Daniil Zhitnitskii

SOFTWARE INTEGRATION WITH ADP HR

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 Zhitnitskii	Bachelor of Engineering	September 2019
Thesis title Software integration with ADP HR		68 pages
Commissioned by BenefitScape		
Supervisor Timo Hynninen		

Abstract

The main goals of this thesis were to study the certain set of the web-development frameworks, which consisted of Java EE Spring Framework and Angular 8 framework, and to create and implement a software product using these frameworks. The work is divided in two major logical parts: theoretical and practical. Each part represents different viewpoints on the process of software development in theory and production.

The theory part of this thesis described all the technologies which were used in the process of final product creation, their basics and working principles. These included descriptions of technologies used in the backend and the frontend of the application alongside with the supporting tools and frameworks. This combination approximately illustrated the knowledge basis required for creation of the final product.

The practical part consisted of chronological description of the web application development. The whole development cycle from the initial idea of the workflow to the final state of the software product were described. Implementation part was focused on the creation of web application called ADP Connect. The application was designed in order to provide a solution for integration with ADP HR platform. The software's main responsibilities were retrieving the certain information from the platform and processing of this information.

The final result of this research was a created fully functional web application with working integration system and implemented SSO. The application was delivered in the requested timeframe with all the requirements satisfied.

Keywords

Java, Spring, Angular, ADP, software integration

CONTENTS

1 INTRODUCTION

Business evolution has always been one of the strongest motivating forces for development of different environments. Information technology industry was not left apart from it. As the scales and amount of operation for companies rapidly grow, the new tools, which will meet the upcoming needs, are required constantly. Cloud computing has become one of these vital tools in IT business. As cloud providers possess enormous resources of computing power, smaller companies are no longer in need of constantly having up-to-date hardware and software components on their own. They can always utilize infrastructures provided by the third-party companies.

Automatic Data Processing (**ADP**) Inc. has become one of the biggest players in the cloud computing market. The company, which performs main part of their operations on the territory of the United States, provides a wide range of solution for HR business segment, payroll processing and tax information handling.

BenefitScape – the commissioner of this study builds their business on taking care of ACA compliance guiding for their client companies in the USA and providing required data to Internal Revenue Service of the USA, known as IRS. Briefly, ACA compliance is a set of rules and actions which takes care of all the employees' benefits (like insurances, retirement payments etc.) and ensures that no company disobeys provisioning these benefits to their workers (U.S. Department of Health & Human Services 2019). BenefitScape's clients utilize different tools for their data provisioning, including ADP HR cloud. However, capabilities of ADP HR cloud currently do not meet the needs of BenefitScape and their partners. The software cannot provide the required data in required format in required time.

So far, the purpose of this work was studying and understanding the theoretical basis for building an application integrated with ADP HR system, exploring the main concepts and preferred patterns of software development under required languages paradigm and the creation of the software itself. The desired outcome of final product is a mechanism for filling in Excel tables automatically with data

for IRS in required format and manageable web-portal for owner organization and clients. The scope of the work includes languages and frameworks basics alongside with advanced materials on the fields required for final product creation. The excessive information covering utilized technologies and non-related materials about the ADP platform and their business are excluded from this thesis project.

Due to lack of resources, BenefitScape could not solve the problem earlier, even though it could improve automation of their operations and ease the working process. The project was supposed to be carried out in a totally new manner comparing to the company's previous projects, meaning that it could not be treated as a routine set of tasks. Because of that, it was decided to provide an opportunity for young developers, prospective future employees to build the solution for the problem.

The result of this study provides an autonomous data processing tool for gathering information, which is supposed to be sent to IRS, and its further formatting in required standards. The solution is represented as a web-application which provides autonomous data gathering tool and user interface for both BenefitScape and their clients. The thesis work is generally aimed for description of project work implementation and the description of the software project itself. Previously, depending on the terms of agreement, all this work was supposed to be done either by a representative on BenefitScape's client side or by one of the members of BenefitScape. But according to the company's data, the majority of clients could not provide data processing on their side. Because of that, the solution provided gives a much more efficient way in accomplishing the task.

2 THEORY PART

The theoretical basis of this work includes definitions and descriptions of the main technological concepts used in the final solution. These include the concepts of the main programming language used in this project, utilized frameworks and technologies in the created product, the main aspects and features of the working

environment (ADP HR) and the organizational aspects of constructing a project under a third-party environment.

2.1 Introduction to Java and Java frameworks

Java has recommended itself as good choice for software developers of any environment and any platform, starting with traditional desktop environment and finishing with the smallest wearable devices running under Android software, for example. Java project has been supported and maintained for many years by Oracle Corporation. Therefore, it was selected as one of main frameworks for the project without considering other alternatives.

Java's main ideology is described by the WORA principle – "*Write Once, Run Everywhere*", the slogan of Java founders – Sun Microsystems (Rouse 2014). Comparing to other programming languages, Java does not translate human understandable source code directly into machine understandable binaries. It provides a middle stage in this process. Before converting the source code into binary code, Java compiler first translates it into bytecode – a special intermediate phase of translation, which can be understood by Java Virtual Machine (**JVM**). Further, JVM takes care of translating bytecode into binary low-level code, choosing a suitable format for the system where the action is taking place. According to this pattern, the same high-level human readable code can be executed on different platforms using an appropriate JVM. (Figure 1).

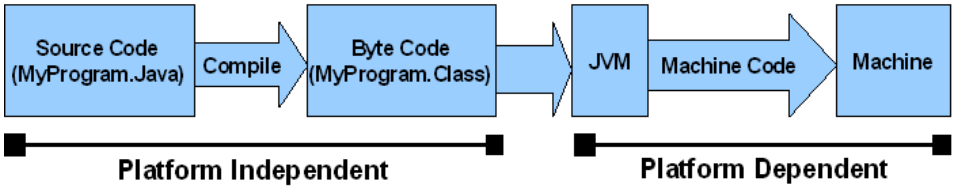


Figure 1. Java source code translation (Malkajgiri 2018)

As all the software environment bases on Java with the support of frameworks for backend, it was decided that this thesis work will also be based on Java. Even though it is commonly supposed that Java is a desktop or mobile application

programming language, it can also be easily used as a backbone for web-applications. The tools required for that, including an integrated development environment (**IDE**), are discussed later in this work.

2.1.1 Java frameworks

This section describes frameworks and their main features which have been used during the development of the final solution created during this thesis. The section focuses on theory and the descriptions of the technologies and reasons for their usage. The practical aspects of the implementation are described in the section 3.2 of this work.

2.1.1.1 Maven

The first thing to mention here is a tool for code compilation, project building and dependency management. Initially, Java Development Kit (**JDK**) – the core artifact of Java development and base for many programs and utilities from different vendors, which can be found in most of the personal computers, can take care of code compilation (Tyson 2018). However, building and dependencies management are not included in JDK functionality. For simple Java projects (like ones not aimed for a production environment) this approach can seem suitable. The only thing needed for a Java program to run is to write at least one text class file with *.java* extension, compile it in the terminal window with *javac* command, which will create a compiled file with the extension *.class* in bytecode, and then run the program with *java* command.

Nevertheless, such approach of executing a program becomes completely irrational when we start to talk about Java projects, containing hundreds of classes and external dependencies. In such conditions the transfer of programs as plain *.class* files does not provide flexibility for code deployment, which is an essential part of integral and continuous development and is happening far more than one time in life. Because of that, a special format called Java Archive (**JAR**) was developed. JAR files are used for storing multiple compiled classes of Java code, or even making it possible to execute programs directly from an archive.

(Oracle Inc. 2019.) Such approach allows transferring all the project sources in one file and makes the deployment process fast and clean. According to the fact that JDK is unable to handle JAR creation itself, the third-party tools are required.

Previously, one of the best alternatives for creating Java executables was Ant framework, developed by Apache Software Foundation (2019). Using XML configuration, Ant can handle any type, which can be described in terms of targets and tasks, including build of Java projects into executable JARs. However, Ant itself is unable of dependency management. In order to fill this gap Apache Foundation proposes the usage of Ant in pair with their other product called Apache Ivy (Apache Software Foundation 2019). Having this couple can seem enough, but Apache Software Foundation provides much handier tool for these tasks. Because the XML configuration for the Ant builds of projects do not have a standardized pattern and all configurations can vary and not suite for other projects, Apache Software Foundation (2019b) has provided developers with a better solution which eliminates the problems mentioned above.

Maven framework provides a unified way of project building and handling dependencies. Started as a temporal solution for an Apache Jakarta Turbine project, which also suffered from the inconsistency of Ant configuration, Maven has become a powerful tool for development simplification. (Apache Software Foundation 2019b.)
The configuration of Maven builds bases on the project object model (**POM**) – an XML file which contains all the information about the *module* it is attached to. According to Apache Software Foundation (2019c), POM files typically contain the following information:

1. Project information. This section of POM files describes a group of projects, to which a given project belongs, the id name of the project, version of the project and the final goal of the build, either a JAR or a **WAR** (Web Archive) file. WAR files are not considered and described in this thesis, as they are beyond the scope.
2. Project properties. This part describes build properties, such as source and final files' encoding, locations for generated executable files and

source files, locations of directories containing tests (source and generated), versions of core frameworks and the Java language, the project utilizes and so on. Also, this section may include information on the parent Maven project which the current project relies on. Maven projects' inheritance are discussed later in this section.

3. Dependencies. As the name says itself, this part of the configuration contains information about external dependencies which are used in the project. Information to link dependencies include a group id and artifact (member of Maven projects group) id, which can also be supplied with a desired version of the dependency for consistent development or other parameters, for example the scope of dependency (if it is required during compilation process, or build process, or runtime etc.).
4. Build settings. They include the final goal of the build, a directory where the generated files should be created, resources to include in the build, definitions of plugins supporting the build etc. Plugins, available for Maven, provide a wide spectrum of possibilities for build extension. Besides core plugins, which are provided by default for every building process, side plugins can give functionality for writing documentation, running SQL scripts from files. Also support for core project frameworks is provided etc. (Apache Software Foundation 2019b.)

Maven projects can be constructed in a hierarchical way. This means, that any project can have a parent Maven project whose properties will be applied to a "child" automatically. Following this approach, working with several Maven projects can be significantly simplified. For example, all the dependencies described in a POM file of the parent project will be distributed among all the child projects, which makes it unnecessary to include these dependencies in every POM file for each of the projects. (Apache Software Foundation 2019b.)

The other convenient feature is that all the operations which are applied to a parent project, such as compilation or installation (build) will also invoke the same operations for all the dependent projects. This way, the whole dependency system has an ability to be checked if it is buildable using one command. This feature is principal, as it often happens that the same project can be built from the

first time on one machine, but fails on another after code transfer, for example to a remote server. For development it is vital for all the provided versions of code to be able to compile and be buildable. Otherwise, the whole development process of a project team can be deadly stacked.

Projects can also be related to each other not only in terms of parent-child, but also with a hierarchical structure. In other words, one project can consist of several projects. This way the root project will require a special `<modules>` tag which contains identifiers of all subprojects under given project.

Classic way of building folder structure in Maven project, proposed by the framework vendor, is described in Table 1. The structure will be considered in more detail in the implementation part of this work.

Table 1. Maven project folder structure (Apache Software Foundation 2019)

`src/main/java`	Application sources
`src/main/resources`	Application 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

Also the IDE is capable of automatic editing POM files when common dependencies are detected in the project and are not listed in the configuration yet.

2.1.1.2 Spring Boot

Spring Boot framework was needed to make use of Java as the web-project backend basis possible. Before talking exactly about Spring Boot in particular, Spring Framework – the basis of Spring Boot – must be discussed in general. Spring Framework is a tool which gives developers the opportunity to focus on the application level business logic without the need to care about specific environment deployment aspects. In other words, it easily adapts to the deployment platform, following defined desired properties. (Pivotal Software 2019a.)

Spring supports different languages for code basis, including Java, Kotlin, Groovy etc. Generally speaking, Spring can be understood as a container of frameworks, like JSF (Java Server Faces) for user interfaces modeling, Hibernate for working with database information in terms of **JDBC** (Java Database Connection) and so on. (JavaTPoint 2018a.)

According to JavaTPoint (2018a), the advantages of using Spring framework are the following:

1. Code templates. Spring gives developers ready code configuration for the various modules of application development, especially for database work and **ORM** (Object Relational Mapping). Thus, a programmer does not need to care about declaring the entire process of creating transaction, which includes opening a connection, creating statement for changes, commiting the transaction etc. It is only required to write a desired statement and that’s it. Spring will take care of the rest, including exception handling.

2. Loose coupling and prompt development. Due to utilized dependency injection and inversion of control (**IoC**), Spring applications are more flexible for maintenance and changing environments, accompanied by increased speed of finishing the applications due to strong frameworks support.
3. Lightweight for programming. Due to **POJO** (Plain Old Java Object) implementation, Spring makes more loose rules for developers, like nonobligatory implementation of every interface or abstract class etc.

Dependency injection and inversion of control, which were mentioned in point two of the advantages list, represent a central part of the framework. The components created with the inversion of control container, which are called **beans**, play the role of bricks in Spring projects. These are Java POJOs with a special set of conventional properties, which will be covered closely in the practical part of this work. One principal thing to mention here is that IoC creates those beans and configures them, linking beans together and providing their instances when requested. (JavaTPoint 2018b.)

One significant disadvantage of Spring is the initialization of a project. It may require a valuable portion of time for developers to set up all the required dependencies, obtain access for a server which will store the application, deploy and run WARs containing application servers to provide services required by applications and many other things. The other problem is that Spring applications can not be easily deployed using JAR files, even though the deployment process was already simplified . These problems were eleminated with the introduction of Spring Boot, which has started a new era in development under the Spring framework paradigm.

Spring Boot is a Spring module which introduces Rapid Application Development to Spring. As mentioned above, it eliminates the problems of pure Spring, which allows the creation of stand-alone Spring-based application with the minimum of initial configuration required. Alongside with providing embedded web-server containers for running the application, Spring Boot helps with the creation of POM files for Spring Maven projects. (JavaTPoint 2018b.)

One more valuable feature of the Spring Boot module is automatic confugiration, whenever it is possible. This feature provides inital bootstrap for configuration settings with predefined preferences. However, in case of need, this feature can be disabled and standard manual configuration can be used. Speaking about web applications, automatic configuration supported by the Maven starter module spring-boot-starter-web allows starting a functional application immediately after initial bootstrap. (JavaTPoint 2018b.)

For ease of application boostrap, Spring Boot provides an initializer tool called Spring Initializr. Using this, all the initial information about Maven or Gradel project, concerning this application, can be added via a simple web-form or text fields in IDE plugins, initial dependenices can be added to POM file from the scratch and also special Spring modules, for example security or MVC frameworks, can be supplied. The Initializr tool can be accessed from the Spring website, which will give a downloadable archive after successful project bootstrap, or with the plugin in an IDE with the required information on modules supplied. (Pivotal Software 2019a.)

Spring Boot application plays the role of the backend component of the final product in this study. It provides the backbone of the entire web application and provides API endpoints for fetching database information and performing required calculations and operations. The exact created and utilized features will be considered in the practical part of this thesis.

2.1.1.3 Spring Security

Spring Security framework is a solution for authentication and access-control, or, in other words, authorization and management for all Spring-based applications. It is a de-facto standard, which can be customized for any level of requried security, from a standard couple of the username and password for basic security to OAuth v2 token-based authentication. As the final solution of this work was supposed to work with the sensetive data, the security aspects were considered as the ones in top priority. (Pivotal Software 2019b.)

As this thesis project bases on the Maven Java project implementation, further descriptions will consider the Spring Security implementation under a Maven project. The first thing to do in order to use the security framework in a Spring project which utilizes Maven is to include Spring Security dependencies in the project's POM file. Depending on a type of security technology (which is going to be) used, the different security modules should be imported. If we are talking about basic security for the web-based application, importing of a web-security module and a configuration module will be enough. The configuration module is required in order to be able to change a default configuration, which enforces the security of all the exposed URLs of the applications with basic authentication. If some of them should remain publicly accessible, or in terms of Spring Security have *ananymous* access, according to Pivotal Software (2019b), or some advanced requests filtering is supposed, then the configuration module should be imported regardless of what type of security is going to be implemented. For implemeting, for example, LDAP or OpenID authentication, the import of corresponding modules is needed. The implementation of advanced authentication techniques was excluded from this thesis project and remain out of scope. Therefore, they will not be considered.

As mentioned above, Spring Security enables basic authentication for all the **API endpoints** (accessible URLs) by default and discards all the requests which do not pass an authentication. Data used for credentials comparison is stored in the code by default, or in terms of the framework the *in-memory authentication* is used (Pivotal Software 2019b). In other words, they are passed as string type parameters to the authentication manager and are stored there for the whole time the application is up. However, such approach does not allow flexible user management and limits the possibilities for creating of new users and using self-registration. Because of that, Spring Security gives an opportunity to specify data sources which will be utilized as the user's data providers. In a most convinient way, it is database stored information. Specifying usage of JDBC instead of in-memory authentication requires just manifesting this type of data provision in a security configuration file, either XML or code-based. (Pivotal Software 2019c.)

The configuration of Spring Security can be done in two ways similiarly to Spring framework in general. The developers have a choice to use the XML-based configuration files or set the configuration using built-in class methods, which will be scanned and applied on application initialization. The XML-based configuration was not implemented in this study and therefore it will be left uncovered. For the implementation of the code-based configuration, it is required to create a bean which extends a basic Spring security abstract class called `WebSecurityConfigurerAdapter`. The created bean should have an overriden implementation of a `configure` method inside which the configuration rules are provided. The exact `configure` function used is the one which takes an instance of `HttpSecurity`, which is a **singleton** (a unique spicemen of class inside program's context) instance. Besides this overriden method, `configure` has several other **overloads** (declarations of several functions with the same name, but which have different number or types of parameters) aimed for the other configuration areas. Http configuration usually includes several principal blocks which are stated using a method chaining, e.g. using several methods of a class in a single expression. (Pivotal Software 2019c.)

The usage of protection mechanisms is usually stated first inside the configuration. The mechanisms give the ability to enable **CSRF** and **CORS** protection policies inside the application, which manifest permissions for inter-site communication. By default, both policies are silently enabled in Spring Security. Therefore, inside the configuration they can be either explicitely enabled or just disabled. (Pivotal Software 2019c.)

CSRF (Cross Site Request Forgery) is a kind of attack, whose main technique is based on accessing a 3-rd party website or applications which attack's victim has already access to. For example, using a malicious hidden script or a script executed by clicking some image, button etc., attacker can send a request to the protected resources under victim's credentials. The credentials can be stored as cookie files, or by active session's storage. The utilized tool depends on the exact case. CSRF protection in the Spring Security ensures that the request for confidential data arrives from the trusted source. This can be done, for instance,

by using a token verification or with usage of RESTful APIs. (Mozilla Corporation 2019a.)

CORS (Cross-Origin Resource Sharing) is a mechanism, which allows accessing website's resources, such as API endpoints, from locations differing from a native server. In order for an external server to be able to trigger the endpoint, the request's origin should be allowed by an API hosting server. This becomes possible using additional headers in the request. The request's originator includes its address along with the request so that the reciever could check whether it should allow this request or discard it. The CORS policy was implemented during initial development steps while local machine was used for development and testing, as frontend and backend parts could not be hosted inside the same virtual server. However. After the initial stage the development was taken under remote server, which allowed storing components in the same locations, eleminating need of CORS policy implementation. Therefore, this policy was not implemented in the final version of the project.(Mozilla Corporation 2019b.)

The next building block in the configuration is stating an accessibility of endpoints. All the available URLs of the application can be publically accessible, or saying in terms of Spring Security have *anonymous access*, open for authorized only access or they can be configured to drop all the requests. In case of the anonymous access no authorization is needed in order to reach the resource. Such approach is suitable for login pages, home pages wich contain some general, not sensetive information etc. If the endpoint is configuraed with an authorized access, then the application inspects every request to the protected resource and checks whether the originator has prooved his identity already or not. In case of unauthorized access, the request can either be simply ignored or a redirect to login page can be returned instead. Dropping of the requests is an option for general protection of the domain of applications. All the above rules can be applied as a general rule to all the requests, which do not match specific address. Discarding such requests can protect the application from reconaissance attacks or sript kiddies' activities. (Pivotal Software 2019c.)

Following section of the configuration specifies policy for handling login and logout processes. This includes pointing the URLs which will invoke these processes, setting view files to be associated with those endpoints (not suitable in context of this thesis as all the UI was written with a separate framework) or declaring actions to take after the successful logging in or out. These actions can include redirecting to specified addresses after the process finishing, or, specifically for logging out, it can also wipe out all the authorization data associated with exact session, like setting the closed session as invalid in order to prevetns its utilization repeatedely with an unauthorized access, and deleting cookie files associated with the closed session from the client's storage (Pivotal Software 2019c). Usage of the session for validating requests will be described in the practical part of this work alongside with logging in/out processes.

2.2 Frontend

Now, when the application backbone technologies were studied and the abstract process of how the application backend should be built, it is also important to understand how the UI of the final product should be constructed. Of course, the standard bundle of HTML5/CSS3/JS could be used. However, development using those pure technologies without any additional support would take a lot of time if the aim is to build a dynamic, responsive and well-designed user interface. Besides that, the application frontend should have an internal logic also in order to decrease the load from the Spring backend for better user experience.
Taking into the consideration all the above-mentioned requirements, it was decided to use an Angular framework as the platform for building a face of the application. The framework provides sufficient flexibility for application, ease of development and suitable MVC model to create the required outcome (Google LLC 2019a.)

The Angular is one of the provided distributions inside a NodeJS platform. All the dependencies required for the application and the supporting tools are handled by NodeJS's central repository. If to be precise, the **NPM** is the exact part of the NodeJS platform which is aimed for a general package management. Talking about supporting tools, there is the one to highlight among them – Angular CLI.

This utility's main purpose is application scaffolding and providing the other functionalities such as testing, deployment etc. This is not a vital tool to use during the development of an Angular application, but, for instance, creation of an Angular functional components becomes a deal for single CLI command instead of files creations, manual scaffolding, import stating and so on. Moreover, Angular CLI provides an embedded webserver for development which supports on the fly recompilation of the code after any changes in source files have been detected. (Google LLC 2019a.)

Angular 8 is a framework for creation of single-page applications using HTML with CSS and Typescript languages. Prior to Angular 2, the framework was built using JavaScript as a core language but starting from version 2.0 JavaScript was replaced with its enhanced version. So far, Angular implements all the features of core typescript and its libraries, which are used for supporting dynamic content functionality. (Google LLC 2019a.)

As it was said previously, Angular 8 is primarily used for creation of **single-page** applications. The single-page is the pattern, whose main idea is having all of the web-application's components under a single parent HTML page instead of traditional separation of HTML pages for every URL and logical segment of the website. This means that the only actual public web-resource is a basic *index.html* (conventional entry point for every website) (Kyrnin 2019), whilst all the other segments of applications are not actually opened using redirection, but with its imitation – routing which will be discussed later.

The main building blocks of any Angular application are **modules**. The modules are collections of functional sets of code, which represent logically related segments of applications. Every app always has a *root* module by default for application bootstrapping. The module can be then accompanied by feature's modules. The modules always have a `@NgModule` annotation before a declaration of the module class. All the logical components of the Angular application, except static resources and HTML templates, are simple JavaScript classes with special annotations by **decorators** (special kind of functions which

lets Angular know specific metadata about the class and the class' general purpose) for them. The most important functionality of modules is a creation of compilation context for **components**. The modules allow import of functionality from other modules and export of their own functionality. Breaking the applications in several modules allows utilization of *lazy loading*, which means loading the functionality only on demand which highly increases user experience due to minimized website's loading time. (Google LLC 2019a.)

Components represent functionality of application's views. The components can be reused as many times as it needed in different locations of the application with an adaptation to the required context. The components are inserted by simply using a typical HTML tag inside the templates with id of the component declared in component's Typescript file. Components' classes are annotated with a `@Component` decorator. Every component consists of at least its functional Typescript file with *.ts* extension where all the data bindings, variables and functions are declared alongside with the required dependencies, and an HTML template file. Besides that, the component may also have a *unit test(s)* (isolated test of small piece of program code) file for checking Typescript code functionality and *end-to-end test(s)* (assessment of the program's behavior when user steps in) file to check how the views behave after simulated user events. The unit test's files have additional *.spec* extension and the end-to-end tests have *.e2e-spec* extension. The template files represent a mixture of a traditional HTML markup with an Angular markup which is used for view pre-processing. The pre-processing includes evaluation of Angular's **directives** – the functional pieces of code like loops or conditions, and resolving a **binding markup** – connection between application data and HTML DOM. After the directives' and binding's processing, the initial DOM document is being adapted and then displayed to the end user. (Google LLC 2019a.)

The last important particle of the Angular application are **services**. They are not vital for the application to exist in a general case, but every advanced case of the development requires usage of the services for better code reusability and maintainability. The services represent combined functionalities which are not

tightened to some exact view and which are supposed to be shared across the components. Utilizing services, it becomes possible to make the components free of routine tasks such as logging, user's data input validation, authentication etc. All this work is done by the services. The services are declared with an `@Injectable()` decorator which includes supplied metadata about the provider of the service. Every service can have different scopes where it is defined. The scopes are required for distinguishing the visibility area of services as they are represented as singleton instances. Using the singleton pattern approach lets keeping vital information such as an authorization status or a routing state shared across the whole visibility area. (Google LLC 2019a.)

One left part which can be included in Angular applications are **directives**; however, they are out of scope of this work and are left uncovered so far.

Discussing services in the Angular applications, it is important to mention a specific category of them – *guards.* The guards represent tools for users' authorization. If one of them is implemented, then every request to any component initiated by the user first goes to the guard in order to check whether user is granted permission to access the desired section or not. In order to create a guard, it is required to make a new service class, which implements `CanActivate` interface with the only one method with the same name is the interface. When the guard is created, it can be used for an any declared route by specifying `canActivate` handlers in routing entries inside the routing module. For one route several guards can be specified. (Chenkie 2017.)

So far, the application's structure under Angular framework development have been discussed. However, there is one detail concerned about the Typescript exactly. Since the main ideology and working principles of the Typescript is not that different from a JavaScript, the precise details are left uncovered. Despite that, **observables** are the thing to talk about specifically. The observables represent a data type, which is used for creation of asynchronous flows in the application. The main roles defined in a context of observables are subscriber and publisher. The publisher creates an observable object which provides access

to some sort of data. The subscribers can use those values after triggering `subscribe` method of an observable. Thus, observables are declarative which means that a defined function for publishing values is not fired until the consumer subscribes to it. After the consumer declares subscription to the observable value, for example he set variable value to be equal to observable, the variable's value is tightened to the state of an observable, which can change at some moment or can be left untouched depending on the flow. This is the point where asynchrony steps in. Let us say that a request to the server backbone of the application is made. This can be done using `HttpClient` injectable class of Angular and its method `get` which returns a response as observable object of any type. When the request result is being assigned to a variable, the program execution will not be paused until the result of the request will be received. Instead, it will continue working normally without brakes. When the data will be received, the actions declared for this moment will be fired and the environment will be adapted according to those.

In order for the subscriber to define the actions, the **observers** are used. Those are handlers for the observable's messages which define a callback function to trigger after a certain type of message from the publisher is received. According to Google LLC (2019b), Observers set callbacks for three possible types of messages:

1. *Next*. A handler for *next* type of message defines actions to take after each received value from the publisher.
2. *Error*. This handler tells about itself by its own name. The only thing to mention here explicitly is that observable's instance execution stops if an *error* message was received.
3. *Complete*. This handler steps in when the observable has provided all the values it was supposed to. If there were any values left which were delayed from the subscriber after completion, they will be normally handled by the *next* handler.

Even though Angular is not a Java based framework, Java IDEs sometimes also provide support for Angular development with an Angular CLI integration.

Because of that, both the frontend and the backend projects can be managed from a single place using different compilation configurations.

Thus, the main concepts for the final solution creation have been discussed from the both points of view of functionality and user interface's building. The next chapter will describe the auxiliary tools which have been used during the development of the final product and finalize the general concepts of this thesis's final goal creation. Further content will describe the environment the integration is aimed for and main concepts and tools of that environment.

2.3 Supporting tools

The modern software's development process cannot be constructed in an effective manner without utilization of the supporting tools. Development of the final product of this work was also performed using the little help of these utilities. Those utilities include version control systems, communication and deployment tools. As the working process with the current thesis' commissioner was closer to a freelance model due to remote conditions of work and an enormous day time difference, making the development process as autonomous as possible has become the vital thing during the creation of this work and the implementation of supporting tools was totally required from the initial moment.

Communication tools included such messengers as Skype and *Slack*. Description of the Skype in author's opinion would be excessive in the context of current work as Skype is common day tool which is familiar to everybody and Skype was not the first choice for communication if possible. However, Slack requires introduction in this talk. Slack is a corporate messenger utilized by lots of companies all over the world from small to large business. Among the biggest Slack users there are such companies as Oracle, EA, Airbnb and so on. Every company is given with their dedicated domain called *workspace* which can be accessed either via web or a desktop application. There are available direct messages with all the members of a workspace and channels for the specific purposes inside the workspace. One important feature which Slack provides is an ability to enhance basic functionality of messenger using extensions. There are

thousands of the available utilities, though the ones required for development of this work are definitely related to supporting systems such as GitHub and Jenkins. (Slack Technologies Limited 2019.)

GitHub is one of the most famous implementations of a Git **VCS** (version control system) (Chakon 2014). The GitHub platform, owned by Microsoft Inc., has become vital thing to know and use for majority of the developers worldwide. Being one of the most popular Git VCS, the GitHub has integration opportunities with lots of development tools. As native GitHub features were not used during the solution implementation and due to the fact that specific advanced features of Git technology were not utilized, the specific details about GitHub platform and thorough details of Git are left uncovered. However, the basic principal flow of the system is described in a nutshell.

The Git, as it was mentioned before, is one of the technologies for VCS process. Implementation of this tools allows storing of all the snapped versions of the code with bundling to the code's authors and editors. All the stored versions of code, or *commits* in terms of Git, can be accessed at any time for reviewing or even the whole code can be reverted to that condition. The commits are tiny snapshots of the whole file system of the project at a moment of time. The one noticeable difference of Git comparing to other VCSs is that Git always provides a snapshot of all the files, whilst other VCS only store a list of changes tightened to each file in the snapshots. The commits usually represent a finished implementation of some feature of the product. The features are supposed to be as atomic as possible. In this case a commit *reverting* (rolling back to the previous version of code) will affect the least possible area of the program. (Chakon 2014.)

Besides available history of commits, Git introduces a *branching* system, which serves as a tool for a parallel development of several features in the application or for implementation of totally new features. A branch can be considered as a parallel copy of the initial set of files at some point with an alternative way of development. The copy can be taken at any moment of time where it is required. Created branches can also have their own branches, so the any required amount

and depth of streams can exist. When the feature is finished, the branches can be merged back in a single branch, taking into consideration changes from all the alternative versions of file. In this case composite version of newly added features and previously existed appears without risk to destroy before-working results. (Chakon 2014.)

One more thing which also should be mentioned about GitHub is that it is a public platform which also allows private project storing. As BenefitScape works with the sensitive employee's data, leak of which can lead to prison sentence for the serving company's employees, the whole project was stored in a repository with a private access. Only developers of BenefitScape are allowed to view and manage the contents of that repository.

The last but not the least supporting tool in this project was a Jenkins server application. Jenkins is a self-contained server which is aimed for continuous integration and automation. The tool is served as an open-source project. Jenkins project was continuously and largely supported by such IT giants as Microsoft and RedHat. Utilizing properly configured Jenkins setup, the final product of this thesis project was provided with an ability to test newly introduced features of the created application immediately on a live development server. Jenkins can monitor an activity of the exact GitHub repository and fetch the latest updates from there. Moreover, after getting the source code Jenkins can build it straight-away with the Apache Maven. In addition to that Jenkins also has an extension for Slack, which means that every completed build of the project can be seen as a simple message in a dedicated channel with no need to check the web-interface of a server. The Jenkins setup and maintenance were not handled by the author of this work, this duty was handed over to BenefitScape's DevOps specialist, therefore precise technical details and concepts are not provided. (Continuous Delivery Foundation 2019).

At this point all the main tools which were required for creation of the final solution were described. The next chapter covers the platform which was the integration aim of the product's development. The materials are introduced in

order to provide understanding of how the integration process should have been carried on and which requirements the 3-rd party company implied for all the products being published in its open storages. At some points the information will be limited on purpose due to the privacy terms of the platform, which do not allow spreading sensitive data with non-partners or non-employees of the platform's holding organization. However, it does not affect the overall comprehension of the provided material and only affects exact isolated points of the system.

2.4 API and data provider

The following sections describe the basic information about the platform the integration was aimed to – ADP HR. Its main concepts and purposes alongside with the integration solutions the ADP provides for third-party organizations like this thesis commissioner.

2.4.1 ADP HR

ADP HR is one of the products from ADP's selection. ADP HR provides solutions for client organizations aimed at the centralized management of these organizations' employees and all the information related to these employees. Being hosted in a cloud environment, the HR solution allows all the client organization's assigned workers reach the system from any place and at any time. (ADP Inc. 2019a.)

The information which is managed includes basic personal employees' information such as addresses, marital statuses and so on. Other piece of information stored in the system includes work related information, for instance hiring or termination date, position the person is assigned to, benefits the employee is eligible for etc. (ADP Inc. 2019a.)

Alongside with the information about the workers, the client companies also provide their information to ADP service. The data provisioning allows integral data management and transfer, if needed. One possible way to use this data combination and actually the way which is vital for US companies is the creation

of reports for a Federal Tax System of the United States. All the companies should provide the required information about their employees, including the benefits the employees agreed for or rejected. The benefits in this context mean insurance plans the companies provide for their employees. Also, the possibilities to distribute insurance to the employee's family are considered etc. (U.S. Department of Health & Human Services 2019.)

Alongside with the core functionality of main products, the ADP platform allows functional extensions for the platform via plugins. Besides the main working domain, a special marketplace was created to satisfy customers' needs. ADP marketplace is powered by the AppDirect fintech platform, and all the concept workflow rely on the rules of AppDirect general marketplaces' solutions. The extensions have different purposes and different set of functionalities. But, all of them are aimed for the post-processing or reorganization of data which is stored inside the ADP's system. The final solution of this bachelor's thesis also represents one of the applications available through the ADP marketplace with the only difference that it does not have public commercial setup. The application is only intended for BenefitScape's clients' use.

ADP platform does not provide a lot of information on their systems and architecture publicly, Because of that it was difficult to obtain enough information about the platform. Some pieces of the provided data were gathered during studying process of the system after direct interaction with it and exploration of the web resources. However, the rich information for software development and building an integration solution is provided and is described in the next section.

2.4.2 Integration aspects

As it was said in the previous section, the ADP marketplace – the distribution point for plugin applications is built with use of the AppDirect marketplace platform (2019). In order to understand how the created software should interact with the platform, it is vital to have a right perception of how the AppDirect marketplace works at all. Further AppDirect marketplace will be referred as ADP marketplace as they are equal in the given context.

Financial and technological processes of applications' distribution in the ADP marketplace are built on the subscription model. Access to the software is gained for a certain period according to the paid amount of time. Several types of subscriptions can exist for a single application, for instance a free trial and a full paid version. The type of subscription can be changed at any moment of time or canceled at all. All the cases of subscriptions' creation, changing, expiration and cancelation are handled as *events* in ADP marketplace and once any action is done, both the marketplace and the application are notified about it. The same logic applies to user's actions such as user's assignment. (AppDirect Inc. 2019.)

AppDirect supports more than 10 different events in the application's cycle. However, as the final product of this work was not and is not supposed for public distribution and is only aimed to BenefitScape's selected organizations, the only utilized events will be covered. According to AppDirect (2019), these include:

1. Subscription order. This event indicates the initial step in the application's lifecycle which happens when some user on the marketplace makes a purchase of the selected application. At this point the application can receive all the required information about the user's organization and user himself for further processing depending on application's business logic.
2. Subscription change. Whenever the type of subscription has been switched from one to another, this event is published. Even though the final solution only supposes one type of subscription, it was decided to create a handler for this case also for a prospective implementation.
3. Subscription cancelation. Cancelation occurs when a user or an organization deletes the chosen application from their active subscriptions in the ADP marketplace.

The list above indicated *subscription* events of application workflow. Those are mainly related to internal flow of the interaction between the applications and marketplace and indicate principal points in a service provisioning. As it was stated above, the ADP marketplace also generates *user* events. Again, various user events can be defined for application in the marketplace. However, only project related items are covered. The events are the user's assignment and the user's unassignment. Functionality of these events describes them with their

names. Whenever a user from the ADP marketplace is added to the access group of the application and when the user is removed, the events are being emitted. These events are vital for a security monitoring of the final product's access as the application's owners are not aware other way which users tried to get in the system. (AppDirect Inc. 2019.)

The events described above form a minimum set required by the ADP marketplace. All the events are supposed to be handled with a dedicated API in the application for integral functionality of the ADP marketplace's environment. Before the application can be published all the required API endpoints should be configured and checked with the special tools provided by ADP marketplace. Even if the application is defined to ignore some kinds of events or some types of events are not supported by the application's internal logic, at least some "dummy" replies should be configured for the minimum set of events. Without the successful tests' results the applications cannot be applied for publishing in the marketplace. (AppDirect Inc. 2019.)
So far, the high-level interactions regarding the application in the scope of ADP marketplace have been considered. The next step in the discussion are authentication and data exchange processes. For a stage of the product's development under the scope of this thesis usage of a local authentication, e.g. traditional username and password credentials authentication, was not considered as the first priority in the application's functionality. The most important way stated by the company was access provisioning to clients' users. In order for that ADP provides **SSO** mechanism and standardized patterns of application's interactions with the platform. The mechanism and patterns can be used by developers for ease of products' creation.

SSO, or Single Sign On, is a standard which allows usage of a single set of credentials for gaining access to the different resources. Following this approach, the user is not required to create accounts in every system the SSO applies to. The user's data is stored by identity provider which is represented as ADP platform in a context of this thesis. One of the main protocols to serve SSO environments is OpenID Connect which is implemented by ADP in their platform.

The OpenID is and identity layer built on top of an OAuth 2.0 protocol which allows authorization of end-users in client systems with help of authorization server alongside with providing the basic information on users in a REST-like manner. (OpenID Foundation 2019.)

ADP marketplace (2019b; 2019c) defines two general types of applications separated by their functionality:

1. End-user application. This type of applications supposes direct interaction of a live person with the system and usage of a web application interface. First of all, as a person's information is intended to be provided to application owner, the user is asked for a consent acceptance for his or her data provisioning and storing. After the consent is received during the first launch of the application by a client organization, it will not be asked further. Then, in order to reach the desired resources, the user is required to confirm his identity with the credentials issued by ADP in the ADP authorization endpoint where the user is redirected from the application's entry point. In case of the successful authentication, the user is redirected back to a predefined URL in the application. At this point the SSO process is over and further goes user's data processing with the authorized status in the application. Alongside with the user's redirect, the application receives authorization code from ADP which can be exchanged for access token for the application. This is maid in order to verify that user data is being asked by a trusted source, which should be an application registered in the ADP marketplace with a valid SSL certificate issued and signed by ADP for every specific application. Using this token, the application can retrieve user information for further proceeding. After that the person can continue working with the application as an authenticated user.
2. Data connector application. Comparing to the end-user type, the data connector applications do not require interaction with a user, and they are intended for an automatic data exchange with the ADP platform. The authentication process in data connectors is not much different from the end-user apps except silent consent acceptance. When the client organizations subscribe to these applications, it is supposed that they

agree on the data provisioning from their resources by default. When the subscription is made, the application can access client's resources with use of his credentials provided during the registration process in the application or a password reset process. (ADP Inc. 2019b) The credentials are exchanged for the access key which can be used for fetching data from client organizations, such as information on their employees, employees' benefits etc.

ADP supposes all the applications to identify themselves either as the one type or the another by default. However, both kinds can be combined in a single app. In this case application can provide sort of a user portal for the clients alongside with mechanism for the automatic data retrieval. Applications constructed using this approach will eventually have two user consents, one for the end-user part of app related to user information and the second one related to the data connector app for the general data retrieval.

Once all the endpoints of the application are configured and tested with a filled-out integration report provided by ADP and the marketplace profile for the application is created, the software can be published in the ADP marketplace for a public access. At this point product release process is over. All the utilized technologies and tools have been covered and the environment for the final product were covered so far, the next chapter will describe specific data related aspects and the outcome which the software should provide closing up the theoretical part of this work.

2.5 Data processing and formatting

The last detail about the final solution's specifications is required data to extract from ADP and the format in which all that data should be organized. The data which the final solution is aimed for is represented with ADP *Workers v2* REST API. This collection of the API endpoints provides general information about workers, including their name, surname, social security number (SSN), date of birth, hiring date, living address in all detail, email and a type of employment (part time or full time) including previous work assignment parameters if the given

person was hired twice or more by the same company. For every employee this information is marked with employee's internal ID inside the company and the Employer Identification Number (EIN) of company where employees belong. (ADP Inc. 2019d.)

The second piece of information which is vital for BenefitScape's outcome result are the employee's dependents. In the context of benefits provisioning dependents mean family members of an employee. The dependents may include wife or husband and children. As insurance plans for employees sometimes include a coverage for the dependents, it is required to state this information to IRS in case if the dependents utilize employee's insurance as well. However, by the moment this work was written, the dependents information was not available through the ADP system alongside with benefits information which include insurance plans and their effective dates required by BenefitScape. Because of that fetching mechanism for this piece of information could not be fully accomplished.

In the response to a REST API call ADP system returns a JSON object which represents an array of all available workers related to a chosen company. Nearly all the required information which could be fetched from the system is contained in those arrays. Though, some information was initially supposed to be filled in manually by client organizations' representatives. More precisely these details are covered in implementation part of this work. (ADP Inc. 2019d.)

3 IMPLEMENTATION PART

Initially, at the moment of the implementation of this thesis project the majority of the technologies required for the successful completion of the project required additional studying. Studying and understanding the technologies took place at the same time with code development. Such mixture of the activities created some difficulties and time delays during the working process. However, the process was going on steadily and the development quality was improving day by day.

This chapter describes the creation process of the final goal of this study. A web application called ADP Connect was developed in the implementation part of this work. The content includes the process of the implementation alongside with technological nuances which appeared during the development process. The creation process description starts with a conceptual sketch of the desired outcome and planning of the initial application's structure followed by the implementation of the core application's parts. The content of the chapter is summed up by an overview of the final result with an analysis and comparison of final result with initial requirements.

3.1 Initial concept and application structure

The original idea and vision of the application consisted of a simple idea. The clients of BenefitScape who utilize ADP as their employees' information storage install the BenefitScape's application from the ADP marketplace. After the application is installed, the client's representatives can access the application on BenefitScape's hosting server and view the information they have provided to the company alongside with some insignificant management of the information available to them. Together with the UI available for users the application provides autonomous mechanism for fetching clients' employees' information out of the ADP system, which should be triggered either by a scheduled event or by a "mechanical" button click. As the material result of its work the application should provide a Microsoft Excel file with all the employees' information filled in and formatted.

Talking about the key roles in the original plan for the production, it can be mentioned that the commissioner represented a role of product owner, the one who was responsible for conversations with clients in order to adapt the capabilities of the application. However, at the period of this work creation the clients did not get an access to the application as it was supposed initially. Therefore, the developers and regular workers of BenefitScape were the ones who set the requirements for the product. The UI design concept and requirements were also proposed by the company. The implementation process of both backend and frontend part of the application alongside with the design of business logic were carried out by the author of this work.

According to the desired workflow and the concept, the final solution receives a shape consisting of four major parts: a user portal, an authentication mechanism, an information retrieval mechanism and an export file's creation module. The capabilities of the user portal did not have clear definitions and predictable ways of development. Because of that, it was decided to leave a minimum freedom level for external users initially in order to create a simpler interface and to minimize the potential hazard for the system in case of an inappropriate usage of the provided resource.

The authentication was initially supposed to have two ways for successful accomplishment: either usage of ADP's SSO integrated authentication intended primarily for clients, or with the use of the internal username and password as an alternative, designed to be used by BenefitScape's employees (not only developers, but also other employees). The different levels of access for external users and company users were also planned. Clients would be able to only see the information related to their company, while BenefitScape's users would have access to all the external users' information. The creation of Excel reports was initially planned to be done separately for each company. Because of that, this functionality was decided to be left for all the users, including the clients' users.

When the concept was defined and the requirements for future product were estimated, the actual work began. The first thing in the development process was selection of the environment where to carry out the development and the project structure for the whole application.

3.1.1 Development environment

As Java is one of the most popular languages nowadays and it has enormous community of developers and manufacturers, there is a wide range of tools for the creation of Java programs. Various Java IDEs provide different options for developers, depending on their scope of software development. Because of that, short analysis to make a right choice for further research was made.

First of all, Java IDEs differ by type of devices, which the created programs are intended to use with, and by environments the programs are created for. For example, if a mobile application for Android operating system is created, Android **SDK** (Software Development Kit) or official IDE from Google – Android Studio can be used for this purpose. It is not purely Java IDE. Android SDK has support for various programming languages. However, Java is one of the most strongly supported there. If we talk about enterprise solutions and implementations of Jakarta EE (previously Java Platform EE), then Eclipse is the choice of the community. However, Eclipse was not the best option for this project as it does not provide as good framework support tools as the final choice does. The reason for the selection will be discussed later in this chapter. All the above-mentioned environments are distributed for free and have many alternatives. However, they were not suitable for the current research work.

As the current research work's aim is building a scalable web-application, with a client/server model, further IDEs for desktop or server environments and solutions for middle level load under paradigm of Java EE will be covered in a closer scope. Those are NetBeans, provided by NetBeans community, and IntelliJ IDEA from JetBrains. The very first difference is access level. NetBeans is fully distributed as freeware under a GPLv2 license. By default, NetBeans only gives tools for desktop Java applications development, but it can be easily extended with plugins, provided by the community, including toolkits for web-development. IntelliJ IDEA is provided in two versions:

1. IntelliJ IDEA Community. This is a freeware version of IDE, which is distributed under Apache License 2.0. According to the information, provided on JetBrains official website (2019), this version of IDE is suitable for developing desktop environments and Android operating systems. It does not provide support for web-development (no TypeScript or JavaScript languages integration) and it does not have database working tools. Because of that, this IDE also turned up to not suit the requirements of this project.

2. IntelliJ IDEA Ultimate. This one is a paid version of the software, which provides full support for web and enterprise development, along with all

the available features for Community versions. It also includes useful plugins for many frameworks, which can be used for development, and handy refactoring tools.

Comparing these two alternatives, it was decided to utilize the IntelliJ IDEA ultimate as an IDE for the project due to better quality of frameworks support and development tools, simpler setting up of projects and user support from the manufacturer. Even though the Ultimate edition is not free (to use) and can seem quite expensive for regular production development, JetBrains provides free access to a whole variety of their products to students for the entire period of studies. Because of that, the choice was made.

3.1.2 Project structure

For ease of deployment and maintainability through the different environments, the project was decided to be built as Maven project consisting of several modules. In the Figure 2 all the modules included in the project can be seen, every folder which has a square symbol near its name represents a separate Maven module.

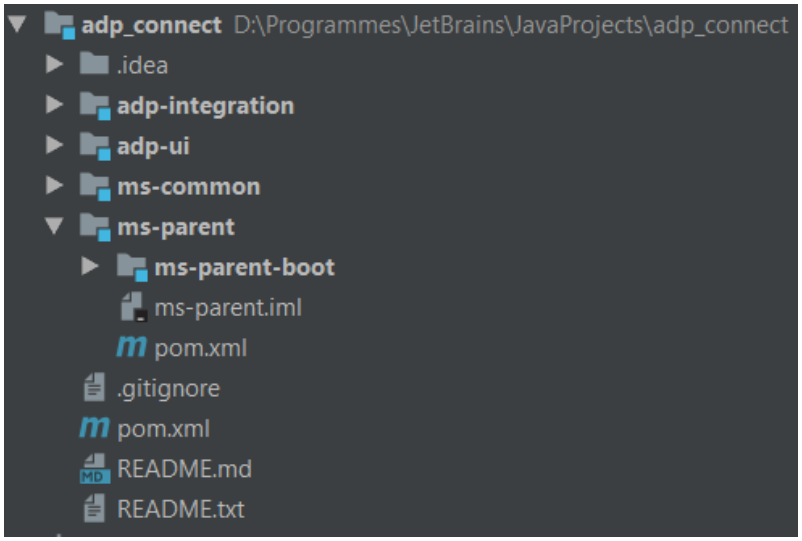


Figure 2. Folder structure

The top adp-connect parent module only provides functionality to declare usage of a deployment plugin. The adp-integration module contains backend part of the project, the adp-ui module stores the frontend part of the application. The

modules called ms-common, ms-parent and ms-parent-boot are BenefitScape's conventional modules for their software. They have been added for future prospective additions and changes to the application. But, for the moment of application creation under the scope of this thesis the ms-common module had no sensitive load alongside with ms-parent module. The ms-parent-boot module provides dependencies handling for the Spring Boot application inside the project module. It states imports for vital dependencies alongside with the general supporting tools like Lombok which provides code generation tool for classes on the basis of annotations. Utilization of this tool eliminates need in explicit declaration of class members' setters and getters. Also, declaration of method used for construction of a string out of a class' instance is also not required, Lombok takes care of it.

The module adp-integration is further organized in a conventional standard of Maven packages organization (Apache Software Foundation 2019c). Folder structure of adp-integration module can be seen in Figure 3.

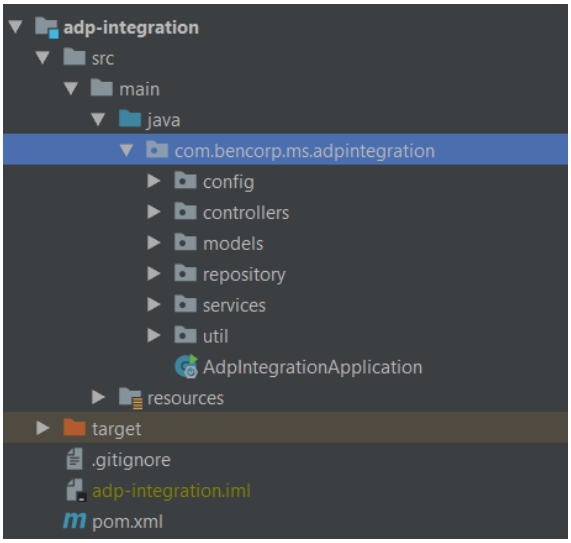


Figure 3. Backend module's folder structure

Following rules of the Maven projects organization, the source code and the static resources of the project should be stored in a src/main folder which contains the code and resources in separate folders. The name of a folder with source code is also following conventional Maven agreement for packages naming. In fact, each piece of the name separated by dots represents a

subfolder, the IDE visually compresses them for better perception of the file system. (Apache Software Foundation 2019c). Top level of the source code's folder contains main class, which points to the program entrance point and may contain configurations for containers and available open connections. The configuration folder contains Spring Security configuration class and supplementary classes required for the application's protection. The controllers represent logic of the backend API and the application's business logic itself; the models contain data classes which are used for representation of database tables and their properties. The repository folder contains interfaces which are used for working with JDBC which mostly include standard SQL operations like SELECT, INSERT etc. The services folder contains classes which are used as collections of methods used across different classes in the application. The util directory contains secondary auxiliary classes which are used for specific cases and cannot be associated with the exact part of application logic.

The frontend module adp-ui is organized a lot like adp-integration module. However, it includes much more configuration files in a root level. Most of them were not utilized during this thesis implementation as the majority of those files contain specific configurations for cases of work with browser or testing which were not considered as the priority problems on a stage of development under this work's scope. Figure 4 represents the structure of adp-ui module and shows main conceptual parts of application's UI backbone.

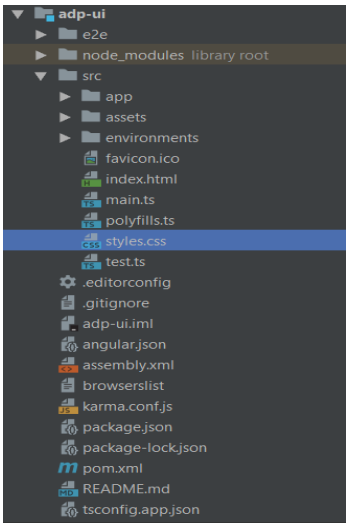


Figure 4. Frontend module's folder structure

The folder called e2e contains end-to-end tests aimed to check if the behaviour of the application corresponds to the desired after a physical interaction from the user. However, the tests have not been implemented in the project in th development stage of the application under this work's scope, therefore they will not be considered further. The node_modules folder contains all the dependenices fetched from the central repository of NodeJS required by the application. The directory named app contains all the source code related to user intefrace's implementation alongside with the root web-page which serves as an entry point for application's website and static resources for application such as images. Root derectory also contains a POM file which stores settings for Maven module's deployment. The left files in root directory contain secondary configurations related to IDE settings, list of supported browsers, unit tests configuration and so on. Those files have been left untouched after the directory's structure bootstrap made by Angular CLI and all the settings were left equal to the defaults.

Going deeper in the description of the frontend half of the project, it is importnat to mention the organization of the suource files. The structure can be viewed in a Figure 5. Every component of the Angular applciation is stored in a separate folder, except the root component.

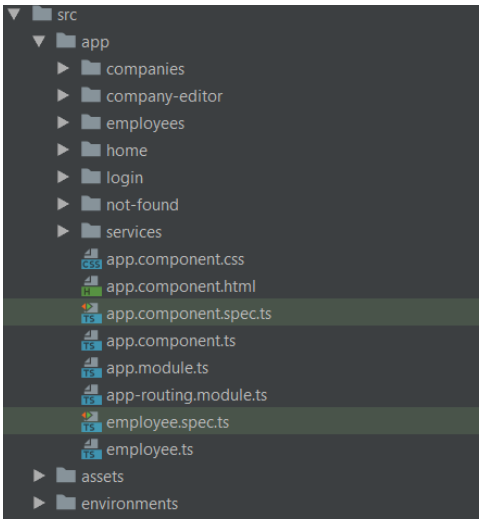


Figure 5. Frontend source code folder structure

IntelliJ IDEA also provides integrated plugin for working with Maven projects which eliminates need in the terminal usage and allows performing all the operations with a couple of mouse clicks. The plugin provides control panel where all the operations available for modules can be done and the core information about modules can be monitored. The tab can be seen in a Figure 6.

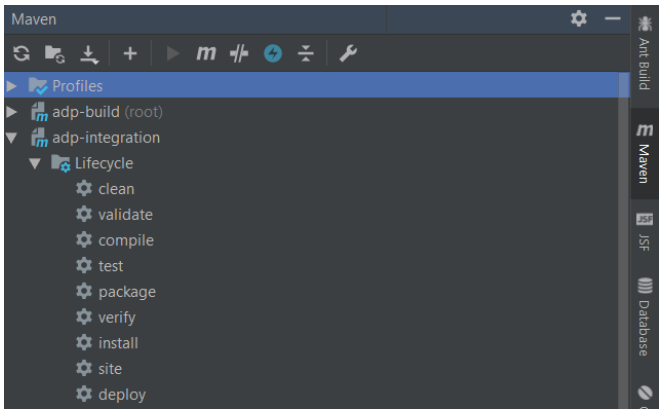


Figure 6. Maven control tab in IntelliJ IDEA

The main case to use the control bar was verifying an ability of the project to build. It was required in order to save time which could be taken by reuploading of the project to server due to the unsuccessful installation.

3.2 Backend of the application

Now it is time to talk about the functionality and the architecture of the project's backend part. The description follows the same way as the Spring project chronologically went in the development, e.g. organized in priority for the entire application and in order implementation of the features.

3.2.1 Connection to ADP platform

The first thing that was implemented in the project and the thing which initiates the whole cycle of a typical working session with the application was constructing the functionality of the entry points of the website. There the user from ADP marketplace is redirected after launching the app in his marketplace home page. The first entry point is used immediately after the app has been started. This entry point provides a redirect link to an ADP authorization module. After verifying

the user's credentials, the second entry point goes into action and continues the application flow.

Before the creation of the connection an SSL certificate signed by ADP is required. The certificates are requested through a special app in the marketplace, where the applications' owners send requests to get connection allowance. When the certificate is received, it is supposed to be stored somewhere in the application's directory as every request to ADP's resources must be accompanied by that certificate. In case of this project, the certificate was stored in the *resources* directory. (ADP Inc. 2019b.)

In order to organize the connection to the platform and an access to internal resources, ADP provides SDKs for different programming languages, including Java. The SDK which was used in this project was supplied using Maven repository, as the kit is also published there as a package. The connection can be configured manually using any approach. However, the SDK provides handy interfaces and useful classes out of the box, which eliminates the need for excessive work. (ADP Inc. 2019e).

The URL endpoints in the Spring Boot framework are introduced as methods inside the controller classes. These classes are specially annotated for the framework using `@Controller` decorator. Methods responsible for handling requests are marked with `@RequestMapping` annotation supplied with parameters related to the request's type (GET, POST, UPDATE etc.) and the URL which the given method corresponds to. This lets the application know that all the requests which are aimed at the URL matching the method's annotation parameter should be handled by this method. The parameters of the method include two variables of types `HttpServletRequest` and `HttpServletResponse`. These are built in the classes of JavaX Servlet API, which are responsible for handling incoming request and providing proper response to it. An example of the configured endpoint can be seen in Figure 7.

```
@RequestMapping(value = "/adp/login", method = RequestMethod.GET)
public void login(HttpServletRequest request,
                  HttpServletResponse response) throws ConnectionException, IOException {

    String redirect = adpoAuthService.getAuthRedirect(request);

    response.sendRedirect(redirect);
}
```

Figure 7. Example of URL endpoint handler

The endpoint described above serves as the first entry point of the application. The incoming request from the ADP's user is processed by a member variable of the controller class. The variable is represented by a separate class defined as a service in the program using a `@Service` annotation. The service initializes the creation of the connection's configuration included in marketplace's SDK and returns an authorization URL defined in this configuration. Alongside with this process, the first security mechanism steps in. The initial requrest is supplied with a special state number which is saved by the application. This is done in order to verify that the request which will come into the application after credentials verification is coming from the same source as the initial request.

One thing to mention here about Spring Boot is a feature that allows class instances to automatically initialize after declaration. With the class members `@Autowired` annotation can be used, which lets the program know that the instance of the class should be created.

When a redirect from the application is received by the user, he or she is prompted to enter his or her credentials in ADP's authorization point. In case of the first application's launch the user is also asked to provide data access consent. According to the fact that the application which is described in this work represents a composition of an end-user application and a data connector application, the only consent for user information sharing is asked, the data access consent counts as accepted by default.

In case the valid credentials are provided, the user is redirected back to the application website using a second entry point – callback, according to ADP's terminology (2019b). The relative address of the entry point is /adp/callback. The

redirect link also contains information for the application which includes authorization code and a state number from the initial request. The authorization code is further exchanged for an OAuth 2.0 authorization JWT token which is used to make a request for the user's information. After the token is received, the application verifies the token as a second tier of security in this scope. The token gets decoded by the application and such information as token issuer and intended recipient are checked.

After the token's validation a request to ADP's website is performed in order to obtain the user's information. The information received from the request is then saved to the database using a service which invokes a user's repository interface. The users who log in following the described approach are saved in the database with a special mark indicating that the user utilized SSO in order to login to application, and the ADP issued credentials were provided for the authorization using Hibernate framework. The user's information fields which are retrieved from the ADP system include the id of organization which the user belongs to, his internal id in that organization, an email and a full name.

When the user's information is received, the information required for the data connector part of the application is retrieved from the ADP's system. In order to get an access to the information needed the subscription credentials assigned to the user in the scope of the application are required. These are not the same credentials the user uses to login in the application. They only provide the ability for the application to get access tokens from ADP and their further utilization. These credentials are provided to the partner organization automatically alongside with the consent for the data connector app. Actually, they can be retrieved at any moment. However, to ensure that the consumer credentials are passed to the local database of the application, their transfer is tightened with the user's login event. The mechanism of retrieving information is not aimed at the only credentials which are associated with the current user. Instead, the request to the storage of all the available credentials for exact application is requested at a time and the database is updated. In case the credentials of some organization(s) already had been transferred in the local storage, nothing is done

and the scanning of the list of credentials goes further. The code which stands behind the subscription credentials retrieval mechanism can be reviewed in Figure 8.

```
private void getDataConnectorCredentials () throws ConnectionException, IOException, URISyntaxException,
        OAuthCommunicationException, OAuthExpectationFailedException, OAuthMessageSignerException {
    //check credentials
    String crJson = IOUtils.toString(connectionService.doPost(connectionService.getSubscriptionCredentialsApiPath(),
            JSONUtil.toJSON(connectionService.prepareCredentialRead()), ConnectionService.AuthType.oAuth_2_0));
    System.out.println("CredentialsRead");
    System.out.println(JSONUtil.normalizeJson(crJson));
    ConsumerApplicationSubscriptionCredentialsReadEvent cr = JSONUtil.toObject(crJson,
            ConsumerApplicationSubscriptionCredentialsReadEvent.class);
    if (cr != null){
        for(Event event : cr.getEvents()){
            for(ConsumerApplicationSubscriptionCredential cred : event.getData().getOutput().
                    getConsumerApplicationSubscriptionCredentials()){
                ADPCredential adpCred = adpCredentialsRepository.findBySubscriberOrganizationOID(cred.getSubscriberOrganizationOID());
                if (adpCred == null){
                    adpCred = new ADPCredential();
                } else {
                    return;
                }
                adpCred.setClientID(cred.getClientID());
                adpCred.setClientSecret(cred.getClientSecret());
                adpCred.setSubscriberOrganizationOID(cred.getSubscriberOrganizationOID());
                adpCredentialsRepository.save(adpCred);
            }
        }
    }
}
```

Figure 8. Subscription credentials retrieval mechanism

Once the subscription credentials have been fetched, the user is further redirected to the home page of the application. The functionality available for users inside the application is covered later in this work.

The process of saving the information loaded from the ADP's platform in the database has one noticeable feature. As all the interactions with the database directly from the code are performed using a Hibernate framework, the information passed to the database or retrieved from there is represented by class instances. However, all the information returned in response to the requests to the ADP's APIs is contained in a form of JSON object.

In order to eliminate the incompatibility of these data types, a dedicated JSON utility is used. The utility class provides methods which allows direct casting of the JSON object into an instance of the selected class. With a help of this auxiliary tool the response received from the ADP's system can be directly assigned to a variable with the required type and further handed over for database processing. The classes which have a direct relation to the data types

returned from ADP implement additional annotations for the class members. The annotations map members of the program's class with the members of JSON objects so that during the process of a conversion there would not be uncertainties in data targets. The annotations for class members are done using a `@JsonProperty("name_of_JSON_member")` decorator. The utilization of the utility is applied not only to the user's information retrieval process, but to all the data received from the platform.

3.2.2 Events handlers

Going further in understanding of the backend implementation, the logic of handling the subscription and user events described in the section 2.4.2 of this work. This part of the application's functionality does not have a strong influence on the workflow of the application, only the exact parts of this module are extremely needed for data integrity. However, since all the applications should follow the rules proposed by the ADP and there is no other way in order to stay published in the marketplace, the entire module of events handling was implemented in this thesis project.

According to the requirements of ADP's marketplace (2019), the separate endpoints inside the application should be provided for handling requests related to the events. In the domain of the application which is described the endpoints for events has relative address /adp/subscription_event and /adp/user_event accordingly. Both endpoints are accumulative and are capable of handling all the types of the events within their area of responsibility. The events are emitted using POST requests (AppDirect Inc. 2019) and the exact type of an event can be fetched from a payload of the request.

Before the processing of the event's information begins, the event is first verified using different tiers of inspection. Failure at any step of the inspection leads to dropping down the incoming request. First of all, the application checks whether the incoming event was sent by an authorized trusted source. For this purpose, the OAuth 1.0 authentication mechanism between the marketplace and the application is implemented. Comparing to a token approach of OAuth 2.0, the

OAuth 1.0 standard uses simplified authentication mechanism which relies on basic username and password check (OAuth Core Workgroup 2007). Both the application and the platform sign their request with these credentials. In case the valid credentials were provided by a sender of the request, the next tier of review starts. The URL of the event is taken from the request. In fact, the initial request from the ADP's system does not contain the full information about the event. Instead of it, the request is supplied with a URL using which the data related to the event can be fetched. In case there is no URL provided, the request is dropped. In the opposite case a request for event information is made from the application to the platform. This step also invoked casting the JSON response into a class instance. In this case it is not done for compatibility with the database interface, but for ability to work with AppDirect SDK. If the received response is empty and does not contain the information requested, the flow stops, and the initial request is dropped. Following the branch of success, the last phase of initial request analysis is made. This step includes actual fetching of a type of the event. Only the requests with the supported events types are allowed for further processing. (Figure 9).

```
//Verify signed request
System.out.println("Validating request");
if(!eventsService.verifyRequest(request)){
    eventsService.respondeError(response, ErrorCode.UNKNOWN_ERROR, message: "Signature is not valid.");
    return;
}

//read request

//check eventUrl is exist
String eventUrl = request.getParameter( s: "eventUrl");
if (StringUtils.isBlank(eventUrl)) {
    eventsService.respondeError(response, ErrorCode.OPERATION_CANCELLED, message: "eventUrl parameter not provided");
    return;
}
//get eventUrl data
String seJson = IOUtils.toString(connectionService.doGet(eventUrl, ConnectionService.AuthType.oAuth_1_0));
System.out.println("SubscriptionEvent");
System.out.println(JSONUtil.normalizeJson(seJson));
SubscriptionEvent se = JSONUtil.toObject(seJson, SubscriptionEvent.class);
if (se == null){
    eventsService.respondeError(response, ErrorCode.OPERATION_CANCELLED, message: "Not able to acquire event information");
    return;
}
ADPFlag adpFlag = ADPFlag.possibleValueOf(se.getFlag());
ADPSubscriptionType adpSubscriptionType = ADPSubscriptionType.possibleValueOf(se.getType());

if (adpSubscriptionType == null){
    eventsService.respondeError(response, ErrorCode.OPERATION_CANCELLED, message: "Undefined subscription type");
    return;
}
System.out.println("Subscription Type: " + adpSubscriptionType.name());
```

Figure 9. Events verification code

Further logic of events' processing relies on the exact type of the event. In case the subscription's type was changed or cancelled at all, a corresponding status of the subscription is set in the required client's row in the database. In case the event with an unsupported type passes the initial inspection, an error stating that provided event type is not supported and the request is dropped. If the event's type is the subscription creation, then some information is acquired from the ADP's website. For some reasons, the information about the client and the client's company is only available through the subscription creation event. Therefore, this information about these things is fetched alongside with the subscription creation event handling. The data includes basic contact information required for client's identification such as a client's address, a contact phone, a contact person, the name of the organization related to the client etc. However, not all the information required can be received during the described process and, actually, not anywhere from the ADP's storage. Some pieces of the information require manual entering, which is covered further in this work. All the information is then passed to the database, separately for the client and his or her company. In case the entry with the same client ID already exists in the database, the entry is just updated without creation of a new row. This situation can happen in case when company resubscribes to the application as the information about the customers is not deleted from the database once the subscription is canceled. The visual model of the event's flow can be seen in Figure 10 which illustrates the general pattern of the entire event handling process.

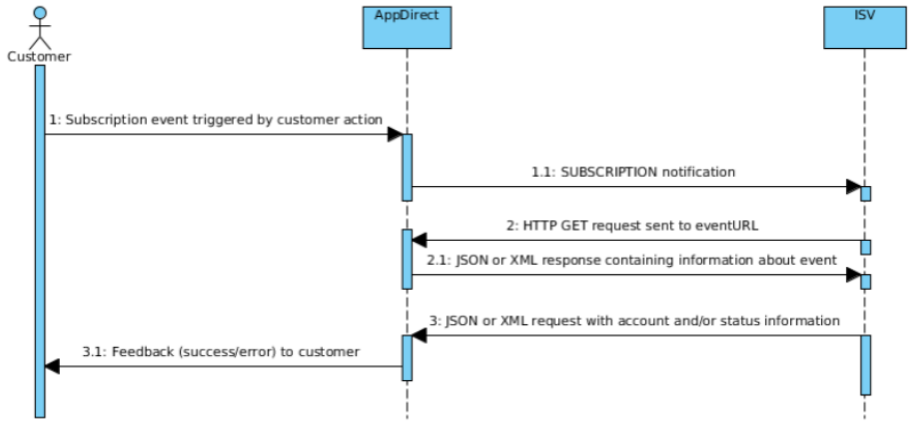


Figure 10. Subscription event notification flow (AppDirect Inc. 2019)

Nearly same workflow applies to the user events. First step in the processing is the event validation which is identical to the same process in case with subscription events. The same parameters of the incoming request are checked, and the same inspection logic is applied to the user events. After the analysis tier of the event's processing is over, the further actions depend on the type of the event. However, in case with the user events the logic is quit alike in case of both possible events. If the user assignment event is received, then the application checks the database for presence of this user. In case the user is found, the application responds with an error and drops the request. If no user with a matching user ID was found, then a new entry in the database is created. In case with a user unassignment event the logic works in the opposite way. If the user requested to be removed from the application was found in the database, the corresponding row is removed. If no entry was found, then the error message is responded. The visual model of the subscription's flow can be seen in Figure 10 which illustrates the flow of subscription event.

Summing up this section and the previous one, the mechanisms required for receiving of the user's information and the client's and company's information were discussed. The combination of these three is enough for proceeding further with the information related to workers in particular. Further description focuses on the application's created REST API for working the frontend part of the project. Previously discussed things have direct relation to the interactions between the application and the ADP platform with the initiation of the process from the ADP, whilst following materials cover cases of events which appear during user's activity in the application.

3.2.3 REST API of the application

As the application's backend and frontend parts exist in different web server's containers, a REST API inside the backend was created in order to provide access to the information stored in the database. The prospective future ways to utilize the API may also include request for the database's information from different sources. However, the version which was created under the scope of this work supposed information provision to the only destination.

The application's infrastructure allows viewing of the content stored in the database and editing the content for the certain types of the data. Depending on the section of the application the appropriate API is invoked. All the API endpoints are stored in the separate controllers differing by a type of the information. In the context of the application, for the needs of the user interface there are controllers for companies, users (utilized only by the application's security and is covered in a corresponding section), employees of the companies and the separate controller responsible for the creation of the XLSX files.

The employee's controller exposes the only one endpoint for the API. A request to this endpoint returns a list of all the employees for the selected company fetched from the database. The ID of the required company is passed with the request as the standard HTTP request parameter. However, the controller is also responsible for loading of the employees from the ADP's storage at the same time. When the employees' page if the application is opened, the automatic call to the employees' API is made which first launches a request to the ADP's Workers v2 REST API endpoint. This is maid in order to keep the information about the workers relevant all the time. After the information received from the ADP's system, the data is first passed to the database and then from the database to the user interface. Following this workflow, the information about the workers displayed in the frontend always corresponds to the latest snapshot of the available data.

The controller for the creation of MS Excel's files bases his work on the employees' controller partly. The logic of the controller only supposes fetching the data from the local database for further processing, which means it is not able to update the information in the database by itself. However, the button responsible for invocation of the described controller is located on the same page which invokes the employee's API. Therefore, when the API of the file creating controller is invoked the information is the database is always up to date (in case the API was invoked as intended using button in the interface). The processing of the data taken from the database for the proper file's construction is covered in a separate section of this work.

```
@RequestMapping(value = "/adp/companies/{oid}", method = RequestMethod.GET)
public Company returnData(HttpServletRequest request,
                          HttpServletResponse response, @PathVariable("oid") String oid) throws IOException {

    User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    System.out.println("Current user: " + user);
    if (user.getOrganizationOID().equals("G2ES6A5P4F47284E") || user.getOrganizationOID().equals(oid) || user.getUsername() != null) {
        return companyService.findByOid(oid);
    }
    return null;
}
```

Figure 11. Example of the application's API endpoint

The companies' controller exposes the biggest amount of the endpoints among all the controllers. The API allows retrieving the information about all the companies, about the particular company and also editing the information of the particular company. Before the information is either provided or altered, the application checks whether the user is righteous for these actions. Following this approach, the endpoint responsible for providing the information about all the companies checks if the user who requests this information belongs to BenefitScape (either local user or logged in with ADP's SSO); the endpoint which provides information about the particular company inherits the same criteria of the assessment with only difference that the users who belong to the requested company are also allowed to see the information. Same logic applies to the endpoint responsible for editing. Figure 11 illustrates an example of the API's endpoint, the one responsible for returning information about one company in particular. As this work's aim was not only creation of the final product, but also studying the technologies involved in the development, the different techniques were used in the API. For instance, the API responsible for workers' information receives the ID of the organization using HTTP parameter, whilst the companies' API makes it in a REST manner including parameter as the part of the endpoint's path.

3.2.4 Data processing and formatting

The last detail about the backend of the application in general terms is the creation process of the files which are further submitted to the IRS. As the previous section explained, the generation of the files and prompt for their

download happens when the user presses the corresponding button on the employee's page. Once this is done, the dedicated API endpoint in the backend goes into use and the flow begins.

The process is divided into two sections. The first one is responsible for providing the logic for the creation of file where the information is contained and making it possible for downloading. The second part is responsible for the fulfilling of the file with the content.

Once the request for the file is received, the endpoint responsible for handling the request to the required URL starts setting up the proper response. Basically, the idea of providing the file for download is simple. First of all, it is required to write proper headers for the response so that the browser knows about the content of the incoming response and the way it should handle it. After that, the required file is returned as a stream to the browser and the user can download it to his or her local machine.

The configuration of the response in this case is pretty simple. The only headers required are the type of the returned file and the name for this file. For MS Excel file the special content type `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet` is used (Microsoft Inc. 2008). When set, the browser knows that it receives the MS Excel 2007+ file and there will be no conflicts of the actual file type and the resolution set in the name. The name of the file in terms of this application is constructed using a base prefix and the timestamp of the moment when the file is created. In this case it becomes easier to distinguish files related to the different content versions. The pattern of the name for files corresponds to the following template: employee_data_YYYYMMDDHHmmSS, where Ys stand for a year, Ms stand for a month, D stands for a day, Hs stands for an hour, ms stands for a minute and Ss stands for a second, when the file was created.

After both of these pieces of the configuration were applied, the file is ready for streaming to the browser. However, before that it is required to fill the file with the

information about the employees. A method contained in a dedicated service class for the file creation process is used for that. The method takes the ID of the organization as the parameter in order to identify which workers should be taken from the database.

First of all, the function creates an Excel workbook out of a pre-filled template file, which is used in order to minimize the amount of code for file creation. In order to process Excel files in Java code the Apache Software Foundation has created a special API for working with MS Office documents, including Excel documents of both old and modern types. The library is called Apache POI. The library allows reading or editing existing Excel documents or creating the new ones. All the operations available through the graphical user interface of the MS Excel can be done using this API. (Apache Software Foundation 2019d.)

When the workbook's variable is assigned with the contents of the template, the required sheet in the workbook is selected fur further processing and assigned to a separate variable. The template file is originally supposed for filling in three categories of the information: the relevant employees of a company, the retired employees or the ones who left and the dependents (spouse, child(ren)) of the relevant employees. However, due to the fact that the RESP API for working with the dependents was not released by ADP when this work was done, the sheet related to the dependents was not processed by the application. The sheet which contains information about non-relevant employees was also left untouched due to lack of time in the project creation. So far, the only sheet which was taken for processing was the relevant employees' sheet.

The selected workbook's sheet is then filled in with the information about the selected company's employees. An array of all the workers is processed, and for each worker a new row in the sheet is created. Every row is further filled in with the information about the selected worker. All the cells are created manually, and every cell has different processing logic. This is done in order to correspond the cells with the columns created in the template file, and also due to the fact that the values in some cells require pre-calculation on the basis of the given data.

For example, the library does not support an automatic conversion of the Date format variable into a classic representation for humans. This can only be reached by styling the cell with a proper format. Figure 12 illustrates the creation of the row and several cells with the different data types in it. The cells with no assigned values belong to a part of the information which could not be obtained from any place at the moment of the project creation.

```
for (Employee empl: employees) {
    XSSFRow row = sheet.createRow(rowNum++);
    Company company = companyService.findByOid(empl.getCompanyOID());
    //Column A matches index 0
    //Will require reorganization if template changes
    row.createCell( columnIndex: 0).setCellValue(company.getEmployerEIN());
    row.createCell( columnIndex: 1).setCellValue(empl.getExternalId());
    row.createCell( columnIndex: 2);
    row.createCell( columnIndex: 3).setCellValue(empl.getSsnOrTin());
    row.createCell( columnIndex: 4).setCellValue(empl.getGivenName());
    row.createCell( columnIndex: 5).setCellValue(empl.getMiddleName());
    row.createCell( columnIndex: 6).setCellValue(empl.getFamilyName());
    row.createCell( columnIndex: 7);
    row.createCell( columnIndex: 8).setCellValue(empl.getEmail());
    Cell hireDate =  row.createCell( columnIndex: 9);
    hireDate.setCellValue(empl.getHireDate());
    hireDate.setCellStyle(dateStyle);
```

Figure 12. Creation of rows and cells in outcome sheet

When the process of filling in data is over, the workbook stored in a variable is saved to a file which is further converted to an input stream and returned to the controller described in the first part of this section. At this point, the application produces its final outcome and reaches the initially set goal.

3.3 Frontend of the application

The development of application's user interface was started as soon as the integration of the SSO mechanism was finished. With the addition of the new website's section in the UI the new functionality in the backend was added. Following this approach, every step in the creation of the website's infrastructure had a fully functional basis behind and no "dead" pages existed there. The description below covers the overall structure and principles of the application's UI and the exact parts in particular.

3.3.1 General structure

As described in the section 3.1, the general structure of the Angular's project consists of several root files and the components for every separate section of the application. Regarding the functional side of these things there is not so much to add. However, the conceptual design for the whole infrastructure needs introduction.

Because the application is built following the pattern of the single page application, all the content bases on one root HTML page. That HTML page contains the very basic information about the application, such as encoding for the page, the name of the tab in a browser and so on. However, it also declares the root component of the application which is responsible for handling all the user interface. The navigation tab and the footer containing copyright information are declared in this exact component. This is done because the navigation tab and the footer are common for all the sections in the applications. Talking about the footer it is not the most important thing from the angle of the functionality, and its sharing is not that vital for an infrastructure to be comfortable and understandable. However, it is required in order to protect the design and the contents from stealing.

The more important thing is the navigation bar. This element of the web page allows switching between the sections of the website, and without it would be impossible to travel inside the application without direct entering of the corresponding URLs. Declaring the navigation for every component would be irrational, since that it is created in the root container. Figure 13 illustrates the template of every page in the application. All the content sections of the website are contained inside the template, excluding the login screen. Non-authorized users are not able to see the functional navigation bar.

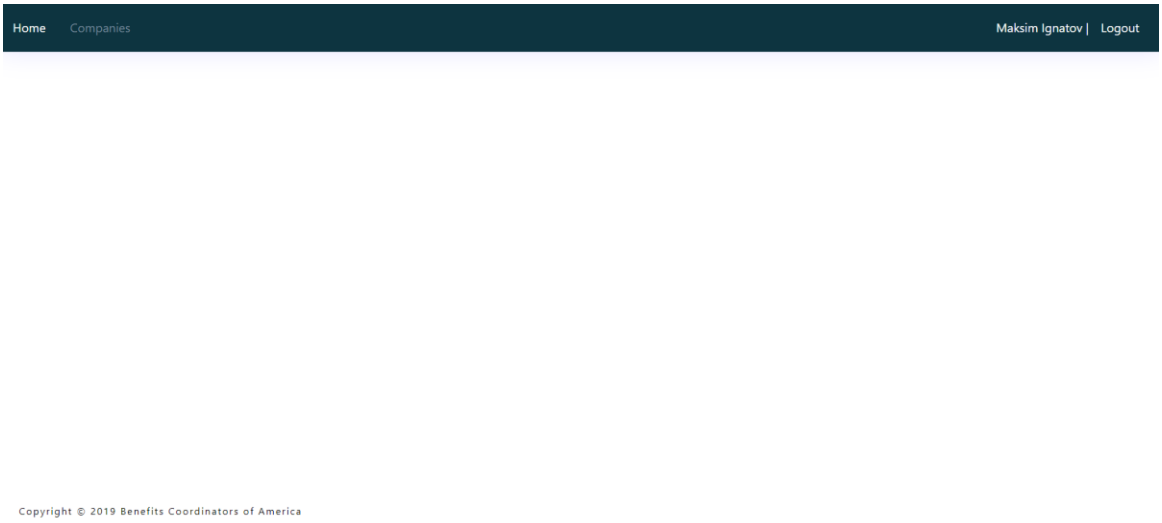


Figure 13. Root container of the website's views

The navigation bar contains the corresponding tab for each section of the application and displays the information about a currently logged in user. Also, the link for logging out of the application is provided.

3.3.2 Application views and their logic

All the sections of the website are created using separate components. Most of them have similar designs from both the angles of visual perception and the code's main features and main HTML components. Therefore, this section covers the example of one of the components from the perspectives of design and code. The considered component is the one responsible for possibility of editing information related to a chosen company. The component was chosen because its functionality includes data transfers in both directions – from database and into it.

Every component which belongs to a certain module should have its class name in the module's annotation. The declarations are made inside the decorator of the module (Figure 14). The component which is described is stated there as a `CompanyEditorComponent`. The decorator also contains the information about imported modules for an application and the service providers (services) defining their scope of being unique.

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    CompaniesComponent,
    NotFoundComponent,
    CompanyEditorComponent,
    LoginComponent,
    EmployeesComponent,
  ],
…
})
```

Figure 14. Declarations of the components inside of a module

The component's Typescript class always contains as constructor method which implements all the services required by this component in order to make utilization of methods contained in the services possible. The continuation of component's workflow goes with the specified interfaces. The components may implement different interfaces responsible for one or another step in the component's work cycle in order to bundle them with the specified action. The component which is described in the example implements the `OnInit` interface which is related to actions which happen after the component's full initialization.

As the company's editor screen allows the only one company's processing, the first thing which is done once the component is initialized is obtaining of the company's id and then make a request to the database in order to obtain currently assigned settings. The initialization logic of different components can vary depending on the required actions. Additional methods can also be created inside the component alongside with the functions required by the user interface's elements. For instance, the component described contains one method for retrieving of the information from the Spring server and one more method which is responsible for handling the submission of the information towards the database.

The methods mentioned above utilize the service which his used nearly in every section of the application - `HttpClient`. The service is responsible for handling

the HTTP requests in both directions. All the requests are made with use of observables in order to provide the ability to have asynchronous flows in the application. In fact, in most cases the observable returned by the service are not assigned to the variables. The results are processed further using the ‘next’ handler inside the subscription (Figure 15).

```
onSubmit(form: NgForm) {
  this.http.post( url: '/adp/companies_edit', this.selectedCompany).
  subscribe( next: (res: string) => {});
  this.router.navigate( commands: ['/companies']);
}
```

Figure 15. Example of HttpClient service’s invokation

The views of the components are represented with standard HTML pages with some additions from the Angular. Using the tools provided by the framework, it becomes possible to introduce the data binding inside these HTML pages in order to create dynamic content. Besides that, Angular also gives opportunity to use typical programming structures such as loops and conditions inside the pages.

The design of the views is done in a laconic way using the jumbotron container of the CSS Bootstrap library. The library is included in the project as one of the dependencies extracted from NodeJS platform and is saved in the application’s file system. Comparing this approach to the traditional ways of including the Bootstrap library (either with styles link or direct saving), this approach provides more efficient way of network’s resources utilization and can be easily updated if needed. The containers further display tables which contain the desired information and provide available functionality depending on the context of the view. The design is consistent throughout the website with the only exclusions for login page and company’s editor page. These sections do not contain tables inside the components. The design of the elements on the web pages can be seen in Figure 16 which represents a company’s editor view. As it was said in the previous section, the root elements of the page (the top bar and the footer) are fixed in their positions, and it can be seen with an overflow of the footer above the scrollable content. All the information inside the fields is taken from the local

variable which represents a model in this context. The model and the view are tightened with a two-way binding. This means that changes on one of these are immediately reflected in the second. Talking about the described case in particular, this feature is utilized in the process of form's submission. When the event is triggered, the application does not send an HTML form to the server. Instead, it sends the model which stands behind the view as it already contains all the changes made by the user. This is done with help of Angular as it allows assignment of the functions related to the submission event instead of pointing the webpage responsible for handling the form's processing.

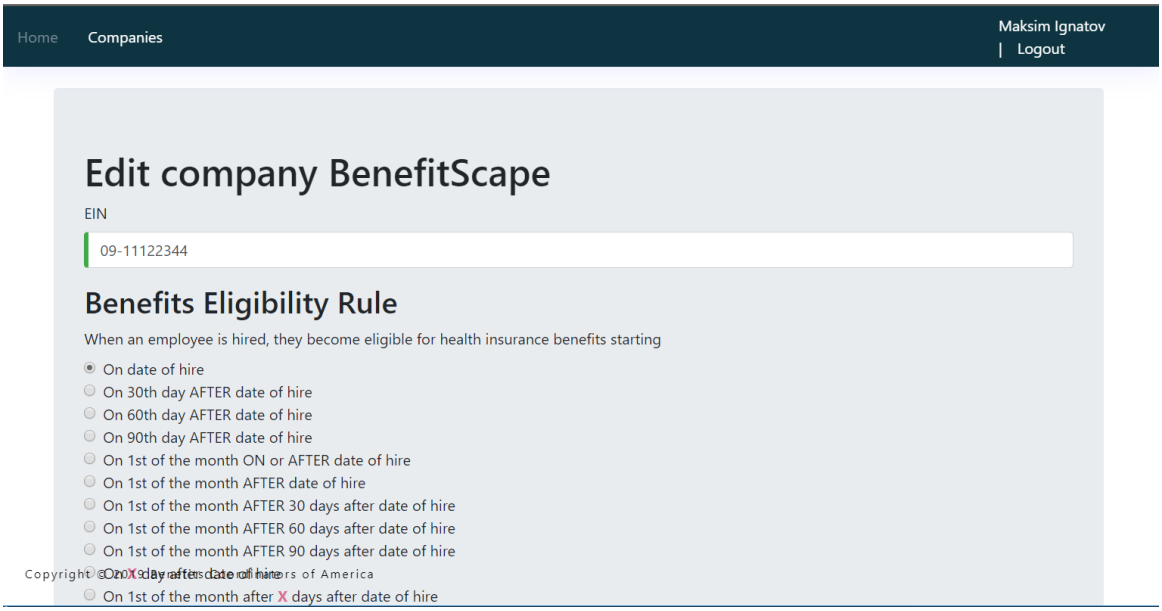


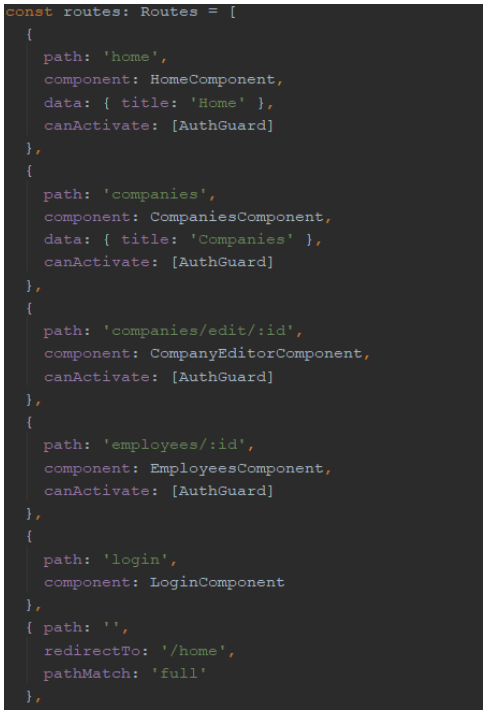
Figure 16. Application's view

So far, all the building blocks of the frontend responsbile for providing the visual result have been covered. The last section describes the routing configuration on the website and the implementation of this module in the application.

3.3.3 Routing

The navigation's functionality inside the application is provided with use of a built-in Angular's routing module – `AppRoutingModule`. Implementation of the module allows utilization of the single-page applications in a manner of the traditional web applications with several separate HTML pages. In order to

configure the routing in the application, the file of routing module should be added to the project's file system and declared inside the application root module.



```
const routes: Routes = [
  {
    path: 'home',
    component: HomeComponent,
    data: { title: 'Home' },
    canActivate: [AuthGuard]
  },
  {
    path: 'companies',
    component: CompaniesComponent,
    data: { title: 'Companies' },
    canActivate: [AuthGuard]
  },
  {
    path: 'companies/edit/:id',
    component: CompanyEditorComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'employees/:id',
    component: EmployeesComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'login',
    component: LoginComponent
  },
  { path: '',
    redirectTo: '/home',
    pathMatch: 'full'
  },
```

Figure 17. Array of applciation's routes

The configuration of the routes inside the module represents a boundling of the route's pathes (relative URL addresses) and the components associated with these pathes. The configuration also contains the supporting services for the selected endpoints which. However, due to the fact that the services implemented in the application are only realted to the security aspects, they are covered in the secuirty section of the implementation's chapter. The relations between the pathes and the components are setup as an array of type `Routes` which can be seen on Figure 17. The array contains mappinf for all the available endpoints, including the separate mapping for cases when "404 Not found" error is returned by the application.

3.4 Security

Once the functional aspects of the application were implemented and the fully working state was achieved, the security aspects of the system were considered. As the application is supposed to work with the confidential data about the client's employees, the unwanted access to that information had to be prevented. Not

only the access of unauthorized users was considered in the planning of security's implementation was considered, but also adding the ability to account the logged in user's action was one of the key issues.

The main mechanism of the security's implementation is contained in the backend of the application. However, the frontend part of the project also includes its security features alongside with the tools for working with the backend's solutions. The main concept which was followed during the implementation of security was aimed to blocking access to information for non-authorized users and validation of already logged in user's authorization.

The Spring Security was utilized mainly in order to provide secure access to the application's content. Previous sections described local security mechanisms which use unique assessment criteria depending on the place where it was implemented. Therefore, this section is focused on general security of the application regardless the website's running process. The first thing which is required in order to make the module work and which was made on the initial stage of the project's creation was including the Spring Security module as the part of the project in Maven dependencies. After accomplishment of this step all the built-in classes responsible for implementation of security and methods of these classes become available for utilization in the project. In case of the given project, the implementation of `WebSecurityConfigurerAdapter` class was used.

The next step of the security's implementation was listing of the endpoint's availability patterns and configuring behavior of the application after the events related to user's logging in and out appear. These settings are specified in the overridden `configure` method of security class. The configuration was set using methods' chaining in a conventional manner of the Spring Security (Pivotal Software Inc. 2019c). Figure 17 shows the full configuration of security module and describes the actions which application should do depending on the requested resource.

```
@Override
protected void configure(final HttpSecurity http) throws Exception {
    http.csrf().disable() HttpSecurity
            .authorizeRequests() ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/adp/login", "/login", "/adp/user_authorize", "/adp/callback").permitAll()
            .anyRequest().authenticated() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry
            .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
            .logoutRequestMatcher(
                    new AntPathRequestMatcher( pattern: "/adp/logout")
            ).invalidateHttpSession(true) LogoutConfigurer<HttpSecurity>
            .deleteCookies("JSESSIONID");
}
```

Figure 18. Configuration of Spring Security

As Figure 18 shows, the only endpoints exposed for publix acces are the ones which are used as the netry points by the ADP's system (local security mechanisms were implemented there) and the endpoints which are used by users or the application in order to initiate authorization process. All other requests require active authentication in order to reach the resources. The configuration also provides the alorithm for handling the loggin out process. According to the code (Figure 17), the dedicated API endpoint for logging out process is assigned and post-process actions are specified. These actions inlcude markign the newly closed user's session as invalid and also wiping out he information related to the session. This is done in order to prevent utilization of the closed session in situations when this information can be stolen and used without the original user being aware of it.

The other important step in implementation of the security was providing logic for authorization's assessment. Spring Security provides basic mechanism for holding the user's session on the basis of username and password. However, due to the fact the the application utilizes different credentials for user's authorization provided by SSO mechanism, the custom tools should have been implemented. Security context of the application – the actual storage of the sessions' inforamtion, bases it work on intercation with the implementations of the abstract classes which represent different auxilary tools for authnetication. Depending on the needs of the required logic varoius implementations of the classes can be created in order to fit the requirements.

The holders for authorization in the Spring Security are represented as special token classes. These are used in order to provide the organization of user's information storing and the ways to gain this information. The application implemented the token class using the JWT id token provided by ADP marketplace duting the SSO process as the credentials for user and the instance of user's class as the principal in the context of session. The token is added to security context during the final stage of SSO process right before the data connectore credentials are obtained. Once the token is set, the application will provide access to the information ofr authorized user. The token is kept alive during the whole period os the session's validity and can be accessed at any moment of time through the security context. Without a valid token the request to any protected endpoint of the API will end up with an "503 Forbidden" error.

The frontend of the application implements the security using the authentication service guard. The special injectable service class was created for that purpose and assigned to every required route. Woeking process of the guard relies on application's backend. Every time the user tries to access the protected route, the guard makes a request to the Spring server in order to check if the current session is authorized. Alongside with that, the fronted recieves the information about the user to display in the navigation bar. The request to the security endpoint is assigned to an abservable varaible, which allows detection of session's state changes in real time. In case the user does not pass the authentication process, he or she will be redirected to the login page in every unauthorized request unless the successfu logging in process is done.

The navigation bar which is common for every view in the application is also adapted to consider the authentication state. None of the elements inside it are displayed unless the authorization process is accomplished. This was done using th Angular's ability to add conditional structures inside the HTML code.

However, the original security module's requirements were not fully implemented. Due to the combination of several factors, the authorization for local users was not implemented in the project and the only avaialble option for authorization in

the application was SSO mechanism based on the ADP's issued credentials. Even though this feature was not implemented, the applciation supposed separate logic for company's users even with the SSO authorization.

The implementation of the security in the application was the last part in the application's functionality realization. After it was over the application was ready for final testing and "cosmetic" debugging. The latest versions were uploaded to GitHub repository and to the live server through Jenkins. The application was insepcted in order to see that every step in the supposed working cycle does not cause any troubles on server and the responses correspond to the supposd values. Minor imporvements according to the commissioner's requirements were made in the UI. At this point the development process was over.

4 CONCLUSIONS

The original problem of this thesis was the automation of data extract from third-party services for the commissioner's needs. Therefore, the final goal of the work was creation of the software which would have automatic data gathering from the selected platform and post-processing of this data as the available functionalities. The solution required by the commissioner of this thesis presented a web-application built based on Java Spring Framework and Angular 8.0. A production-ready product required called ADP Connect was developed, tested and implemented in the requested timeframe. All the requirements for the project set initially were developed and implemented alongside with room for future development. The comissioner stated, that the final product provided good opportunities to decrease the time consumption related to client's information gathering and processing. However, in the period when this work was created the solution was not implemented in the production due to unknown reasons.

During the development process a certain skillset was gained by author of this work and understanding of the software creation process in the production environment was gained. Alongside with these, the informal comparison of different options for development was made by the author of this work on the basis of previous experience and the experience gained during this work's

implementation. All the work including the research, learning and implementation took about 350 hours in total.

However, the solution provided has certain areas which could be improved in the future and were not achieved due to the lack of time resources and lack of knowledge of the project's author. The security of the application could be done in a better manner from the backend viewpoint. Stronger assessment criteria could be implemented alongside with the enhanced protocols for authentication and authorization. For instance, filter chaining could be implemented in Spring Security in order to provide more tiers of access inspection and to make the assessment more modular. Also the user interface design could be more functional and advanced. The solution provided used a simple design without dozens of functional sections for the application. However, the design still fitted the commissioner's requirements and followed their common design patterns.

The design implementation during this thesis implementation was the most chalenging part. Due to the fact that the author's previous areas of responsibility mostly included working with the server-side components of the applications and websites, the implementation of the entire user interface, especially using the advanced Angular 8.0 framework, took the most time and management resources.

REFERENCES

ADP Inc. 2019a. Product home page. WWW page. Available at: https://www.adppayroll.com.au
[Accessed 22 August 2019]

ADP Inc. 2019b. Introduction to the end-user app authorization process. WWW document. Available at: https://developers.adp.com/articles/guides/authorization-process-end-user-apps
[Accessed 23 August 2019]

ADP Inc. 2019c. Introduction to data connector app authorization process. WWW document. Available at: https://developers.adp.com/articles/guides/auth-process-data-connector-apps
[Accessed 24 August 2019]

ADP Inc. 2019d. Workers v2 API docimentation. WWW document. Availabel at: https://developers.adp.com/articles/api/workers-v2-api
[Accessed 24 August 2019]

ADP Inc. 2019e. Developer Libraries. WWW document. Avaialble at: https://developers.adp.com/articles/guides/all/devlib/devlib
[Accessed 6 September 2019]

Apache Software Foundation. 2019a. Welcome Ant. WWW document. Available at: https://ant.apache.org/projects/index.html
[Accessed 3 August 2019]

Apache Software Foundation. 2019b. What is Maven? WWW document. Available at: https://maven.apache.org/what-is-maven.html
[Accessed 3 August 2019]

Apache Software Foundation. 2019c. Introduction to the standard directory layout. WWW document. Available at: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
[Accessed 14 August 2019]

Apache Software Foundation. 2019d. Java API to access Microsoft Excel format file. WWW document. Available at: https://poi.apache.org/components/spreadsheet/
[Accessed 9 September 2019]

AppDirect Inc. 2019. App distribution documentation. WWW document. Available at: https://help.appdirect.com/appdistrib/Default.htm#Dev-DistributionGuide/distrib-get-started.html%3FTocPath%3DGetting%2520started%7C_____0
[Accessed 23 August 2019]

Chacon, S., Straub, B. 2014. Pro Git. Ebook. Available at: https://git-scm.com/book/en/v2
[Accessed 18 August 2019]

Chenkie R. 2017. Angular authentication: Using route guards. Article. Available at: https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3
[Accessed 15 August 2019]

Continuous Delivery Foundation. 2019. Jenkins Documentation. WWW document. Available at: https://jenkins.io/doc/
[Accessed 19 August 2019]

Google LLC. 2019a. Angular architecture overview. WWW document. Available at: https://angular.io/guide/architecture
[Accessed 15 August 2019]

Google LLC. 2019b. Observables. WWW document. Available at: https://angular.io/guide/observables
[Accessed 16 August 2019]

Heller, M. 2018. Choosing your Java IDE. Article. Available at: https://www.javaworld.com/article/3114167/choosing-your-java-ide.html
[Accessed 31 July 2019]

JavaTPoint. 2018a. Spring Tutorial. WWW document. Available at: https://www.javatpoint.com/spring-tutorial
[Accessed 6 August 2019]

JavaTPoint. 2018b. Spring Boot. WWW document. Available at: https://www.javatpoint.com/spring-boot-introduction
[Accessed 7 August 2019]

JetBrains LLC. 2019. Making development an enjoyable experience. Product description. Available at: https://www.jetbrains.com/idea/features/
[Accessed 31 July 2019]

Kyrnin J. 2019. Understanding the index.html page on a website. Article. Avaialbel at: https://www.lifewire.com/index-html-page-3466505
[Accessed 15 August 2019]

Malkajgiri P. 2018. Workflow of Java code translation into bytecode. Image. Available at: https://qph.fs.quoracdn.net/main-qimg-e37d93ce61204fe7c8ef7e5999c0aaf6
[Accessed 10 June 2019]

Microsoft Inc. 2008. Office 2007 file format MIME types for HTTP content streaming. WWW document. Available at: https://blogs.msdn.microsoft.com/vsofficedeveloper/2008/05/08/office-2007-file-format-mime-types-for-http-content-streaming-2/
[Accessed 9 September 2019]

Mozilla Corporation. 2019a. CSRF. WWW document. Available at: https://developer.mozilla.org/en-US/docs/Glossary/CSRF
[Accessed 14 August 2019]

Mozilla Corporation. 2019b. CORS. WWW document. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[Accessed 14 August 2019]

Oauth Core Workgroup. 2007. Oauth Core 1.0. WWW document. Available at: https://oauth.net/core/1.0/
[Accessed 6 September 2019]

OpeinID Foundation. 2019. Welcome to OpenId Connect. WWW document. Available at: https://openid.net/connect/
[Accessed 23 August 2019]

Oracle Inc. 2019. Using JAR files: The Basics. WWW document. Available at: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html
[Accessed 2 September 2019]

Pivotal Software, Inc. 2019a. Spring Overview. WWW document. Available at: https://spring.io/projects/spring-framework
[Accessed 5 August 2019]

Pivotal Software, Inc. 2019b. Spring Security. WWW document. Available at: https://spring.io/projects/spring-security
[Accessed 9 August 2019]

Pivotal Software, Inc. 2019c. Securing a web application. WWW document. Available at: https://spring.io/guides/gs/securing-web/
[Accessed 9 August 2019]

Rouse M. 2019. Write once, run everywhere (WORA). WWW document. Availabel at: https://whatis.techtarget.com/definition/write-once-run-anywhere-WORA
[Accessed 10 June 2019]

Slack Technologies Limited. 2019. Slack websiet home page. WWW page. Available at: https://slack.com/intl/en-fi/
[Accessed 18 August 2019]

Tulika, D. 2017a. Overview of Java. WWW document.
Available at: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html#GUID-17B81887-C338-4489-924D-FDDF2468DEA7
[Accessed 10 June 2019]

Tulika, D. 2017b. Java and Object-Oriented Programming Terminology. WWW document.
Available at: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html#GUID-17B81887-C338-4489-924D-FDDF2468DEA7
[Accessed 10 June 2019]

Tyson M. 2018. What is JDK? WWW document. Avaialble at:
https://www.javaworld.com/article/3296360/what-is-the-jdk-introduction-to-the-java-development-kit.html
[Accessed 1 September 2019]

U.S. Department of Health & Human Services. 2019. About the Affordable Care Act. WWW document. Available at: https://www.hhs.gov/healthcare/index.html
[Accessed 10 June 2019]