requirements. If all checks are successful, access is granted.

On the other hand, there is a SAML 2.0 protocol which is an alternative to OIDC. Even though the SAML protocol is much older, it has also proven over the years to be reliable and effective. Nevertheless, when considering Java development, this protocol may cause difficulties at finding the common language. The newer software OpenID Connect works in the heart of web applications. It also provides strong communication with Java and JavaScript languages, as all the tokens are presented in proprietary JSON format. (Keycloak 2019c.) Moreover, it is officially recommended in the Keycloak (2019c) documentation to utilize the newer protocol due to enhanced functionality and better support.

2.2 Frontend

As the backend principles and technologies were covered in the previous section, it is time to dig into the frontend part. User interface (UI) is usually developed based on three main languages. They are JavaScript, HTML and CSS. HTML defines the webpage structure and content. Furthermore, it allows linking current page to other HTML sources for navigation. CSS stands for Cascading Style Sheets. They define the design of the web-page elements and all the visual enhancements. In turn, JavaScript makes everything alive bringing animations. It also includes the logic rules that make interaction with the end-users possible. (Codesido 2009.) Furthermore, this simplifies the backend part, as not all the logic has to be written there.

Nevertheless, this approach seems to be a little outdated. Multiple frontend development solutions combine all these languages in one place. Therefore, consideration of the three best tools was made. It is worth mentioning that each

of these frontend development mechanisms is based on the DOM interface. Document Object Model represents both: the HTML structure of a web-page and all the programmatic modifications made using JavaScript. (Aderinokun 2018.) It has a tree-like structure that can be seen in Figure 6.

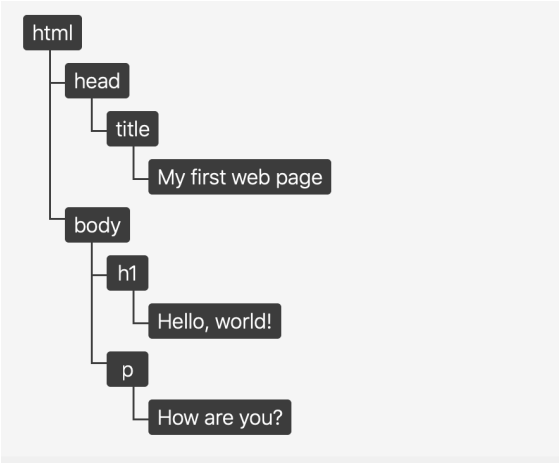


Figure 6. DOM basic structure (Aderinokun 2018)

In other words, DOM is neither the copy of an HTML document nor a direct representation of the web applications output (Aderinokun 2018). Despite that, it is an interface that brings all web development stages together. It inherits page structure from the HTML and allows it to be modified programmatically. That is exactly how developers make their web applications change state and become interactive.

Below is a description of three main frontend libraries. According to Borrelli (2019), these are the following:

- Angular/Angular.js is a framework developed and supported by Google. It uses TypeScript programming language. During web development modifies actual DOM that requires a lot of system resources and may be problematic. However, great community experience and support are provided for Angular.
- Vue.js is a JavaScript framework. It is an independent development that has gained popularity recently. Great performance and absence of significant bugs. Utilizes virtual DOM principle taken from React. It allows for cross-browser development.

- React.js combines pros from both worlds. Allows development using not only JavaScript but also TypeScript. Furthermore, virtual DOM guarantees rapid performance and high stability. Features cross-platform support.

React has recommended itself as a stable and productive developers' tool. Virtual DOM is an important feature nowadays allowing end-users to access the web applications from any browser of their preference. (Borrelli 2019.) In addition, React community has grown gigantically in the past six years. Thus, finding support is an effortless experience.

2.2.1 React

According to Facebook Inc (2019a), React is an open-source JavaScript library that is widely used to create user interfaces (UI). It was originally developed by Facebook for internal purposes. Nevertheless, the library became available to the public in 2013 during a software conference. Ever since, React has been growing and catching up with the market-leading development software. Cross-platform mobile development was made possible with the release of the React Native framework. It allows creating native platform applications for iOS, Android and Windows Phone. (Facebook Inc 2019b.) Nevertheless, the cornerstone of React is the possibility of web application development, especially the single page cross-platform programs that are supported on all current web-browsers. Single page application (SPA) describes a web application that has all the views/pages based on one HTML code. Furthermore, the user interaction is built by dynamically loading the required resources: HTML pages, Cascading Style Sheets or JavaScript appliances. (Borrelli 2019.) Therefore, the SPA is very similar to mobile applications in the way of user interaction and overall performance. The main difference being that SPA are running on top of a web-browser utilizing its resources, while mobile applications rely on device hardware. (Flanagan 2006, 497.)

The main benefit of working with React is the necessity to only learn the features required by the current task. There is no need to explore all the React or JavaScript features to develop high-end software. Also, one of the key React principles according to Facebook Inc (2019b) is "Learn once, write anywhere". In

other words, React will run in conjunction with most software solutions written in a wide diversity of programming languages.

There are some fundamental laws of working with React. First of all, the HTML document tag structure is partially retained. When a piece of data needs to be displayed, a render function is called (Facebook 2019c). Moreover, React is a component-based development library. Different elements and their functionality are represented as components. Furthermore, each of them may be called any time in the render function described above. (Kagga 2018.) An example from one of my previous projects can be seen in Figure 7.

```
45    public render() {
46      if (!this.props.field.name) {
47        return null;
48      }
49
50      return (
51        <div>
52          <input type="date" onChange={this.onDateSelect} />
53        </div>
54      )
55    }
```

Figure 7. Render function example

It is clear that the HTML tag is returned by a render function. Thus, it will be displayed in a web application. Moreover, JavaScript code may be inserted in between tags using curly braces. This is also shown in Figure 7 on line 52.

One of the most important features of React is using the Props and State approach. Props contain data constants that remain unchanged for all program classes. A value of a Props variable may be accessed from absolutely any method or component using syntax such as “this.props.variable”. Under no circumstances, Props should be changed as they are considered as pure functions which “always return the same result for the same inputs” (Facebook Inc 2019e). On the other hand, State contains the same types of data as Props. Nevertheless, it is personal for all React components and may be modified. State is heavily utilized in the development process. It usually contains functions that

support UI. Changing State components allows building a logic behind the UI actions. Therefore, making it interactive for an end-user. (Facebook Inc 2019d.)

Following Facebook Inc (2019f), all the components in React have their own lifecycle. It represents a process from the first mounting component until it is unmounted. A component can be modified in the middle of that process. React supports many lifecycle methods. Nevertheless, the following are three key ones:

- ComponentDidMount method is used for any actions to take place right after a component has been mounted. Most often these are various API requests.
- ComponentDidUpdate is called as soon as the component is updated. Good for updating DOM in parallel with State updates made inside of a component.
- ComponentWillUnmount method is referred to last moment before unmounting a component. Mostly used to exempt system resources and save memory.

Correct application of the methods listed above gives multiple advantages during the development process. (Facebook 2019f.) Code looks neat and human-readable. Lifecycle phases are shown in Figure 8.

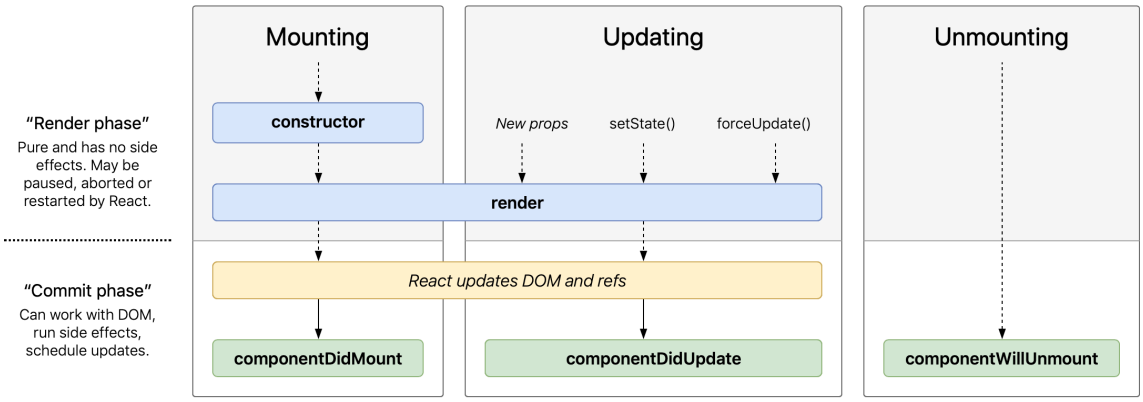


Figure 8. React lifecycle (Facebook Inc. 2017)

The features described above make React development a totally controlled process. The implementation part of this project will contain a more detailed look at these theoretical grounds.

2.2.2 TypeScript

Since the first day JavaScript has been released it was constantly growing. Complications and dependencies have been added daily. Initially, JavaScript was dedicated to client-side application development only. (Dubey 2018.)Therefore, languages complexity and bulk code never allowed programming language to become object-oriented. JavaScript remained a scripting language not only by design, but also by an implementation.

The TypeScript was developed with a goal of tackling drawbacks of JravaScript. Being an object-oriented language by design, TypeScript allowed easy serverside or backend development out of the box. Furthermore, it has kept a full JavaScript support. Figure 9 provides a visualization of TypeScript having the full support of all the current JavaScript versions.

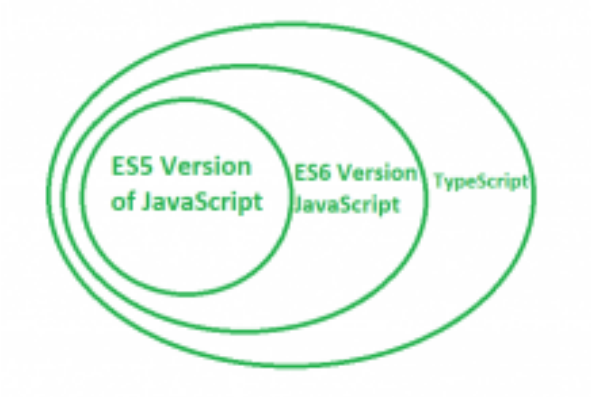


Figure 9. Typescript is a superset of JavaScript (Dubey 2018)

That approach allows developers to integrate old code snippets into a completely new TypeScript structure. Moreover, JavaScript code may be converted to a TypeScript by simply changing the file extension from .js to .ts. (Dubey 2018.)

There is a notable difference between the two languages. While JavaScrips can be executed by any platform without conversion, Typescript has to be converted to a JavaScript code first. That is the only downside of a newer language as the conversion takes marginally more time for code compilation. However, that makes TypeScript just as widely supported as it’s predecessor. It is supported

across all possible platforms, web-browsers and devices. Besides, language has many other significant advantages over JavaScript. (Microsoft Inc 2019.) Here are some of them:

- Tidy and easy to read code.
- Compilation right during the code writing process.
- Not complicated for developers to learn the object-oriented structure.
- The constant support of ESNext (latest ECMAScript features).

Furthermore, the main reason why developers choose TypeScript over JavaScript is coding simplicity that enhances both development and troubleshooting experiences. (Dubey 2018.)

React received support for TypeScript in version 3.0. Meaning that any React library project may be equally created with JavaScript, TypeScript or a mixture of both languages. It can be done with a "Create React App" tool that is used to initiate a new project in React. All operations are carried using the console commands. Ending of command will define the main programming language for the project. Thus, having a "--typescript" flag sets the language to TypeScript and generates all the necessary code snippets for an effortless start. (Ross 2019.)

2.3 Databases

In order for an application to obtain data, it has to be sorted and presented in an organized manner. Furthermore, different data sets require references to each other. There are also various types of data like text, numbers or boolean values. Databases address all of the statements above. They have the ability to store data in any imaginable pattern, linking data sets and providing specific information sections on request. This type of databases is called “relational” and is managed by a Structured Query Language (SQL). SQL is widely used in database management systems that enhance software development significantly. (Rouse 2019.) MySQL is a great example of a relational database management system. It is easily implemented on top of any software solution. Most operations are carried out via console or by sending SQL requests. Advantages like cross-platform support have granted MySQL popularity in the developer's community. (Oracle 2019.) Therefore, many database management tools have been developed on top of the system. One of the most eye-catching solutions is

MariaDB. While being just a MySQL fork it is developed by the community and has a totally free open-source approach (MariaDB Foundation 2019a).

According to MariaDB Foundation (2017), MariaDB's database engine of choice is XtraDB. The database engine is a relational database component responsible for database storage. XtraDB is a branch based on an older engine called InnoDB. Being fully compatible with its precursor, the engine has gained multiple enhancements, such as:

- Improved input/output system and memory utilization.
- Support for multithread data reading and writing.
- Throughput manual control.
- A capture data system adapted for multi-processor systems.

The upgrades described above has made the XtraDB engine well optimized for high-performance environments. (MariaDB Foundation 2017.) It is also worth mentioning that the current implementation of the InnoDB engine differs significantly in MariaDB and MySQL. Therefore, using XtraDB engine eliminates backward compatibility between two management systems. (MariaDB Foundation 2019b.)

First and foremost security measure that should not be overlooked is a password policy. Only a root user has access to all the databases operated by the engine. Thus, a password must be strong and stored in an encrypted form only. In addition, database engines like MariaDB provide SSL and SSH encrypted data transition protocols that exclude information fraud. (W3resource 2019.) Besides, additional security measures must be applied by a network administrator. Following W3resource (2019), it is clear that MariaDB uses network port 3306. Thus, it has to be locked only for database operations and restrict any other access possibilities.

In conclusion, it was decided to put the MariaDB engine in use for this thesis project. Not only it has an eight percent performance increase over its main competitor, but also the wide community support enhances the database development experience.

2.4 Additional tools

The entire software development process could be viewed as a set of blocks required to build an application. These blocks are nothing else but major development technologies and environments used in this process. However, the blocks would not stick together without the help of additional smaller tools which support a development process. The field of supporting tools covers the entire software production process. Those could be communication applications which help team members to produce better collective results. Moreover, another great example is version control tools such as GitHub. They allow teams working on the same projects, tracking previous versions and avoiding any code conflicts (Brown 2019).

On the other hand, various tools support actual development environments. One of those tools is a Swagger web-based toolkit. It allows writing RESTful APIs using their proprietary OpenAPI specification. It is a specific language format that is used for API development. Swagger is accessed via a web browser, therefore, it is easy to access for developers. The API editor not only provides live error detection, but also allows a developer to test all his generated API endpoints on the same web-page. In addition, freshly written Application Programming Interface can be transformed to the desired programming language code by a Swagger Codegen. It is a tool that creates a server-side or client-side code skeleton. Furthermore, Swagger Codegen creates all the necessary classes for the API endpoints that the developer specifies. (SmartBear Software 2019b.) All these functions significantly simplify the RESTful API server development as writing generated classes manually could be exceedingly time-consuming.

Another small tool that was put into a big use is called NPM. It is an open-source CLI tool that runs JavaScript or TypeScript programs. The main advantage is that NPM has a built-in lightweight application server that automatically deploys an application and supports all its services without any additional setup. The application may be accessed via the web-browser by following a “localhost:3000” address. In case more applications are deployed, their addresses will differ only with the port numbers. NPM has a large community that puts a lot of effort into its

support. Furthermore, documentation found from the official NPM's website provides answers to all the possible related questions. (NPMJS 2019.)

Last but not least is Homebrew. It is a CLI package manager with a native Mac OS support. It allows searching for any possible software and downloading it for free. The repository approach is used in this tool. Homebrew is absolutely legal to use. Furthermore, it has an outstanding performance as any program is downloaded with only a single CLI command. Not only software is downloaded with Homebrew, but also it is installed and prepared for utilization without additional actions required from the user. Besides, the previous two supporting tools described in this section were downloaded using the Homebrew tool.

3 IMPLEMENTATION PART

The implementation part of this project started with only a marginal knowledge concerning the technologies described in the theoretical part. Therefore, a lot of things were studied alongside the working progress. Even theoretical fluency in most of the following topics did not help sufficiently in a real theory to practice transition. Nevertheless, drawbacks were resolved successfully during the implementation. Furthermore, confidence has raised gradually during the development process. It has allowed gaining assurance in fields of full-stack application development that are described both in thw theory and implementation parts of this thesis.

The following chapters describe a detailed process of Pharmacy Aggregator application development. Description begins with designing an initial concept. All the assumptions and technologies of the conceptual application may evolve as the progress flows. Following part describes the development process for each application's section. Furthermore, this part includes the full author's experience of development such as drawbacks, development environment imperfections and solutions to different problems. The following section contains bringing pieces of software together and attuning the relations between them. Last but not least, the comparison between expected and actual outcomes is made at the end of the

implementation part. It is supplemented by the evaluation of the progress made and final solution images.

3.1 Product planning

The initial idea of a project was formed during practical training at the Metatavu company. One of their projects was related to the Finnish healthcare system. It is a wide fild which is full of open development gaps. Being one of the most important things nowadays, healthcare system has many complicated processes, which confuses people a lot. This issue also affects pharmacies. The usual routine of acquiring medicines is overcomplicated as people not only have to visit a doctor to obtain prescription, but also have to find their prescribed medications somewhere. The closest pharmacy does not necessarily have what they need. Therefore, customers have to either go to another one, or wait for the delivery of the medicines.

The Pharmacy Aggregator application was planned as a tool to tackle the problem described above. Not only would it make people get their medication on time, but also benefit pharmacies' income. The main idea behind the application is that users are presented with one screen view. It contains a search field where the needed medication's name should be entered. Afterwards, the software displays a table containing availability data for the desired product in the customer's local pharmacies. That only feature should give a clear understanding of where the medicine can be picked up and at what time. Furthermore, in some countries where medication is sold without prescriptions, this application could even save someone's life by timely guiding them to the right pharmacy. Apart from the search field, the initial screen was decided to have a login bar on top. That displays a current user logged on to the application. Furthermore, the bar contains a request button. Pressing this button opens a clear form that is used to request any prescribed medicines in case they are not available in the pharmacies nearby. The form also contains personal user credentials, such as identification number and home address.

By the initial idea the pharmacies' data should be stored in a database connected to the server application. That information is then accessed by the UI application. The identification mechanisms provided by Keycloak allow configuring various access levels. This eliminates the possibility of any unwanted sensitive data being accessed by a third party. Furthermore, having different account types allows having pharmacy accounts that will update medicine availability lists. However, the administration features were not planned to be included, as this application is only a prototype.

The application was introduced as a sketch with all the main functionality described. Therefore, the next steps in the work planning process are to choose the appropriate development environments for both the frontend and backend implementations. The following section contains a description of the best development solutions and the reasoning behind choosing one of them.

3.1.1 Backend development environment

Java development has been popular for over twenty years by now. Therefore, there is plenty of Java IDEs and different tools to work with them. Some of the solutions are open-source and supported by the community, while others remain either paid or distributed under the annual subscription plans. The main difference between them is not only Java and the framework supported versions, but also the targeted software. Some IDEs on the market aim at mobile development, while others are best suited for server-side related tasks.

There are three most popular Java development environments nowadays. Each of them has been on the market for a long time. All the environments have gained significant popularity in the developers' community. Net Beans IDE is the earliest one in chronological order. As an open-source project included to the Apache Software Foundation, Net Beans is widely supported by a big community. Updates are released quickly. The coding process is very developer-friendly. Net Beans provides features such as code correction while typing and harmless code refactoring. Furthermore, IDE has a built-in Maven and Ant framework toolsets. (Heller 2018.) This lets executing all the commands inside a development

environment and not having to use a console. It is also worth mentioning that Net Beans was acknowledged as an official IDE for the Java version eight (Singh 2019). Nevertheless, there are a few noteworthy weaknesses which are taken from my development experience. The first and the foremost is poor documentation support provided by Oracle. Therefore, any functionality question related to Net Beans may remain unanswered. That is a big disadvantage. Furthermore, IDE is not equally well optimized for all the operating systems. For instance, Net Beans' performance on Mac OS is far below the expectations. These complications often make developers consider other development environments.

Another strong competitor is named IntelliJ IDEA. It comes in two versions. The first one is free of charge. However, it is designed for mobile application development mainly. On the other hand, the paid version is dedicated to the web development process. It is packed with features and plugins that are necessary for building backend servers written in Java. (Heller 2018.) IntelliJ IDEA has the best development performance on the market. Edge cutting code completion and compilation features simplify Java programming significantly. Furthermore, IDE has a large number of frameworks supported right out of the box. Above all, the paid features of IntelliJ IDEA are provided for students at no cost at all.

Last but not least is Eclipse made by the IBM company. According to Heller (2018), IDE's name originates from the company's intent to "eclipse the MicrosoftVisual Studio" development environment. And the name has paid off as Eclipse was long praised by the community as the best Java IDE ever created. Nowadays, the latest Eclipse version is fully packed with automatic correction features that enhance the development process to a great extent. What is more important, Eclipse is not only distributed for free, but also supported across all the known desktop platforms. JDK is known for its plugin-based structure. In other words, there is a specific section in Eclipse's interface responsible for plugins downloading. These add-ons allow to significantly increase the functionality that the development environment provides. For instance, plugins may add support

for other programming languages or useful Java frameworks. These features combined make Eclipse lightweight and productive. (Heller 2018.)

Both IntelliJ IDEA and Eclipse are strong IDEs with similar features available to the developer. The first competitor outperforms Eclipse in terms of stability as IDE responds quicker and smoother, based on my experience. Nevertheless, the choice was partially affected by the corporate IDE choice in the Metatavu company. I have gained experience in all the aspects of working with Eclipse there and it was decided to keep using Eclipse for the thesis project. Furthermore, Eclipse is the only IDE to have an integrated WildFly server connectivity. Bearing in mind that this application server is used for the current project, this makes Eclipse a preferable IDE among its competitors.

3.1.2 Frontend development environment

Compared to Java, React development has not had time to grow such a large developers' community. Although it is showing significant growth increase in the past couple of years, there are not many development environments available specifically for React. These are usually fully-fledged solutions that offer capabilities for complete web-development including support for TypeScript (JavaScript), HTML and CSS. Therefore, in companies, it is quite common that developers choose frontend IDE based on their preferences, while backend IDE is common for all employees.

Two main React development environments to consider for thesis project are Atom and Visual Studio Code. Ever since Atom has been released by GitHub, it was praised as a "hackable text editor for the 21st Century" (GitHub 2019). It has a plugin modular structure that affects the root of the IDE. Therefore, developers have the ability to precisely adjust the Atom for their needs by installing plugins. (GitHub 2019.) However, Atom's main downside is a slower function in comparison to its main competitor.

Visual Studio Code was developed by the Microsoft company. It also implements a plugin-based structure. However, plugins only change the top levels of IDE

without any effect on roots. (Wouk 2019.) This attitude offers high overall performance and response speed. Visual Studio Code was originally planned as a lightweight development environment that has support for any desired programming language. Therefore, it has gained a major community with huge threads relating to any possible coding issue. This fact significantly simplifies programming with Visual Studio Code. What is more important, GitHub was acquired by Microsoft Corporation. This means that Microsoft's own product, Visual Studio Code, will be developed and maintained better than its stepbrother.

Furthermore, I have experience of working with both tools. It was found that Visual Studio Code offered a plain simplistic design and all the necessary features to develop high standard UI. The reasons stated above made Visual Studio Code the best choice for this thesis work.

3.2 Server development

Functionality for Pharmacy web aggregator was built on top of an existing Java server. The thesis project was started during the author's training in a software development company. Therefore, the initial idea found it's implementation on the existing REST API backbone application. This made the backend implementation significantly simpler and allowed to fit the entire project into a reasonable amount of hours. The following sections contain description on a Java server application development steps.

3.2.1 RESTful API

The API for the Java backbone application was designed using the Swagger Editor toolset. That allows modifying the API code in a live mode. All the generated specifications are shown on the right-hand side of the application. Furthermore, Swagger Editor has an automatic error detection mechanism. It marks sentences of code containing errors red. Although the error description is provided alongside with the markings, it may be unclear for developers. Nevertheless, those confusion situations are easily resolved by referring to the online Swagger documentation (SmartBear Software 2019a
).

After the API code has been created it has to be converted to Java classes implementing API's functionality. That is when another tool called Swagger Codegen becomes crucial. It allows generating client files straight from the Swagger Editor online tool or from a swagger.json file using the CLI. As the second option better suits the author's preferences it is seen in more detail. In order to provide a standard Mac OS Terminal application with the Codegen capabilities, the Homebrew package manager is used. As seen in Figure 10, a Homebrew command installs a complete Swagger Codegen toolkit locally to be used inside of a console application.

Figure 10. Swagger Codegen console toolset installation

That only command deploys the toolset and makes it functional straight out of the box.

Planning API functionality

Any stable Application Programming Interface has to be thoroughly planned before it is implemented. There is a list of requirements that the author has set for this project's API. These are:

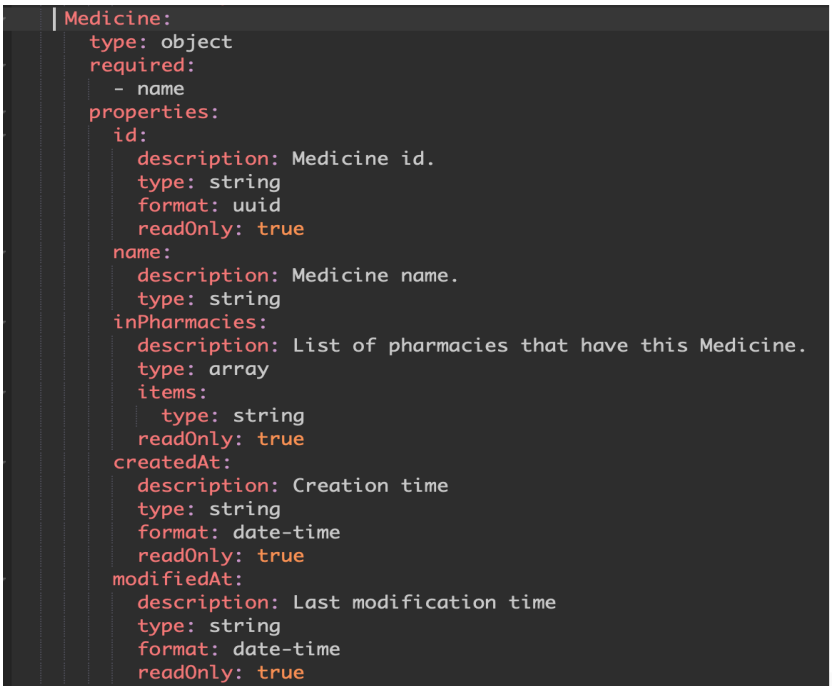
- API should be targeted at the medicines which are distinguished based on their unique IDs.
- All the necessary error codes must be supported.
- Created API endpoints for all medicines and specific ones separately.
- GET and POST methods support for the global endpoint.
- GET, PUT and DELETE methods support for the local endpoint.

Therefore, by design it is planned to have two endpoints "/medicines" and "/medicines/{medicineId}". The first endpoint stands for the multiplicity of medicines and the second one represents any single unique medicine. According to the list of points above, the first endpoint has two functional methods. GET

method allows to retrieve all the medicines and POST method is used to create an entity of medicines or update an existing one. On the other hand, the second endpoint has GET, PUT and DELETE methods which allow retrieving one medicines data, creating a new medicine entry and deleting an entry respectively. In addition, calling these methods must produce correct response codes. For instance, a successful request should produce code “200” that represents “Success”. Code “400” refers to an invalid request and code “403” tells that request was made by an unauthorized source. Last but not least, a response containing "500" code signals that error happened on the server-side.

Creating API Specification

The plan was completed and the goal reached by designing the desired API. As the project was based on already existing RESTful API Java server, the new functional API for medicines had to be inserted to an existing specification. This means that the design had to correspond to the existing one. Therefore, OpenAPI specification code was written in accordance to an already existing one. The first step was creating the "Medicine" component's definition. It can be seen from Figure 11 below.

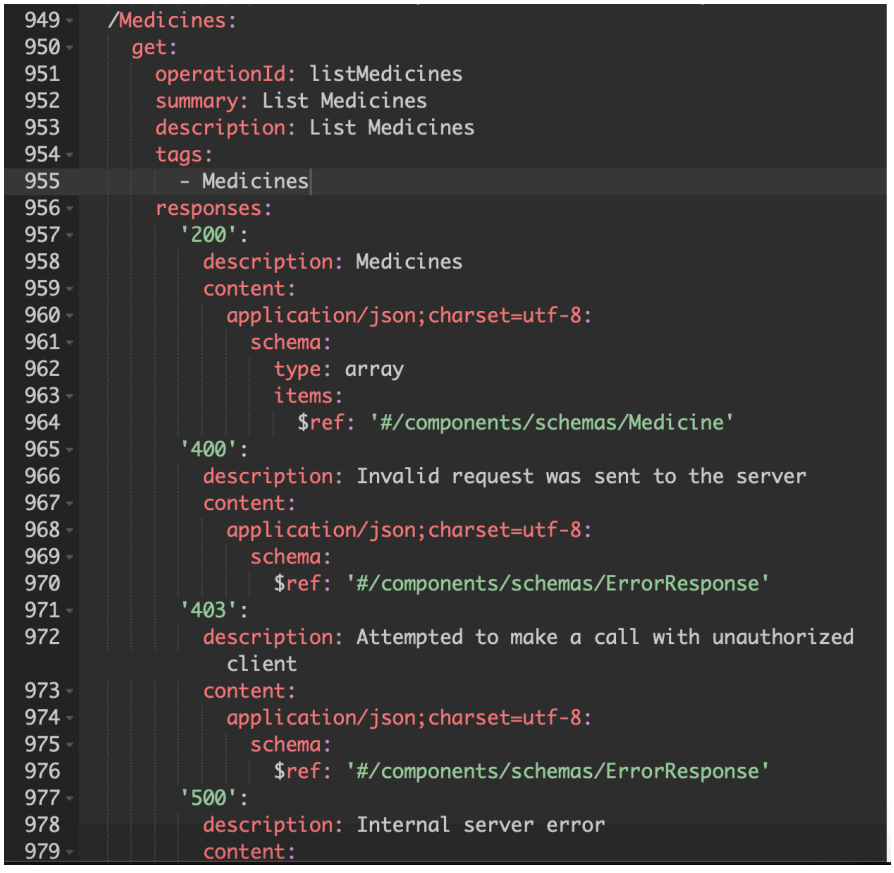


```yaml
Medicine:
  type: object
  required:
    - name
  properties:
    id:
      description: Medicine id.
      type: string
      format: uuid
      readOnly: true
    name:
      description: Medicine name.
      type: string
    inPharmacies:
      description: List of pharmacies that have this Medicine.
      type: array
      items:
        type: string
      readOnly: true
    createdAt:
      description: Creation time
      type: string
      format: date-time
      readOnly: true
    modifiedAt:
      description: Last modification time
      type: string
      format: date-time
      readOnly: true
```

Figure 11. Swagger editor API component definition

This figure contains the OpenAPI Specification definition for the Medicine object. It has properties like “createdAt” and “modifiedAt” which were required by the initial design of an existing API. Furthermore, “id” property stands for a unique identification string written in a natively Java supported UUID format. That allows to precisely define the medicine and distinguish similarly named drugs. Parameter "name" is standing for medicine's name. On the other hand, property “inPharmacies” is defined as an array containing string format data. In other words, this array contains names for all the pharmacies containing specified medicine.

Following the plan, the next step was aimed at defining endpoints and their communication capabilities. Therefore, according to both the initial plan and the API design requirements, two endpoints were created. First one has a “/Medicines” path. This endpoint also has two main requests which are GET and POST. According to Figure 12, each of requests was given an appropriate behaviour defined by HTTP status codes.



```
949  /Medicines:
950    get:
951      operationId: listMedicines
952      summary: List Medicines
953      description: List Medicines
954      tags:
955        - Medicines
956      responses:
957        '200':
958          description: Medicines
959          content:
960            application/json;charset=utf-8:
961              schema:
962                type: array
963                items:
964                  $ref: '#/components/schemas/Medicine'
965        '400':
966          description: Invalid request was sent to the server
967          content:
968            application/json;charset=utf-8:
969              schema:
970                $ref: '#/components/schemas/ErrorResponse'
971        '403':
972          description: Attempted to make a call with unauthorized
               client
973          content:
974            application/json;charset=utf-8:
975              schema:
976                $ref: '#/components/schemas/ErrorResponse'
977        '500':
978          description: Internal server error
979          content:
```

Figure 12. Swagger editor API endpoint description part

This code snippet contains a logic behind the GET request made to the “/Medicines” endpoint. Code “200” stands for a successful REST API request. Therefore, the reference address is "/components/schemas/Medicine” which points to the correct component and, therefore, returns an array of Medicine items. Swagger Editor tool provides a live example of how a request can be processed. It may be seen in figure 13 below.

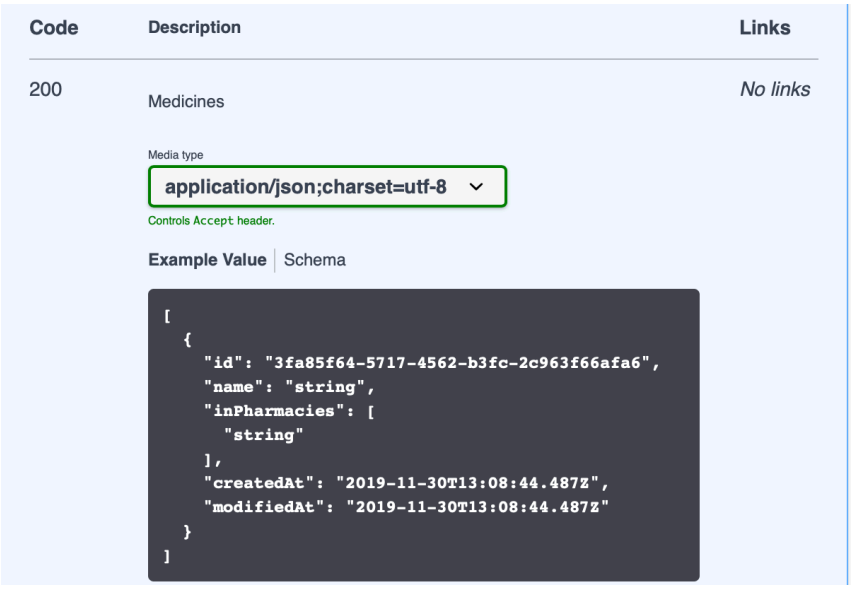


Figure 13. Swagger editor API successful request example

This example represents a possible output backbone is HTTP status code “200” is received. An array of all medicines will be returned as a response.

On the other hand, another endpoint is a branch of “/Medicines” endpoint that allows retrieving, updating or creating specific entries on one of the medicines contained in an array. This endpoint has a “/Medicines/{MedicineId}” path as seen from Figure 14.

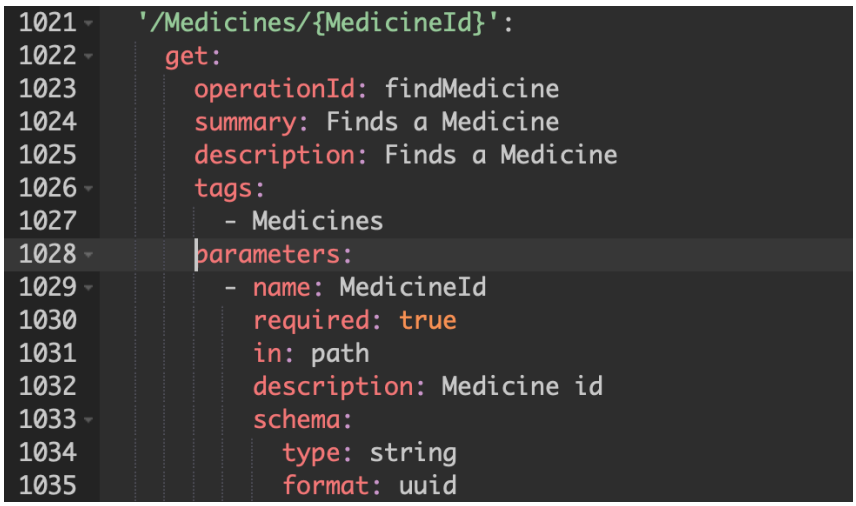


Figure 14. Swagger editor API MedicineId parameter

In order to obtain specific medicine, a corresponding parameter is used here. It accepts a UUID string form and looks for this Id inside of a “/Medicines” endpoint. The further action pattern is exactly the same as in the first endpoint’s case. HTTP response codes are given in a similar manner. The only difference is that the "200" code produces a response containing only one Medicine’s data.

The remaining methods such as POST for the “/Medicines” endpoint, PUT and DELETE for the "/Medicines/{MedicineId}" were implemented similarly. All of them were designed and written to provide all the required functionality. Response codes were also used appropriately. It is worth mentioning, that “404” error code which stands for “Not found” was not included into an OpenAPI code as it exists by default. In case the GET request does not succeed in matching the search parameters to existing resources “404” error code is thrown automatically.

Generating API Client

As it was mentioned at the beginning of Section 3.2.1, the Swagger toolkit also includes CLI tools for automatic client classes generation. Furthermore, Swagger Codegen was successfully installed using the Homebrew package manager. Since the API specification needed for the application was created, it has been downloaded in a JSON format. Next, a terminal command “*swagger-codegen generate -i Downloads/openapi.json -l java*” execution has generated a folder

containing Java classes based on the API. These classes provide full functionality for the medicines endpoints. They allow a full CRUD spectrum of actions which are: create, retrieve, update and delete. Furthermore, all the methods generated already contain the logics for authorization and various cases for HTTP status code responses. In addition, Swagger Codegen creates a Java class of significant importance which is seen in Figure 15.

```java
@javax.annotation.Generated(value = "io.swagger.codegen.v3.generators.java.JavaClientCodegen", date = "2019-11-18T15:41:34.713Z[GMT]")
public class Medicine {

  @NotNull
  @NotEmpty
  @Column(nullable = false, unique = true)
  private UUID id;

  @NotNull
  @Column(nullable = false)
  public static String name;

  @NotNull
  @Column(nullable = false)
  private List<String> inPharmacies;

  @Column(nullable = true)
  private OffsetDateTime createdAt;

  @Column(nullable = true)
  private OffsetDateTime modifiedAt;

  public UUID getId() {
    return id;
  }

  public void setId(UUID id) {
    this.id = id;
  }

  public Medicine name(String name) {
    Medicine.name = name;
    return this;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    Medicine.name = name;
  }

  public List<String> getInPharmacies() {
    return inPharmacies;
  }

  public void setInPharmacies(List<String> list) {
    this.inPharmacies = list;
  }

  public OffsetDateTime getCreatedAt() {
    return createdAt;
  }

  public OffsetDateTime getModifiedAt() {
    return modifiedAt;
  }
```

Figure 15. MariaDB creating a local database

This is a “model” class taken directly from the API specification. It defines actions for Medicine such as getName, getId, getInPharmacies, setName and e.t.c. They will be required in Section 3.2.2 of this work when designing a Data Access Object class for the Medicines. Furthermore, the rest of the actions which require API interaction will be processed using the functions form Figure 15.

3.2.2 Adding database support

In order to implement the main software functionality,the backend application was given database support. This process could be divided into several milestones, which are the following:

- designing the database structure
- creating a database in MariaDB
- creating a Liquibase entry for the database
- writing Java DAO supporting classes

As the backend part is implemented on top of an existing server solution, there is an existing Liquibase changelog.xml file that contains database information. Therefore, the process of adding the new entries was made following the existing rules. The structure of a new table was made similar to the API definition which Section 3.2.1 already introduced.

To simplify the planning process it was decided to sketch a table abstract visualization first. It is shown in Table 3 below.

Table 3. Database table plan sketch

Medicine ID	Medicine Name	Pharmacies	CreatedAt	ModifiedAt
1b2v3-123n3	Paracetomol	String []	11.21.2019	14.21.2019
234n3-j565jk	Aspirin	String []	12.21.2019	14.21.2019

As MedicineID column has unique entries which always differ, this column has been chosen to be a Primary Key. It means that in case the functionality of application grows, other tables may be linked with this one based on the MedicineID column with the Foreign Key attribute. Nevertheless, it was decided to keep a single-table structure as PSQL extensions allow having an array of values in Pharmacies column (PostgreSQL Global Development Group 2019). In addition, CreatedAt and ModifiedAt columns were added to match the already existing database implementation rules.

First of all, a local instance of a MariaDB database had to be created. As Figure 16 shows, it was done using the terminal.

```
DaniilSmirnov1@MacBook-Pro-Daniil:~$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.3.14-MariaDB Homebrew

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create user if not exists sa identified by 'sa';
Query OK, 0 rows affected, 1 warning (0.082 sec)

MariaDB [(none)]> grant all on mmtest.* to sa;
Query OK, 0 rows affected (0.009 sec)

MariaDB [(none)]> drop database if exists mmtest;
Query OK, 17 rows affected (0.219 sec)

MariaDB [(none)]> create database mmtest default charset utf8;
```

Figure 16. MariaDB creating a local database

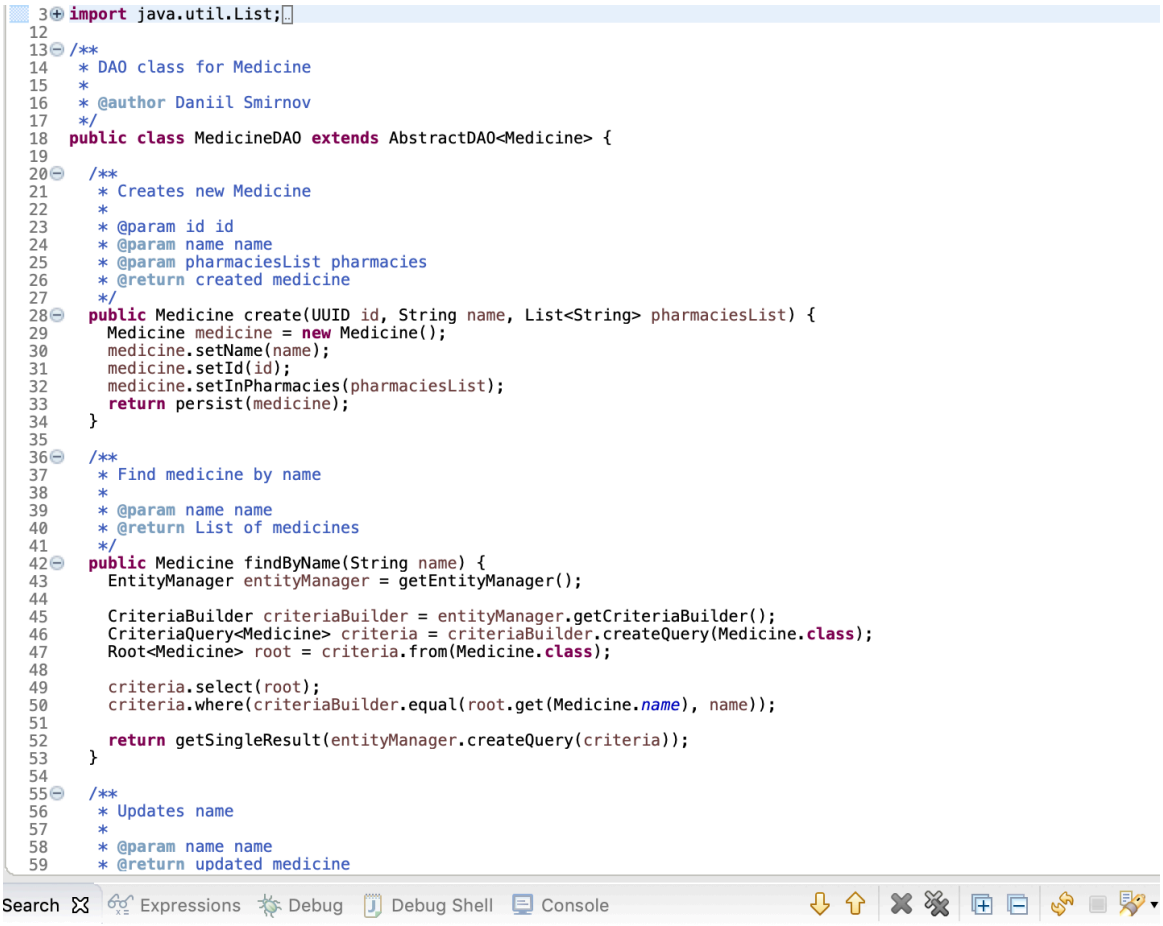
These commands have created a local instance of a database that was used with the Java backend application. All the links for this local database were already provided inside a Maven's configuration pom.xml file. Moving further, in order to add a new table to the database the Liquibase code from Figure 17 was written and added to the main changelog.xml file.

```
...
445    <changeSet id="medicines-table-v2" author="Daniil Smirnov">
446      <createTable tableName="Medicines">
447          <column name="id" type="binary(16)">
448              <constraints unique="true" nullable="false" primaryKey="true"/>
449          </column>
450          <column name="name" type="varchar(191)">
451              <constraints nullable="false"/>
452          </column>
453          <column name="pharmacies" type="VARCHAR(32) []">
454              <constraints nullable="true"/>
455          </column>
456          <column name="createdat" type="timestamp">
457              <constraints nullable="false"/>
458          </column>
459          <column name="modifiedat" type="timestamp">
460              <constraints nullable="false"/>
461          </column>
462      </createTable>
463    </changeSet>
```

Figure 17. Database changelog.xml entry

Figure 17 represents a "changeSet" which contains information on a new table. That includes not only columns, but also the column data types used. A number contained in brackets limits the number of characters that entries in a column may have. This is implemented mainly to use fewer resources and to enhance the application's performance. Furthermore, some columns are made nullable, as they do not necessarily have any data.

The operations described previously in this chapter have created a ground for writing Java DAO classes. Data Access Object is a layer between the database and the rest of the application. It is responsible not only for making database requests, but also for processing the replies. (Sun Microsystems, Inc 2002.) The class written for medicines may be partially seen in Figure 18 below.



```
3 import java.util.List;
12
13 /**
14  * DAO class for Medicine
15  *
16  * @author Daniil Smirnov
17  */
18 public class MedicineDAO extends AbstractDAO<Medicine> {
19
20   /**
21    * Creates new Medicine
22    *
23    * @param id id
24    * @param name name
25    * @param pharmaciesList pharmacies
26    * @return created medicine
27    */
28   public Medicine create(UUID id, String name, List<String> pharmaciesList) {
29     Medicine medicine = new Medicine();
30     medicine.setName(name);
31     medicine.setId(id);
32     medicine.setInPharmacies(pharmaciesList);
33     return persist(medicine);
34   }
35
36   /**
37    * Find medicine by name
38    *
39    * @param name name
40    * @return List of medicines
41    */
42   public Medicine findByName(String name) {
43     EntityManager entityManager = getEntityManager();
44
45     CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
46     CriteriaQuery<Medicine> criteria = criteriaBuilder.createQuery(Medicine.class);
47     Root<Medicine> root = criteria.from(Medicine.class);
48
49     criteria.select(root);
50     criteria.where(criteriaBuilder.equal(root.get(Medicine.name), name));
51
52     return getSingleResult(entityManager.createQuery(criteria));
53   }
54
55   /**
56    * Updates name
57    *
58    * @param name name
59    * @return updated medicine
```

Search | Expressions | Debug | Debug Shell | Console

Figure 18. Data Access Object class for Medicines

The code contains the methods for retrieving information on the required drugs. Furthermore, new entries may be created or removed from the table.

3.2.3 Implementing authorization

Keycloak is responsible for the authorization and authentication process in the Pharmacy Aggregator application. The authentication process is carried by entering username and password as soon as the application page loads. As soon as Keycloak receives correct credentials, it associates it with the existing user. If

a match is discovered, a bearer token is issued to that user's session. A bearer token is a random set of characters which is intended only for machine use. Therefore, it does not carry any human-readable information. That token allows its owner to access resources provided by the application. However, the list of actions is often restricted. Keycloak brings a useful feature here which is named Scopes. A scope allows separating user groups and providing them with limited functionality which is usually decided by the administrator.

The configuration of Keycloak is carried using a web-based tool called Administration Console. It can be accessed from a local machine where the server application is deployed. That is usually done by navigating to a "localhost:8080" address in a web browser. Nevertheless, to avoid any interference the port offset was implemented in a standalone.xml setup file. Once the Administration Console is opened, it has a choice of "clients" which stands for applications that are currently secured by Keycloak. In some cases, multiple pages of the same application may be considered as different clients to provide fine security tuning. When the required client is chosen, there are four main security configuration tabs presented. These are the following:

- Resources tab represent locations or data inside the application which has to be secured.
- Authorization Scopes tab allows creating properties for user permissions (better suited for groups of users).
- Policies tab serves for creating condition-based access rules.
- Permissions tab is used to combine rules from previous tabs into the actual authorization allowances.

A policy may be created based on many different conditions. However, the most applicable for the current project was a "User-based" policy.

As the application developed during this implementation part is only a test version, it does not have any actual users. Therefore, the simplest authorisation approach was chosen. A resource named Medicines was created to represent the API endpoint of "/Medicines". Next, a policy called user-role-policy was created. It has the purpose of allowing inside only customers with the "user" role assigned by the administrator. In addition, Figure 19 shows permission that was created lastly.

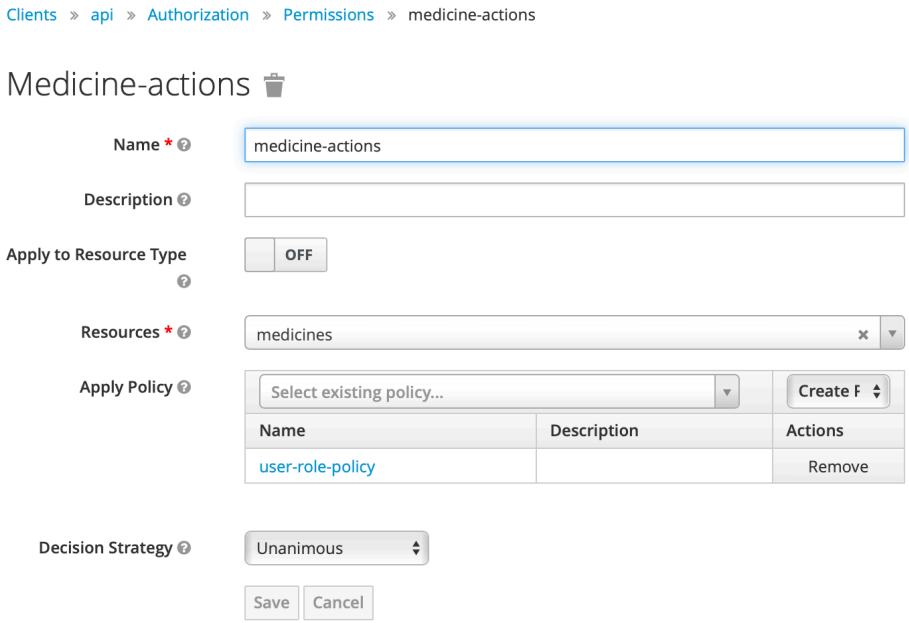


Figure 19. Keycloak Permission creation

This permission combines the previously created resource and policy to produce a final rule used for approved authorization. The unanimous Decision Strategy option, seen in the bottom of Figure 19, makes the permission work only in case all prerequisites are met successfully. This setup allows users to gain authorized access to the Pharmacy Aggregator application. The processes of token retrieval, verification and distribution were already defined by dedicated Java classes in a server application as previous applications based on this API were utilizing authorization services.

3.3 Web application development

The initial idea of the Pharmacy Aggregator application was modified during the implementation process. A series of conversations with Finnish university professors indicated that it would be virtually impossible to order prescription medicines using a web application. Therefore, a decision was made to include not only medicine search capabilities, but also a function to request medicines. This added functionality allows the final application to be used even when the majority of medicines is only available on a prescription basis.

3.3.1 UI Planning

The user interface was decided to be minimalistic, as this is just a prototype application. The planned functionality does not require a large number of elements on the screen. The application was designed in a single page approach. Therefore, most of the elements must be present on one initial screen. At first, upon user login, an authentication screen provided by Keycloak is displayed. As soon as the user is authorized, the main view opens. It should contain a search text bar which would be used for entering medicine names. Under that there is a window showing the result of a search. If the search button is pressed and no results were found the result window must display a “No results found warning”.

According to the initial design changes, the application screen should contain a medicine request form. It must contain fields for user personal data such as name, surname, phone number, e-mail, address, personal identification number, medicine and a desired pharmacy. Furthermore, a field that allows uploading a file could be useful for sending the prescription documents. The request form is located on the main page under the search field. It is a good location as the user will know exactly how to order new medicines as soon as he starts using the application. In addition, a bar containing user account details is planned to be located on top of the application’s screen. It is the usual location in most of the applications nowadays.

3.3.2 UI Implementation

Implementation started with a Keycloak setup. It has been configured and integrated into the React application. To deliver a high safety level, the user is redirected to the login page upon application launch. In case of success, the user is presented with the rendered application where he can use the full functionality. These redirect rules and the Keycloak configuration is partially shown in Figure 20.

```
41    public render() {
42      return (
43        <BasicLayout>
44          { this.props.authenticated ? (
45            <div>
46              <Redirect to="/" />
47            </div>
48          ) : (
49            <Grid centered>
50              <Grid.Row>
51                <Header textAlign="center" >{strings.redirectingTokeycloak}</Header>
52              </Grid.Row>
53            </Grid>
54          )}
55        </BasicLayout>
56      );
57    }
58    private doLogin() {
59      const kcConf = {
60        clientId: process.env.REACT_APP_AUTH_RESOURCE,
61        realm: process.env.REACT_APP_REALM,
62        url: process.env.REACT_APP_AUTH_SERVER_URL,
63
64      };
65      const keycloak = Keycloak(kcConf);
66      keycloak.init({onLoad: "login-required"}).success((authenticated) => {
67        if ( this.props.onLogin ) {
68          this.props.onLogin(keycloak, authenticated);
69        }
70
71      });
72    }
73  }
```

Figure 20. React setting for Keycloak

Paths to the API and Keycloak are Environment variables declared in a separate ".env" file. This file with the addresses can be seen in Figure 21 below.

```
.env
1   REACT_APP_API_BASE_PATH=http://localhost:8080/v2
2   REACT_APP_REALM=test
3   REACT_APP_AUTH_SERVER_URL=http://localhost:8280/auth
4   REACT_APP_AUTH_RESOURCE=api
5
```

Figure 21. Environment Variables for Keycloak setup

Furthermore, the actual Keycloak client name is also specified in this file. It is “Api” which represents our application in the Keycloak setup and is used to perform permission checks that were configured in Section 3.2.3. Once the user is authorized, he can not only use the UI, but also his session requests to the server now have a bearer token which allows the actions possible for that token type. Only one user was actually added for future testing purposes. Credentials

are simple with both username and password set to "admin". This user was set to have full administrator’s capabilities. It is fine for raw product testing purposes.

After establishing a Keycloak link, the actual UI application was developed. The approach here was that the main React class will render elements based on certain actions. While the search bar is always visible, elements like the ordering form are only called, if the user presses a corresponding button. Here is a brief description of the main UI elements. The search button function is to compare a search field value to the values from a medicines array. The API call is made to retrieve the array. Next, it is inspected using the “map” function. Inside that function, each medicine’s name is compared to the search field value. In case of a match, the result is shown under the search field with an order button. This gives the user the ability to instantly order or book a needed drug. By the initial idea, user accounts already provide sufficient proof of identity to be able to reserve medicines without any additional actions. Nevertheless, this feature was not implemented, as it is still a test application. A bit of code responsible for elements render is shown in Figure 22.

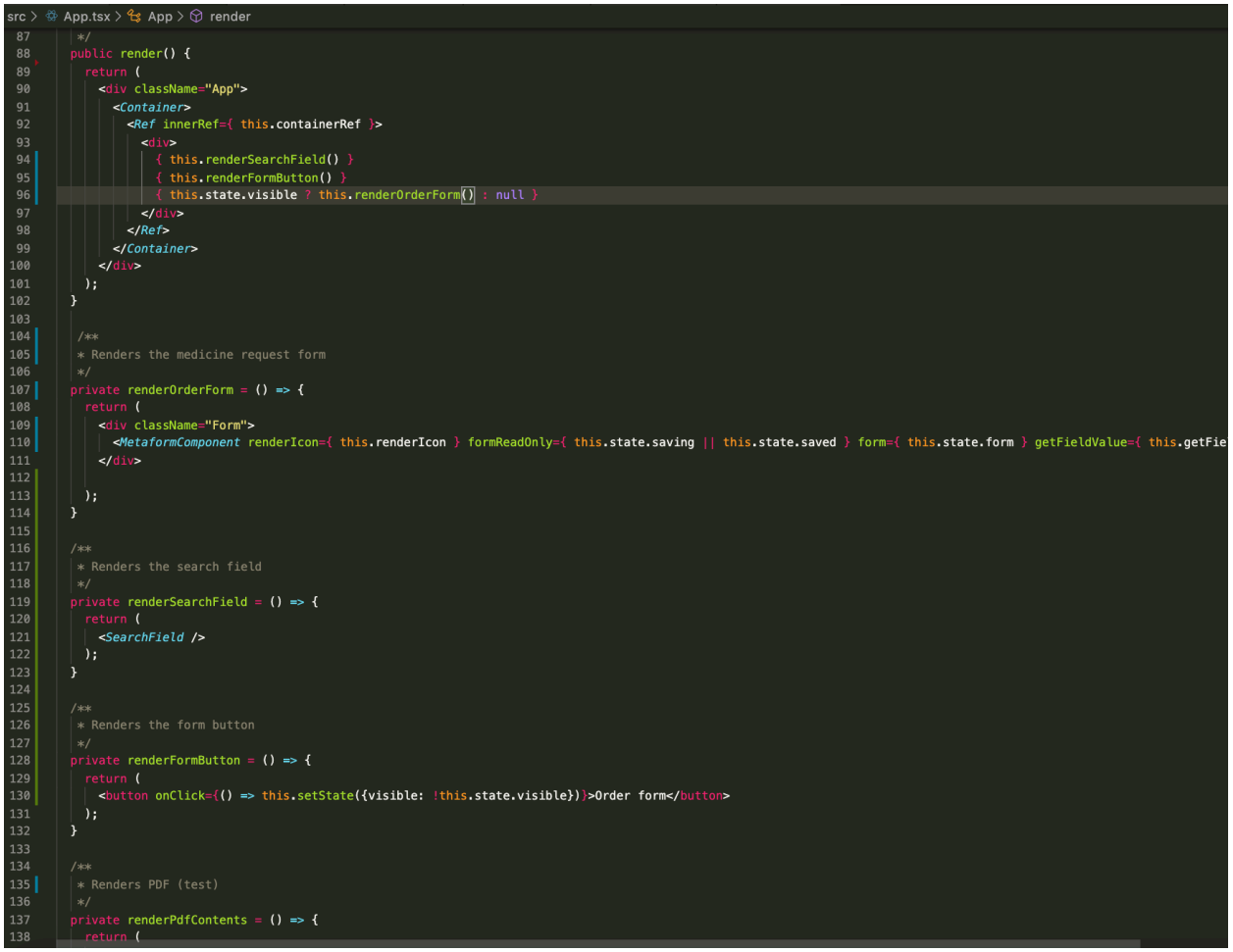
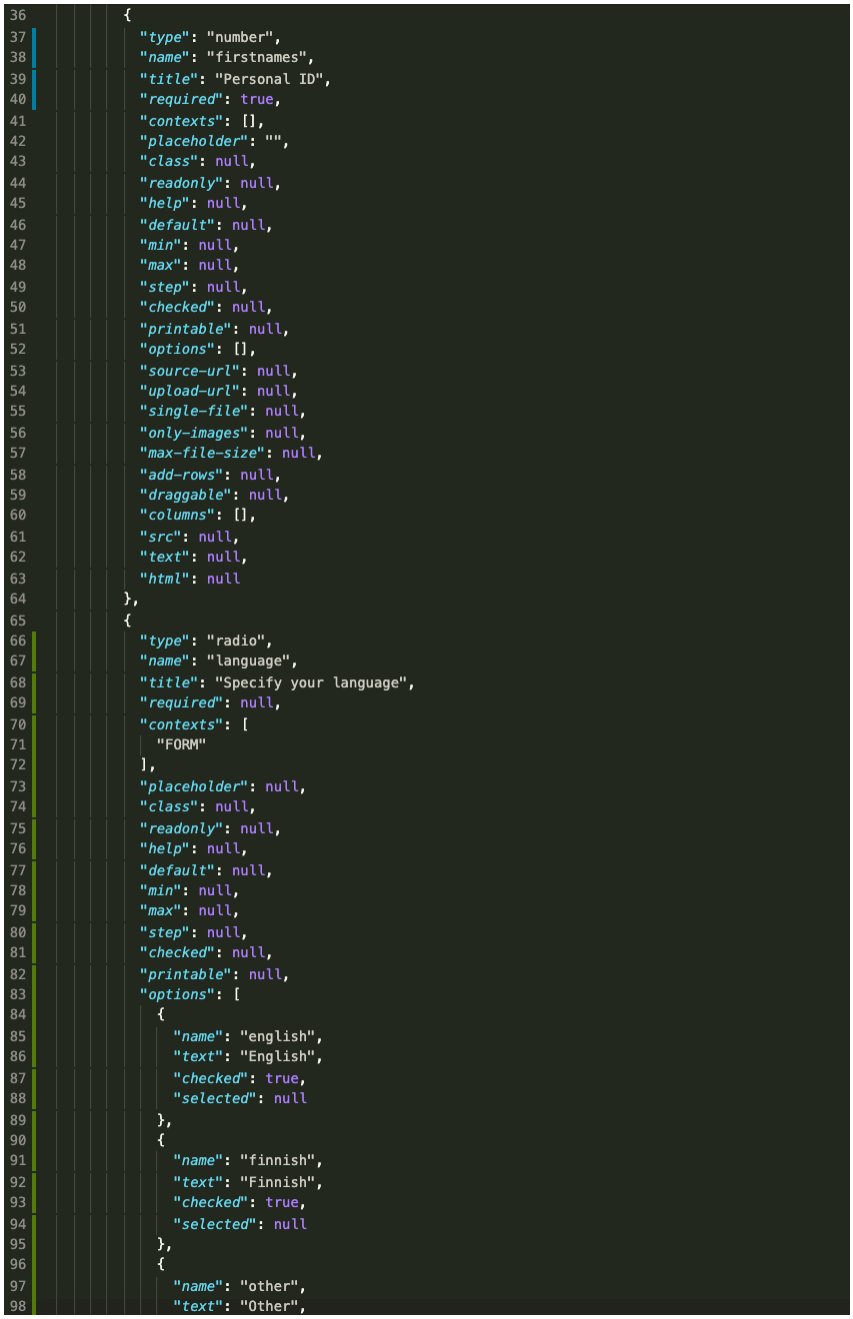


Figure 22. Main UI React class

The private methods which render elements are called in the main render method. Moreover, the correct order is kept.

An order form became one of the most important application's elements. The decision was made to create multiple React components that would render form elements from a JSON file. Therefore, a JSON standardised format was created with the purpose of fitting all the possible types of components. This is shown in Figure 23. Not all the parameters were used while implementing this UI. Nevertheless, it is good to have certain expandability for the medicine request form, since a set of fields may change depending on many factors.



```
{
  "type": "number",
  "name": "firstnames",
  "title": "Personal ID",
  "required": true,
  "contexts": [],
  "placeholder": "",
  "class": null,
  "readonly": null,
  "help": null,
  "default": null,
  "min": null,
  "max": null,
  "step": null,
  "checked": null,
  "printable": null,
  "options": [],
  "source-url": null,
  "upload-url": null,
  "single-file": null,
  "only-images": null,
  "max-file-size": null,
  "add-rows": null,
  "draggable": null,
  "columns": [],
  "src": null,
  "text": null,
  "html": null
},
{
  "type": "radio",
  "name": "language",
  "title": "Specify your language",
  "required": null,
  "contexts": [
    "FORM"
  ],
  "placeholder": null,
  "class": null,
  "readonly": null,
  "help": null,
  "default": null,
  "min": null,
  "max": null,
  "step": null,
  "checked": null,
  "printable": null,
  "options": [
    {
      "name": "english",
      "text": "English",
      "checked": true,
      "selected": null
    },
    {
      "name": "finnish",
      "text": "Finnish",
      "checked": true,
      "selected": null
    },
    {
      "name": "other",
      "text": "Other",
```

Figure 23. Main UI React class

For demonstration purposes, two component types specifications are displayed in this figure. While the “number” field is standard, the “radio” field contains a choice button called “other”. An important function was implemented to display an extra text input field specifically for that case. As a result, the user is always presented with the freedom of choice. Despite that, this JSON file contains a lot of code. The process of adding the setup for new React components is just a matter of copying a skeleton code and pasting type, name, title and so on. Moreover, React code does not take as much space with the implementation of the JSON

component specification file. Figure 24 displays a code snippet returning a “number” field React component.

```
42
43   /**
44    * Component render method
45    */
46   public render() {
47     if (!this.props.field.name) {
48       return null;
49     }
50
51     return (
52       <form>
53         <input
54         type="number"
55         placeholder={ this.props.field.placeholder }
56         id={ this.props.fieldId }
57         aria-labelledby={ this.props.fieldLabelId }
58         name={ this.props.field.name }
59         title={ this.props.field.title }
60         required={ this.props.field.required }
61         readOnly={ this.props.formReadOnly || this.props.field.readonly }
62         value={ this.props.value || "" }
63         onChange={ this.onChange }
64         onFocus={ this.props.onFocus }
65         />
66         <footer style={{ color: "red" }}>{this.state.validationMessage || ""}</footer>
67       </form>
68     );
69   }
70
71   /**
72    * Event handler for field input change
73    *
74    * @param event event
75    */
76   private onChange = (event: React.ChangeEvent<HTMLInputElement>) => {
77     this.props.onValueChange(event.target.value);
78     this.setState({
79       validationMessage: event.target.validationMessage
80     })
81   }
```

Figure 24. Number React component

Component is created using native HTML5 input types. Furthermore, other parameters are retrieved from a JSON file in Figure 23. The other components were built in a similar way which created a form system that is not only functional, but also easy to modify for a wide range of purposes.

3.3.3 Styling

When it comes to application development, the UI's look and feel perform one of the key roles for the end-user experience. Therefore, a Pharmacy Aggregator application was designed to look attractive to potential users. The overall design approach was kept minimalistic. That has not only made the interface clear and

simple to use, but also allowed keeping the code clean. This is even more important, as it helps with the application's troubleshooting.

Cascading Style Sheet was used to enhance the visual experience of this thesis project. A CSS file was referenced from the main React class that is responsible for rendering the entire application. This style sheet is shown in Figure 25.

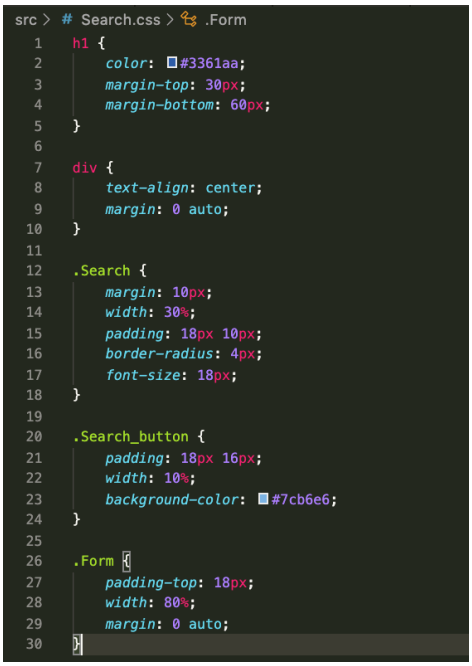


Figure 25. CSS used in a Pharmacy Aggregator application

This file contains general styles which apply to all the and <h1> HTML elements. Furthermore, dedicated style rules are also present for the search field, search button and the medicine request form.

3.4 Finalized version demonstration

Implementation steps were left behind meaning that the product is ready. The pair performance of a Java REST API server and the React user interface application shows stable performance. Figure 26 shows a login screen.

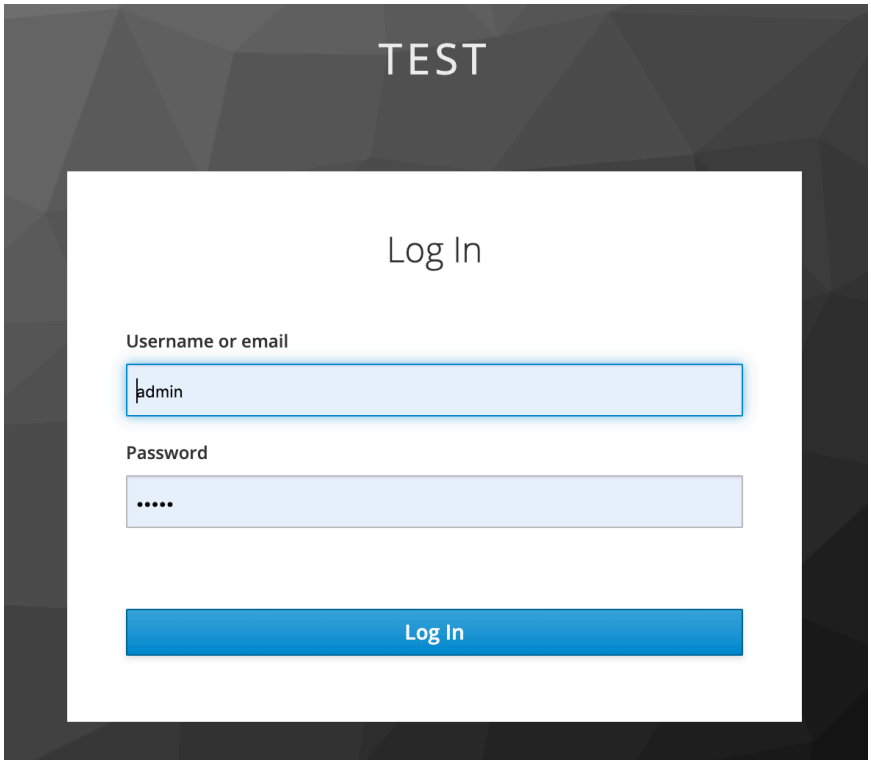


Figure 26. Application login screen

It is the first view that the user sees upon the application's launch. It is followed by the main screen shown in Figure 27.

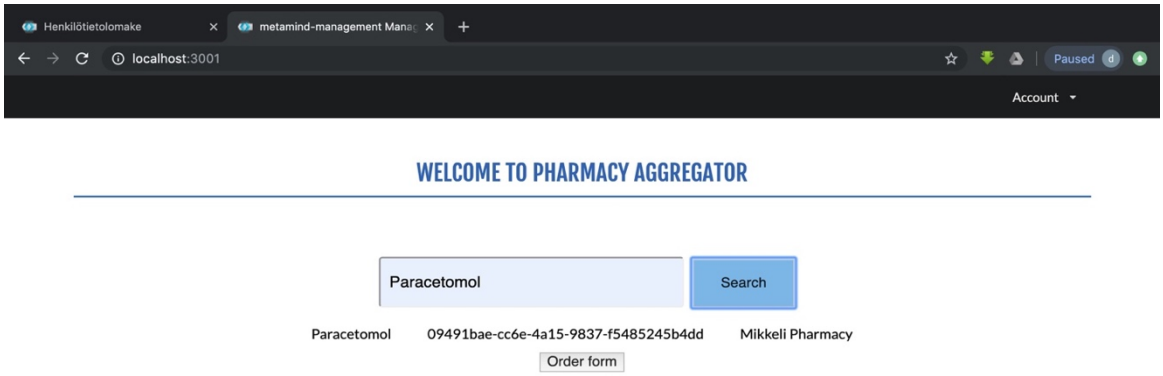


Figure 27. Main application screen

There is a search bar that is used to find medicines. After the drug availability is checked, the user may click the order form button to open a request form shown in Figure 28 below.

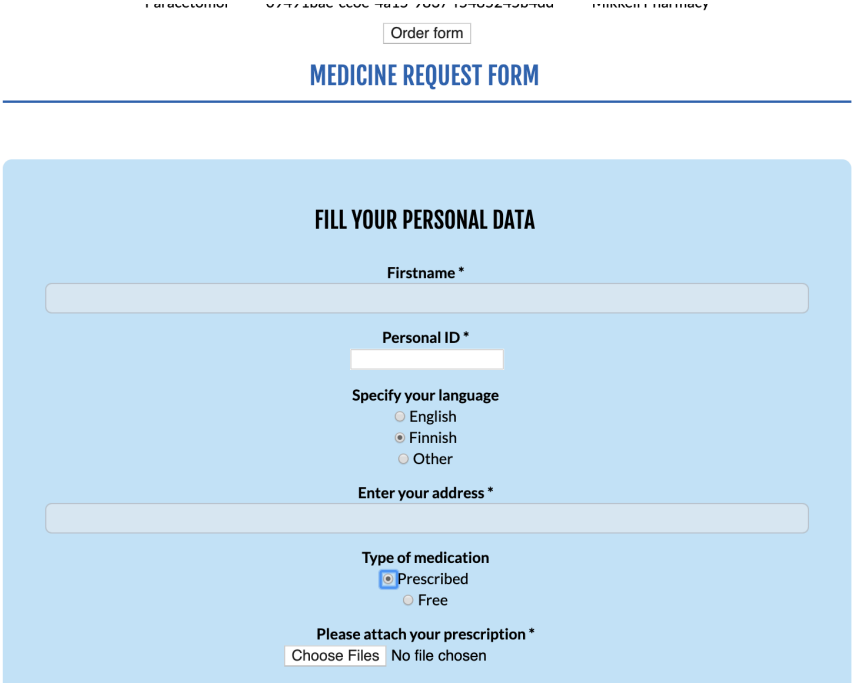


Figure 28. Request form view

This form is then stored by the application. In case the application finds a real use case, the request form may be sent to actual pharmacies to implement the project's planned functionality in a real scenario. However, by now this is only a prototype application that was created to apply the theoretical knowledge, gathered during the study.

4 CONCLUSIONS

The initial result for this thesis work was planned as a functional web application which performs a defined set of tasks. However, this plan has been modified multiple times in accordance with the new information gathered on the medicine distribution subject. While the initial plan assumed that the Pharmacy Aggregator application will be used in Finland, an implementation has changed it to the absence specific location. This happened as the law regulations can vary significantly for different countries. Therefore, a final development decision was made to make it country neutral.

The new design brings a possibility for the Pharmacy Aggregator web application to be used in most of the countries nowadays. It allows both ordering the

medicines directly from the UI or booking required ones for further collection from a specified pharmacy. The application has a simplistic design which prevents any confusion for the end-users. All the planned and modified functionality was implemented successfully in Chapter 3 of this work. The web application allows searching for medicines, authorized medicines ordering and reserving medicines from a pharmacy. Furthermore, additional functions were added, such as attaching the prescription file while placing a medicine order.

The development process involved a significant amount of tools, technologies and frameworks to be used. I have gained not only theoretical knowledge while writing the theory basis, but also practical experience on all the points described in Chapter 2 and Chapter 3. The implementation part required a lot of time and research as the software development process always allows for different ways to solve one problem. However, having studied the theory thoroughly, I was able to make some correct development decisions and come up with a completely functional solution.

As Pharmacy Aggregator is a prototype application, it has bugs and minor UI issues. Moreover, the interface may seem a little too simple for the actual product. It is only a matter of fact that the application did not have a chance to work in a real-life scenario. However, it’s purpose was to show how the theoretical skills in various IT topics may be combined to produce a working application. And the Pharmacy Aggregator is not only well implemented, but also carries a bright idea of making people’s lives slightly better.

REFERENCES

Aderinokun, I. 2018. What, exactly, is the DOM. WWW document. Available at: https://bitsofco.de/what-exactly-is-the-dom/ [Accessed 6 October 2019]

Apache Software Foundation. 2018. Introduction to the Standard Directory Layout. Image. Available at: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html [Accessed 6 September 2019]

Apache Software Foundation. 2019a. What is Maven. WWW document. Available at: http://maven.apache.org/what-is-maven.html Accessed 25 August 2019]

Apache Software Foundation. 2019b. Maven Getting Started Guide. WWW document. Available at: https://maven.apache.org/guides/getting-started/index.html [Accessed 25 August 2019]

Apache Software Foundation. 2019c. Frequently Asked Questions. WWW document. Available at: https://ant.apache.org/faq.html#ant-name [Accessed 27 August 2019]

Apache Software Foundation. 2019d. Introduction to Maven Plugin Development. WWW document. Available at: https://maven.apache.org/guides/introduction/introduction-to-plugins.html [Accessed 4 September 2019]

Apache Software Foundation. 2019e. Introduction to the Dependency Mechanism. WWW document. Available at: http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html [Accessed 4 September 2019]

Borrelli, P. 2019. Angular vs. React vs. Vue: A performance comparison. WWW document. Available at: https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/ [Accessed 7 October 2019]

Brown, K. 2019. What Is GitHub, and What Is It Used For. WWW document. Available at: https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/ [Accessed 20 November 2019]

Codesido, I. 2009. What is front-end development. WWW document. Available at: https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost [Accessed 1 October 2019]

Conor. 2005. The Early History Of Ant Development. WWW document. Available at: http://codefeed.com/blog/the-early-history-of-ant-development [Accessed 27 August 2019]

Dubey B. 2018. Difference between TypeScript and JavaScript. WWW document. Available at: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/ [Accessed 25 October 2019]

Facebook Inc. 2017. React lifecycle methods diagram. Image. Available at: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/ [Accessed 20 October 2019]

Facebook Inc. 2019a. Getting Started. WWW document. Available at: https://reactjs.org/docs/getting-started.html [Accessed 10 October 2019]

Facebook Inc. 2019b. React Native. WWW document. Available at: https://facebook.github.io/react-native/ [Accessed 12 October 2019]

Facebook Inc. 2019c. Render Props. WWW document. Available at: https://reactjs.org/docs/render-props.html#use-render-props-for-cross-cutting-concerns [Accessed 16 October 2019]

Facebook Inc. 2019d. Component State. WWW document. Available at: https://reactjs.org/docs/faq-state.html#what-is-the-difference-between-state-and-props [Accessed 19 October 2019]

Facebook Inc. 2019e. Components and Props. WWW document. Available at: https://reactjs.org/docs/components-and-props.html#props-are-read-only [Accessed 20 October 2019]

Facebook Inc. 2019f. State and Lifecycle. WWW document. Available at: https://reactjs.org/docs/state-and-lifecycle.html#adding-lifecycle-methods-to-a-class [Accessed 20 October 2019]

Flanagan, D. 2006. JavaScript - The Definitive Guide. Fifth Edition. O'Reilly, Sebastopol, California: O'Reilly

GitHub. 2019. Atom frontpage. WWW document. Available at: https://atom.io [Accessed 8 November 2019]

Harrington, J. 2017. Keycloak web configuration interface. Image. Available at: https://blog.jonharrington.org/static/integrate-django-with-keycloak [Accessed 26 September 2019]

Harvey, M. 2001. The Nuts and Bolts of College Writing. WWW document. Available at: http://nutsandbolts.washcoll.edu/nb-home.html [Accessed 6 August 2010]

Heller, M. 2018. Choosing your Java IDE. WWW document. Available at: https://www.javaworld.com/article/3114167/choosing-your-java-ide.html [Accessed 6 November 2019]

JavaTpoint. 2018. Difference Between Ant And Maven. WWW document. Available at: https://www.javatpoint.com/difference-between-ant-and-maven [Accessed 01 September 2019]

Javin, P. 2017. What is the difference between Maven, ANT, Jenkins and Hudson. WWW document. Available at: https://javarevisited.blogspot.com/2015/01/difference-between-maven-ant-jenkins-and-hudson.html [Accessed 30 August 2019]

Kagga, J. 2018. Understanding React Components. WWW document. Available at: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb [Accessed 18 October 2019]

Keycloak. 2019a. About Keycloak. WWW document. Available at: https://www.keycloak.org/about.html [Accessed 20 September 2019]

Keycloak. 2019b. Server Installation and Configuration Guide. WWW document. Available at: https://www.keycloak.org/docs/latest/server_installation/index.html [Accessed 25 September 2019]

Keycloak. 2019c. Securing Applications and Services Guide. WWW document. Available at: https://www.keycloak.org/docs/latest/securing_apps/index.html [Accessed 25 September 2019]

Langley, N. 2002. Write once, run anywhere. WWW document. Available at: https://www.computerweekly.com/Articles/2002/05/02/186793/write-once-run-anywhere.htm [Accessed 20 August 2019]

Liang, Y. D. 2010. Introduction To Java Programming. Eighth Edition. Upper Saddle River, NJ: Pearson.

MariaDB Foundation. 2017. Choosing the Right Storage Engine. WWW document. Available at: https://mariadb.com/kb/en/library/choosing-the-right-storage-engine/ [Accessed 3 November 2019]

MariaDB Foundation. 2019a. About MariaDB. WWW document. Available at: https://mariadb.org/about/ [Accessed 1 November 2019]

MariaDB Foundation. 2019b. InnoDB and XtraDB. WWW document. Available at: https://mariadb.com/kb/en/library/innodb/ [Accessed 4 November 2019]

Microsoft Inc. 2019. TypeScript. WWW document. Available at: http://www.typescriptlang.org [Accessed 28 October 2019]

Niemeyer, P., Knudsen, J. 2000. Learning Java. First Edition. Sebastopol, California: O'Reilly & Associates, Inc.

NPMJS. 2019. About NPM. WWW document. Available at: https://docs.npmjs.com/about-npm/ [Accessed 24 November 2019]

OpenID. 2019. Welcome to OpenID Connect. WWW document. Available at: https://openid.net/connect/ [Accessed 28 September 2019]

Oracle. 2019. MySQL. WWW document. Available at: https://www.oracle.com/database/technologies/mysql.html [Accessed 31 October 2019]

PostgreSQL Global Development Group. 2019. PSQL Documentation. WWW document. Available at: https://www.postgresql.org/docs/9.5/arrays.html [Accessed 16 November 2019]

Red Hat. 2014. Getting Started with WildFly 8. Table. Available at: https://docs.jboss.org/author/display/WFLY8/Getting+Started+Guide#GettingStartedGuide-WildFly8DirectoryStructure [Accessed 15 September 2019]

Red Hat. 2017. What is WildFly. WWW document. Available at: https://wildfly.org/about/ [Accessed 10 September 2019]

Red Hat. 2019a. WildFly is Jakarta EE 8 Certified. WWW document. Available at: https://wildfly.org/news/2019/09/12/WildFly_Jakarta_EE_8/ [Accessed 10 September 2019]

Red Hat. 2019b. Developer Guide. WWW document. Available at: https://docs.wildfly.org/16/Developer_Guide.html [Accessed 12 September 2019]

Ross, B. 2019. How to Use Typescript with React and Redux. WWW document. Available at: https://medium.com/@rossbulat/how-to-use-typescript-with-react-and-redux-a118b1e02b76 [Accessed 29 October 2019]

Rouse, M. 2019. Database (DB). WWW document. Available at: https://searchsqlserver.techtarget.com/definition/database [Accessed 31 October 2019]

Singh, V. 2019. Best Java IDE 2019. WWW document. Available at: https://hackr.io/blog/best-java-ides [Accessed 6 November 2019]

SmartBear Software. 2019a. Swagger Editor Documentation. WWW document. Available at: https://swagger.io/docs/open-source-tools/swagger-editor/ [Accessed 13 November 2019]

SmartBear Software. 2019b. About SwaggerHub. WWW document. Available at: https://app.swaggerhub.com/help/index [Accessed 14 November 2019]

Sun Microsystems, Inc. 2002. Data Access Object. WWW document. Available at: https://www.oracle.com/technetwork/java/dataaccessobject-138824.html [Accessed 18 November 2019]

W3resource. 2019. MySQL Security. WWW document. Available at: https://www.w3resource.com/mysql/mysql-security.php [Accessed 4 November 2019]

WildFly. 2019a. Java EE Api. Image. Available at: http://www.mastertheboss.com/other/faqs/what-is-wildfly [Accessed 14 September 2019]

WildFly. 2019b. Getting to know WildFly folder structure. WWW document. Available at: http://www.mastertheboss.com/jboss-server/jboss-configuration/getting-to-know-wildfly-folder-structure [Accessed 15 September 2019]

Wouk, K. 2019. Visual Studio Code vs Atom: Which Text Editor Is Right for You. WWW document. Available at: https://www.makeuseof.com/tag/visual-studio-code-vs-atom/ [Accessed 8 November 2019]